WHITESTEIN Technologies

Living Systems® Process Suite

Business Activity Monitoring

Living Systems Process Suite Documentation

3.3
Mon Nov 1 2021

Whitestein Technologies AG | Hinterbergstrasse 20 | CH-6330 Cham
Tel +41 44-256-5000 | Fax +41 44-256-5001 | http://www.whitestein.com

Contents

Chapter 1

Model Monitoring

To monitor runtime data use the *Business Activity Monitor* library. The library resources contain dashboards that display widgets with Jasper Reports. The dashboards can be shared among users and users can set the parameters for parametric reports.

Though the library is based on the *forms* module, its widgets support as their content also forms designed with the *ui* module.

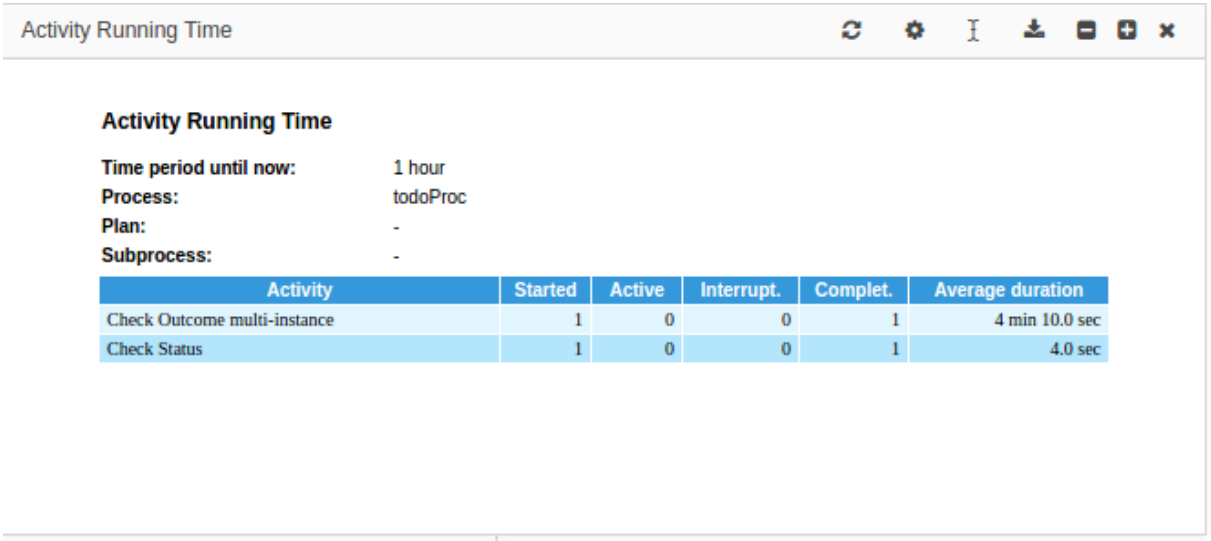
> **Important:** The BAM library makes use of Vaadin Charts which are developed by Vaadin: make sure to obtain the Vaadin Chart licenses before developing your own charts. Users of the your Application User Interface and Management Console do not require any additional licensing.

When creating data for monitoring, consider using `monitoring assignments and monitoring flags`.

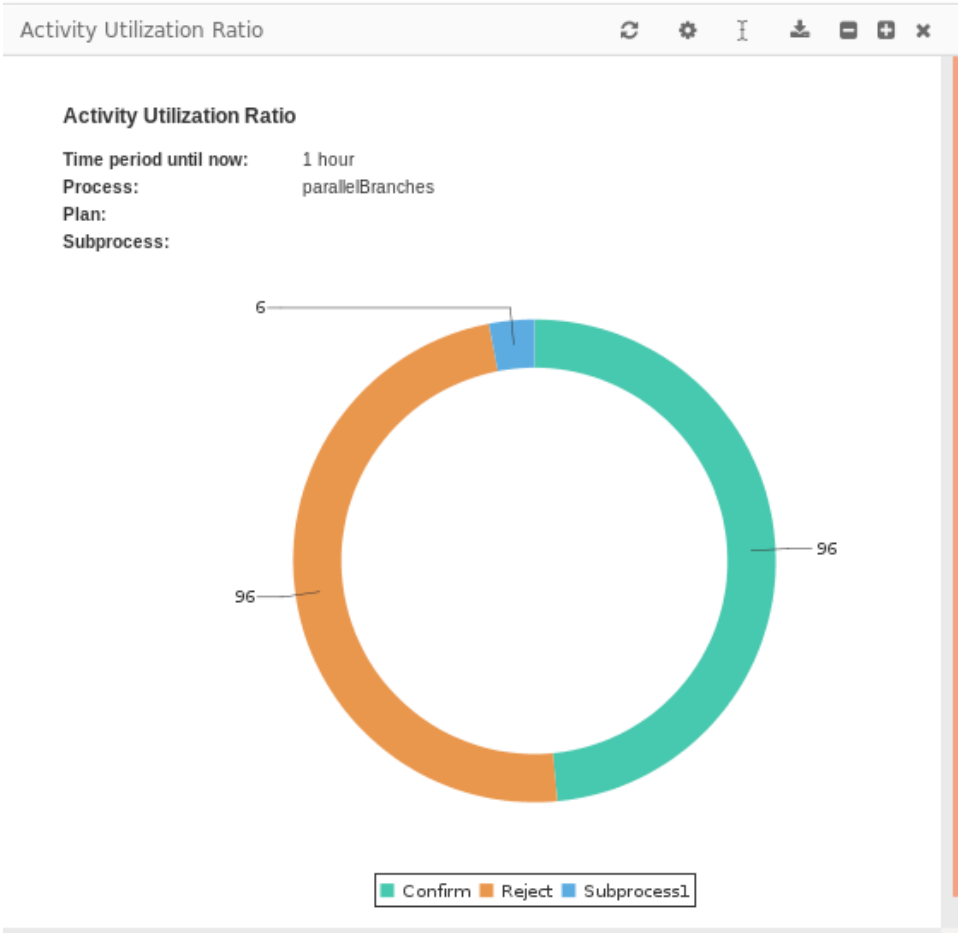
PDS comes with an example implementation of the Business Activity Monitor so you do not need to develop everything from the beginning. The example application comes with the following reports:

- **Activity Running Time**: how long it takes to complete a task (human or non-human)

  Only complete tasks are included: Instances of a looped task are considered part of a single task.



- **Activity Utilization Ratio**: how often an activity in a model was triggered in ratio to the number of the model's instances



- **Model Instances by Status**: bar chart with models indicating the status of their model instances

  Each bar represents all instances of a particular Model and is color-coded to indicate portions of instances in different statuses.

- **Person Utilization by Task**: instances of a task sorted for persons who completed (submitted) them
- **Plan Utilization Ratio**: number of times individual plans of a goal were used to achieve the selected goal
- **Process Running Time**: average running time of finished process instances of the selected processes (note that you can select multiple processes)
- **Process Running Time Distribution**: average running time of finished process instances of the selected processes for a defined period of time
- **Started Model Instances**: line chart with the number of instances of the selected models started during the selected period
- **Started Process Instances**: line chart with the number of instances of the selected Process started during the selected period
- **Users Activity Period**: bar chart with times of activity of the front-end application users

  The widget is only available if the `user tracking`.

- **Vaadin Chart Form**: example widget with a form as its content

Important: Some BAM reports rely on the data provided by the process logs: The data is available in the LSPS_PROCESS_LOGS table and the logging is governed by the CREATE_PROCESS_L↩ OG server setting. Generally, the process logging should be disabled in production. If you are using

BAM reports, consider setting CREATE_PROCESS_LOG to MODULE and disable the setting on all modules where it is not necessary to prevent possible performance issues.

1.1 Generating and Deploying Business Activity Monitor Example

The example *Business Activity Monitor* application is a document with dashboards that can hold widgets with Jasper Reports. The application is created with the *bamLibrary* Module.

To generate and deploy the default Business Activity Monitor, do the following:

1. Create a GO-BPMN Project that will hold the BAM Module.

   Make sure the project contains the BAM Library (to add the library, right-click the project and click **Add Library**, and use the *Select built-in library* option.)

2. Go to **File** -> **New** -> **Example**.
3. In the dialog box, select *BAM Dashboard* and click **Next**.
4. Select the target project and click **Finish**.

   The system generates the *bam* Module with the application resources with the following resources:

   - *bam.docs*: the document that creates the form with the dashboards
   - *Reports*: directory with resources of individual widgets:
     - Jasper Report displayed in the widget
     - form definition of the dialog where the user changes the report parameters
     - method definition with the methods for the widget including the *getParameters()* and methods that return parameters for the report
     - other resource, such as, localization, query definitions, etc.

5. Make sure your server connection is up and running.
6. Upload the *bam* Module (right-click the Module, click *Upload As*, and *Model*).
7. To access Business Activity Monitor, log in to the Application User Interface, go to **Documents** and open the **Business Activity Monitor** document.

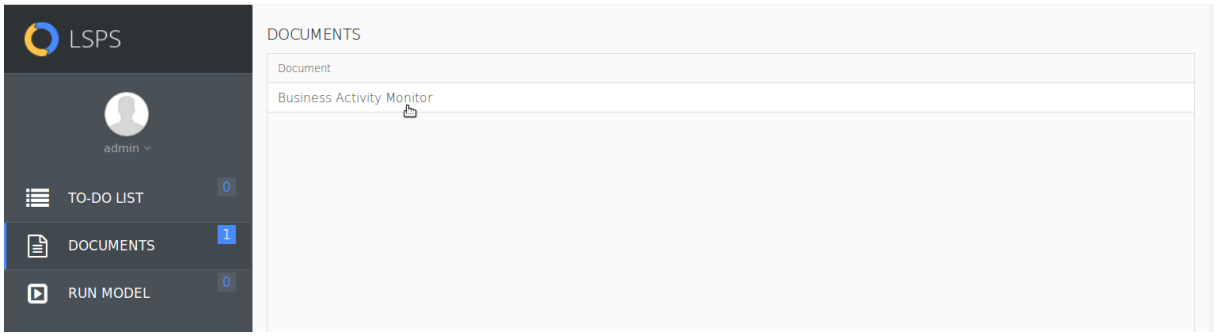


Figure 1.1 Opening BAM document

In the document, you can create your dashboards or display public dashboards with their widgets. Note that when you add a widget to your dashboard it remains empty: Define the report parameters to display data in a report.

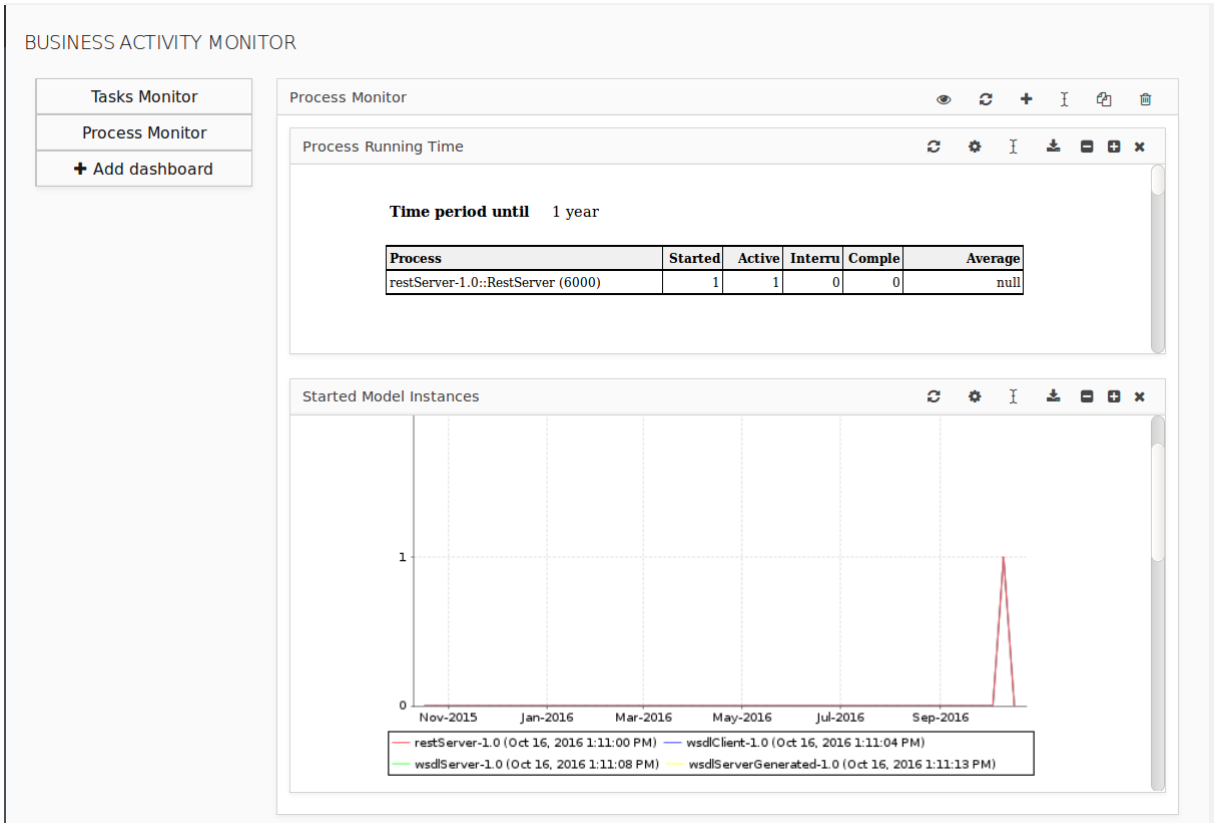


Figure 1.2 Business Activity Monitor with a private dashboard displayed

1.1.1 Defining a Default Dashboard

To define a default dashboard that will be available to every user, do the following:

1. Create an executable module that will import the `bam` module.
2. Create a resource that will create the default dashboard, for example, a process with the code in an assignment of a process element. As part of the code, you need to do the following:
   - Create or obtain the BAM users that should have the dashboard.
   - Create the dashboard as a `DashboardData` object with its widgets.
   - Add the dashboard to the list of the user's dashboards as a `DashboardOrder` object.

```
//The dashboard is made available to the current user:
def BAMUser bamUser := getOrCreateCurrentBAMUser();
//The dashboard instance with no widgets:
def DashboardData newDashboard :=
  new DashboardData(
    name -> "DefaultPublicDashboard",
    author -> bamUser,
    isPublic -> true
  );
newDashboard.widgets := [];
~
def TimePeriodParameter timePeriod :=  getParameterTimePeriod(1);
def Set<ModelParameter> allModels := findParameterModels().toSet();
def Set<Integer> modelIds := collect(allModels, {model:ModelParameter -> model.id});
```

```
~
def WidgetData startModelInstancesWidget := new WidgetData(
    name -> "Started Model Instances",
    moduleName -> "hll-reports",
    caption -> "Started Model Instances",
    x -> null,
    y -> null,
    height -> 4,
    width -> 3,
    state -> null,
    parameters -> [PARAM_TIME_PERIOD -> timePeriod.name,
            PARAM_TIME_PERIOD_ID -> timePeriod.id,
            PARAM_MODEL_IDS -> modelIds]
);
newDashboard.widgets := add(newDashboard.widgets, startModelInstancesWidget);
//Add the dashboard to the list of dashboards:
new DashboardOrder(dashboard -> newDashboard, user -> bamUser)
```

3. Run the module.

1.2 Customizing Business Activity Monitor

To create a custom Business Activity Monitor, we recommend to start from the example Business Activity Monitor application. However, you can create your Business Activity Monitor application from scratch as well.

1.2.1 Adding a Custom Widget with a Jasper Report

> **Important:** If you are modifying or reusing the example Jasper Reports or creating a new report that will use a template of styles (in the example application it is `LspsStyle.jbtx` template), do not use Jaspersoft® Studio 6.3.1 since it remains hanging due to an issue when loading the common style template.

To add a custom widget that will display a Jasper Report to the Business Activity Monitor, do the following:

1. In the Reports folder of the *bam* module, create a folder for your widget.

   Alternatively create the widget resources in a separate Module and import it into the *bam Module*.

2. Copy the Jasper Report into the folder and refresh the folder (select the folder in GO-BPMN Explorer and press F5).

3. Create a form that will allow the user to define the report parameters:

   - Typically you will use input components, such as Combo Box, Input Field, etc. Make sure to define IDs for the input components, so you can acquire their value with the *getParameters()* method.
   - Create a local variable of the ReportFrame type: the variable will hold reference to the ReportFrame with the report when changing parameters.
   - Create a component that will apply the parameters on the ReportFrame and refresh it, typically you will use a button with the action defined as its click action:

     ```
     { _ ->
       reportFrame.setParameters(getParameters());
       reportFrame.refresh()
     }
     ```

4. In the methods file of your parameter Form, define the following methods:

- *constructor with ReportFrame* as its input parameters

```
public MyWidget(ReportFrame reportFrame) {
  this.reportFrame := reportFrame
}
```

- *getParameters()* method, that will return the new parameters: The report parameters are defined as a map of parameter names and their values.

```
private Map<String, Object> getParameters() {
  //load the value from the input component into a local variable:
  def String newParameterValue := myInputField.getValue();
  //return the values mapped to their report keys:
  [
   "jasperReportParam" -> newParameterValue
  ]
}
```

- *showReport()* method that will display and update the report.

```
private void showReport() {
  Forms.navigateToUrl(embeddedJasperReportUrl(thisModel().name,
  "Reports/myReport.jrxml", getParameters()))
}
```

5. In the *bam.docs*, add the widget to the *BamDocument* UIDefinition:

```
new BamApplicationForm(new BamConfiguration(widgets ->[
        new WidgetDefinition(
        name -> "My Widget",
        moduleName -> thisModel().name,
        reportWidget -> new ReportWidget (designPath -> "Reports/MyWidget/myReport.jrxml"),
        parametersForm -> {reportFrame:ReportFrame -> new MyWidget(reportFrame)},
        defaultParameters -> [->],
        width -> 3,
        height -> null
    ),
    ...
```

1.2.2 Adding a Custom Widget with a Form

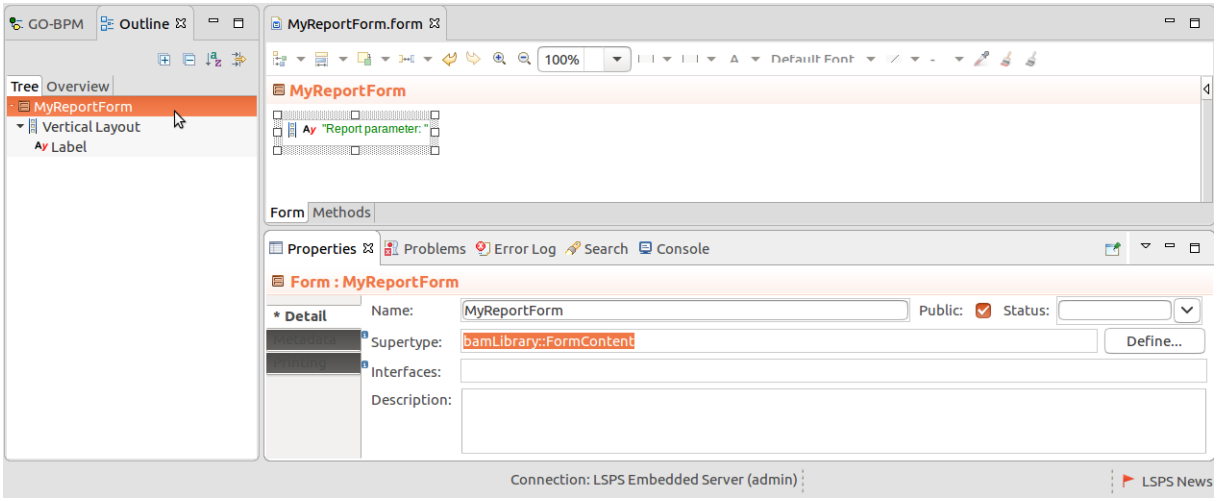
To add a custom widget that will display a Form definition as its content, do the following:

1. In the Reports folder of your *bam* module, create a folder for your widget.

   Alternatively you can create the widget resources in a separate Module and import the Module into the *bam Module*.

2. In the folder, create the form definition that will be used as the content of the widget.

3. Set the supertype of the form to `bamLibrary::FormContent`.



4. Add the widget to the Bam document:

   (a) In GO-BPMN Explorer, select the *bam.docs* definition.

   (b) In the document editor, select *BamDocument*.

   (c) In the UI definition property, add the WidgetDefinition with the content of the form widget: `form↩ Widget -> new FormWidget(reportForm -> { widgetData:WidgetData -> new <MYFORM>() })`

```
new WidgetDefinition(
    name -> #"Simple Widget with a Form",
    moduleName -> thisModel().name,
    formWidget -> new FormWidget(reportForm -> { widgetData:WidgetData -> new WidgetForm()})
)
```

1.2.2.1 Sending Parameters to a Form of a Custom Widget

To define the default parameters and parameter-input form for a widget with a form, do the following:

1. Define your default parameters of the WidgetDefinition as a map and make the form take the default values of widget parameters. Alternatively, pass the WidgetData object as its input parameter: the object holds the parameter values for the widget.

```
new WidgetDefinition(
  name -> #"Simple Widget with a Form",
  moduleName -> thisModel().name,
  defaultParameters -> ["widgetParameter" -> "default param"],
  formWidget ->
     new FormWidget(
       reportForm -> { widgetData:WidgetData -> new EkoForm(defaultParameterValue)}
     )
  )
)
```

   (a) Create a parametric constructor with the parameters and set them as value on the respective components

```
public EkoForm(String defaultParameterValue){
   label.setValue(defaultParameterValue);
}
```

   (b) Implement the `refresh()` method on the form that will refresh any components that depend on the parameters.

```
public void refresh(WidgetDefinition widgetDefinition, WidgetData widgetData, Popup param
  this.widgetData := widgetData;
  if label != null then
     label.refresh();
  end
  }
```

2. Create the form for parameter setting:

   (a) Design the parameter form with input components for parameters.

   (b) Create a local variable of type ReportFrame: the variable will hold the ReportFrame object passed from the WidgetDefinition object.

   (c) Create a parametric constructor with a ReportFrame argument and store it in a local variable.

   ```
   public ParameterInputForm(ReportFrame rf){
      reportFrame := rf;
   }
   ```

   The ReportFrame will serve to apply parameter values in the widget. Hence make sure that the parameter values are reflected back in the *parameters* map of the ReportFrame.

   ```
   reportFrame.widgetData.parameters := ["widgetParameter" -> myParameterValue]
   ```

   (d) Add the `refresh()` call on the report frame from your form to apply the parameter values and close the popup with the parameter form.

   (e) In the *UIDefinition* property of the bam Document, define the *parametersForm* property of your Widget↩ Definition: note that this is a closure with the ReportFrame of the WidgetDefinition as its input argument. Create the parameter form with the report frame as its parameter. (The report frame holds all the data for your widget instance).

   ```
   new WidgetDefinition(
     name -> #"Simple Widget with a Form",
     moduleName -> thisModel().name,
     parametersForm -> {reportFrame:ReportFrame -> new ParameterForm(reportFrame)},
     defaultParameters -> ["widgetParameter" -> "default param"],
     formWidget ->
       new FormWidget(reportForm ->
         { widgetData:WidgetData -> new EkoForm(widgetData)}
       )
     )
   ```

1.3 Creating Business Activity Monitor from Scratch

To create a *Business Activity Monitor* from scratch only using the *bamLibrary* Module, you will need to implement the widgets and then create a document with the Bam application:

1. Import the *bamLibrary* Module into your Module (Right-click Module > Module Imports; in the dialog box, click Add and double-click *bamLibrary*).

2. Create the widgets:

   (a) Copy the JasperReports for your widgets into the Module.

   (b) Create the parameter form that will allow you to change the parameters of the report:

      i. Create a Form definition.

      ii. Insert components that will allow the user to define new parameter values. Make sure to define IDs on the component so you can acquire their values when setting new report values in the `get↩ Parameter()` method later.

      iii. Create a local form variable of type ReportFrame: the ReportFrame object will allow passing of parameters from the form to the report.

iv. Declare the constructor of the Form that will take the ReportFrame object as its member variable.
`public MyParamForm(ReportFrame reportFrame) {this.reportFrame := reportFrame}`

v. Create the `getParameter()` method for the parameter form: it must return a Map of the parameters and their new values.

```
public Map<String, Object> getParameters(){
   def String newParameterValue := MyParamInput.getValue();
   [ "myParam" -> newParameterValue ]
}
```

(c) Create a component, such as a Button, that will send the new parameters to the ReportFrame and refresh the report:

i. In the parameter form, insert a component that will send the data.

ii. Define the send logic and refresh of the report, for example, you can create a Button with a Click Listener:

```
{ e ->
  if reportFrame == null then
    Forms.navigateToUrl(
      embeddedJasperReportUrl(
        thisModel().name,
        "myReport.jrxml",
        getParameters()
     )
    )
  else
      //set the new report parameters from the ReportFrame:
    reportFrame.setParameters(getParameters());
    //Display the report:
    reportFrame.refresh()
  end
}
```

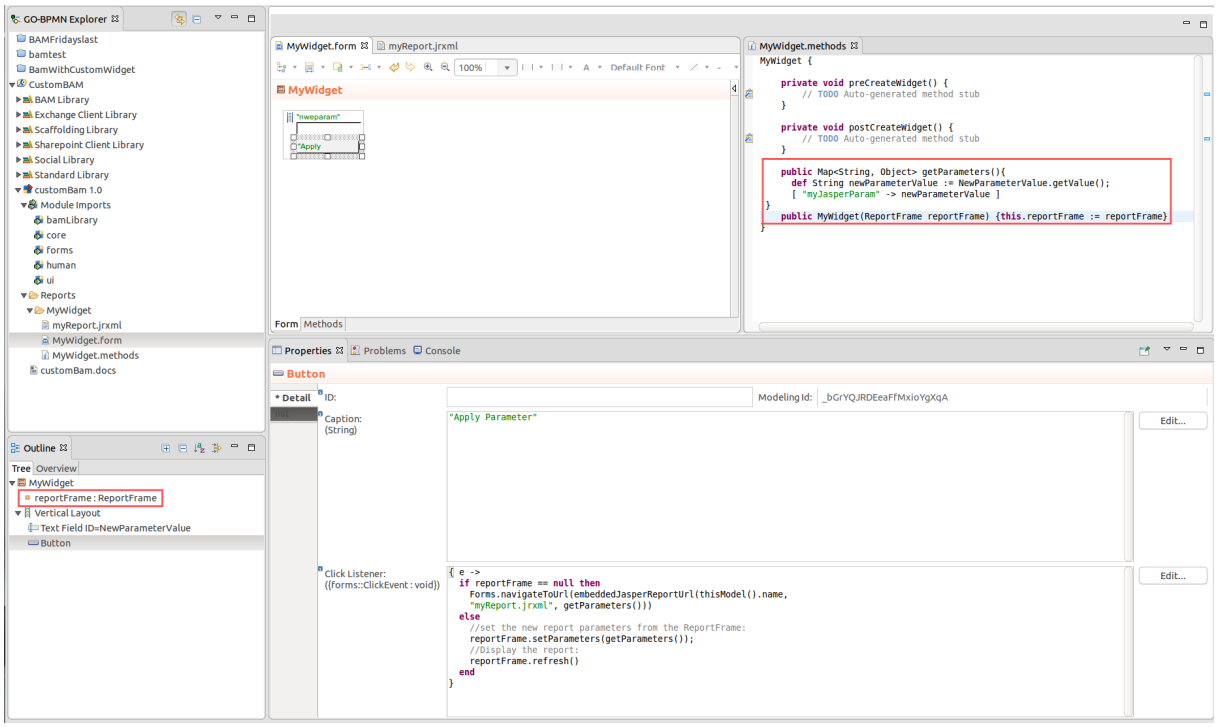


Figure 1.3 Parameter widget definition with the ReportFrame local variable and added methods highlighted; also not the Apply button Click expression in the Properties view

3. Create the BAM application:

(a) Create a document: this will be your BAM document.

(b) Define the properties of the Document and define the UI definition as the `BamApplicationForm()` with a BamConfiguration as its parameter.

- BamConfiguration holds a list of WidgetDefinitions of widgets available for the Dashboards of the BamApplicationForm;
- WidgetDefinitions define the available Widgets and their properties:
  - name: name of the Widget
  - module name: name of the parent Module
  - designPath: path to your report relative to the module
  - parametersForms: form that is used to collect parameters for the report via ReportFrame (if undefined, no param settings available)
  - defaultParameters - default parameters for the report
  - width: default width of the widget
  - height: default height of the widget

```
new BamApplicationForm(
  new BamConfiguration(
    widgets -> [
      new WidgetDefinition(
        name -> "My BAM Dashboard Widget",
        moduleName -> thisModel().name,
        //We pass the ReportFrame with data about the Widget to the parameter form:
        parametersForm -> {
          reportFrame:ReportFrame -> new MyParameterForm(reportFrame)
        },
        defaultParameters -> [
          "myWidgetParam" -> "Default parameter value for my report."
        ],
        designPath -> "myJasperReport.jrxml",
        width -> 4,
        height -> 4
      )
    ]
  )
)
```

You can now deploy the Module with your custom BAM to the LSPS Server.

1.4 Configuring the Data Source for Reports

By default, reports use data from the LSPS_DS data source. To use data from another data source, do the following:

1. Create the data source to your database on your server with LSPS: refer to the documentation of your application server.

   Example datasource configuration on WildFly

```
<xa-datasource jndi-name="java:/jdbc/REPORT_DS" pool-name="REPORT_DS" enabled="true" use-java
  <xa-datasource-property name="URL">
    <!--For WF 10 and newer: -->
    jdbc:h2:tcp://localhost/./h2/reports;MVCC=TRUE;LOCK_TIMEOUT=60000
    <!--for previous WF: jdbc:h2:tcp://localhost/./h2/reports;MVCC=TRUE;LOCK_TIMEOUT=60000-->
  </xa-datasource-property>
  <driver>h2</driver>
```

```
    <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
    <xa-pool>
      <min-pool-size>10</min-pool-size>
      <max-pool-size>20</max-pool-size>
      <prefill>true</prefill>
    </xa-pool>
    <security>
      <user-name>lsps</user-name>
      <password>lsps</password>
    </security>
  </xa-datasource>
```

2. In your reports, set the report property `com.whitestein.lsps.monitoring.datasource-jndi-name` to the data source, in the example above, `REPORT_DS`:
   (a) In Jasper Studio, open your report.
   (b) In the report Outline view, click the root component.
   (c) In the *Properties* view of the root component, click the Advanced tab.
   (d) Under the *Misc* node next to the Properties item, click `...`.
   (e) In the popup, enter the property name `com.whitestein.lsps.monitoring.datasource-jndi-name` and the value with the JNDI name of the data source, in the example, `java:/jdbc/REPORT_DS`.
3. Design your report.

1.5 Working with the Front End

1.5.1 Creating Dashboards

Dashboards are spaces that display widgets with Jasper Reports or forms. By default, a dashboard belongs to the user who created the dashboard and cannot be accessed by other users unless the dashboard is marked as *public*.

To create a new dashboard, do the following:

1. Click **Add dashboard**.
2. In the dialog on the *New dashboard* tab, define the dashboard name.
3. Select the *Public* checkbox if applicable.
4. Click **Add**.

The dashboard appears in the list of dashboards above the **Add dashboard** button.

1.5.1.1 Publishing and Unpublishing Dashboards

To change the setting of the *public* flag on your Dashboard, click the respective button in the Dashboard caption.

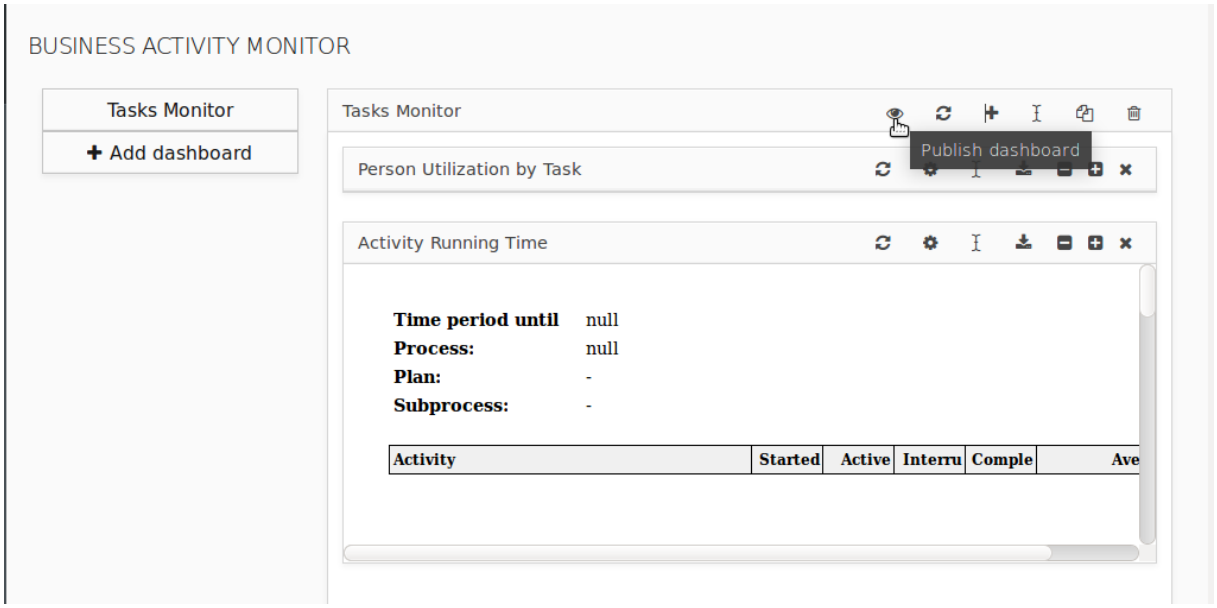


Figure 1.4 Publishing a private dashboard

1.5.1.1.1 Adding Public Dashboards

To add a public dashboard created by another user to your BAM, do the following:

1. Click the **Add dashboard** button.
2. In the popup, select the *Public dashboard* tab and select the dashboard in the combo box.
3. Click **Add** to confirm.

The public dashboard appears in the list of dashboards above the **Add dashboard** button. Note that public dashboards created by other users are read only.

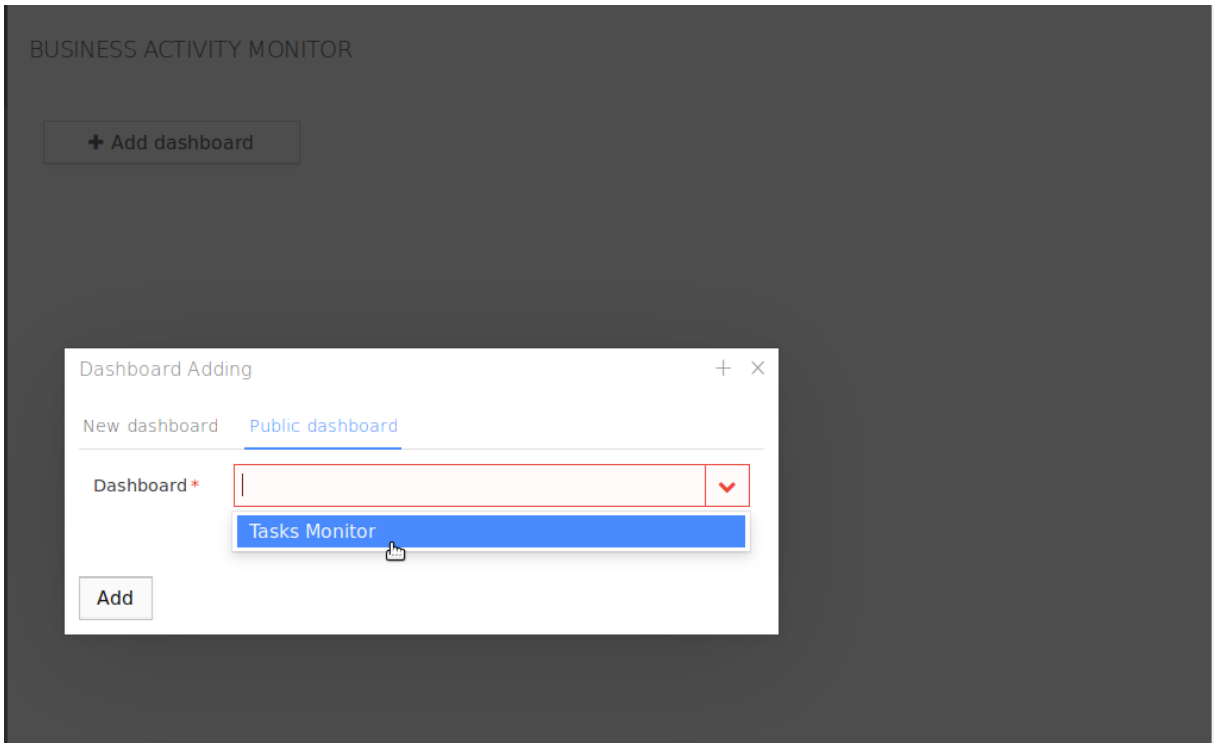


Figure 1.5 Adding a public dashboard

1.5.1.2 Copying Dashboard

Since the user can edit only the dashboard *they* created, it is convenient to be able to copy dashboards.

To create a copy of a dashboard, click the *Copy* button in the caption of the dashboard.

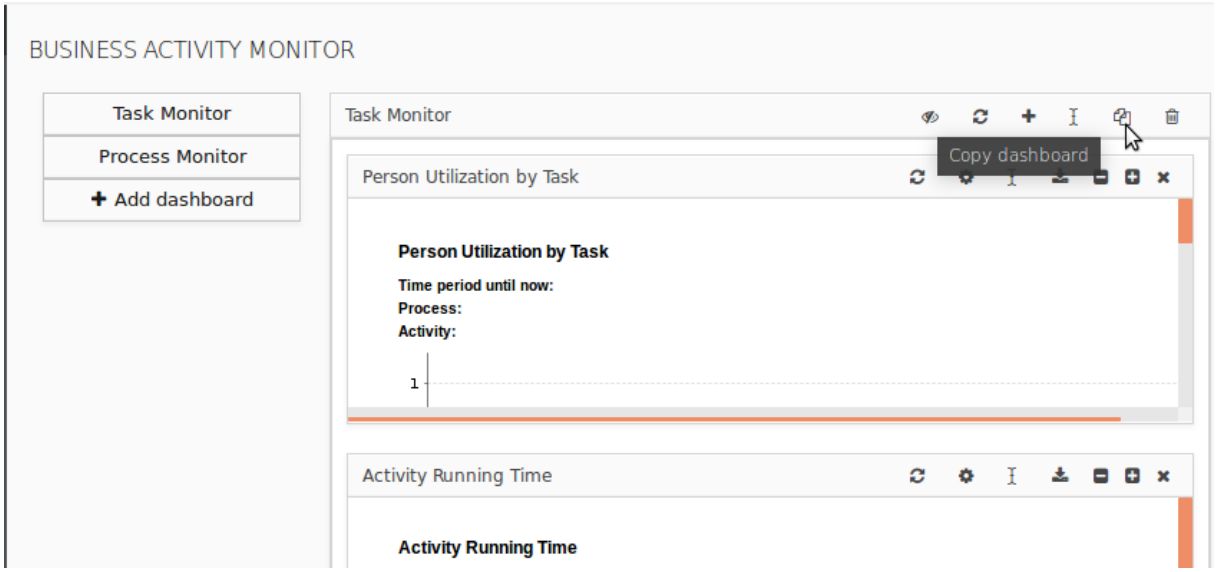


Figure 1.6 Copying dashboard

1.5.1.3 Renaming Dashboards

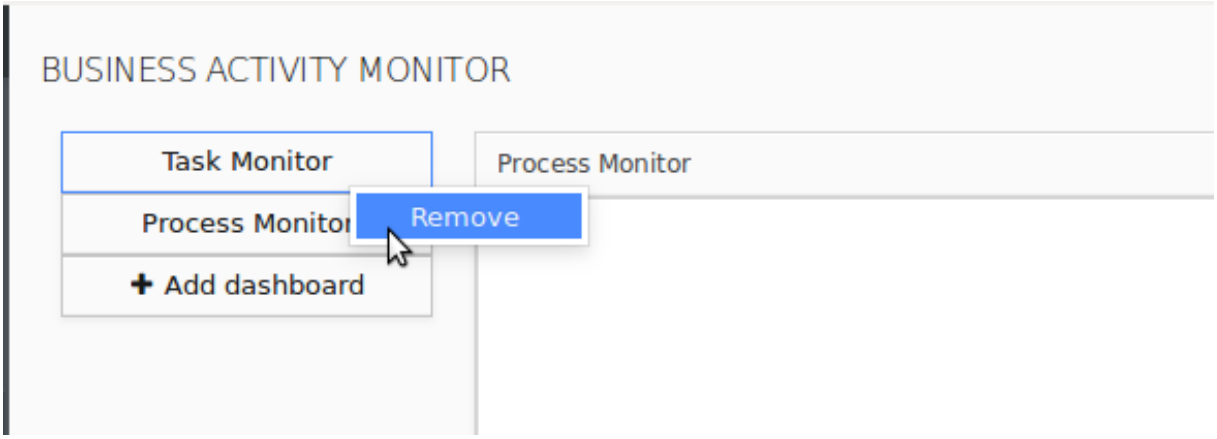
To change the caption of your dashboard, click the Rename button () in the dashboard caption.

Note that you cannot edit public dashboards.

1.5.1.4 Deleting Dashboards

To delete a dashboard, do one of the following:

- Right-click the dashboard button in the Business Activity Monitor document and click *Remove* in the context menu.



- Click the *Delete* button in the dashboard caption.

Note: When you delete your public dashboard, the dashboard will no longer be available to any users.

1.5.2 Displaying Widgets

BAM widgets are components with Jasper Reports that are displayed on a dashboard. Widgets can take parameters that are then used by the Jasper Reports.

To display a widget on a dashboard, do the following:

1. Open the dashboard.
2. In the caption of the dashboard, click the **Add** (+) button in the caption of the dashboard.
3. In the dialog box, select the widget and click **Add widget**.

On your dashboards, you can drag-and-drop the widgets and resize them as required. On public dashboards, the positions and dimensions are fixed.

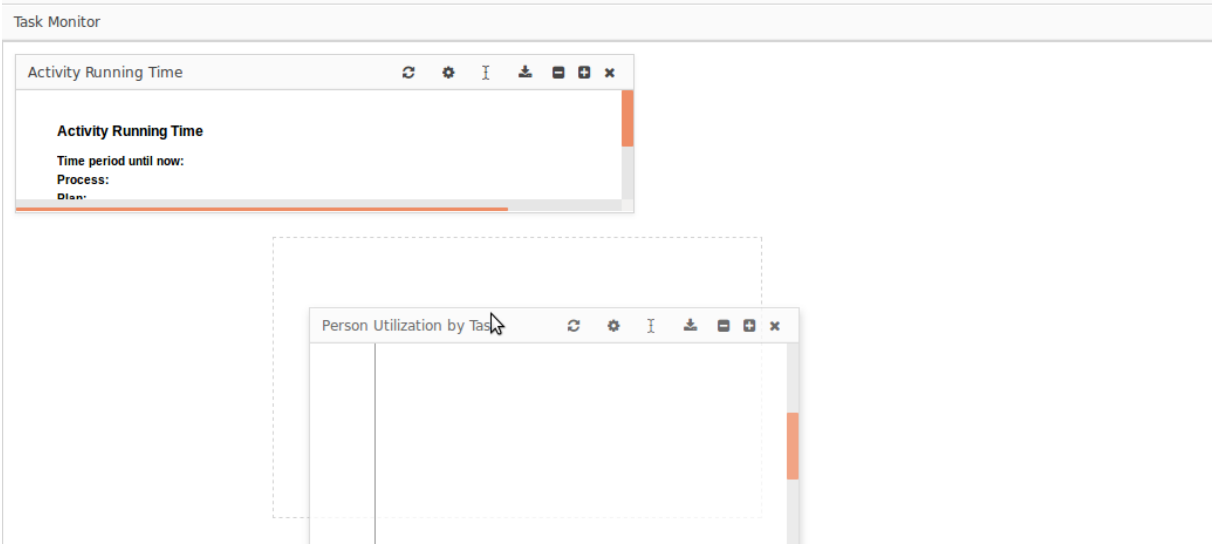


Figure 1.7 Dragging a widget

1.5.2.1 Changing Widget Parameters

To change parameters used by a widget, click the *Settings* button in the widget caption and set the parameters in the popup box. If the report does not require any parameters, the button is not available.

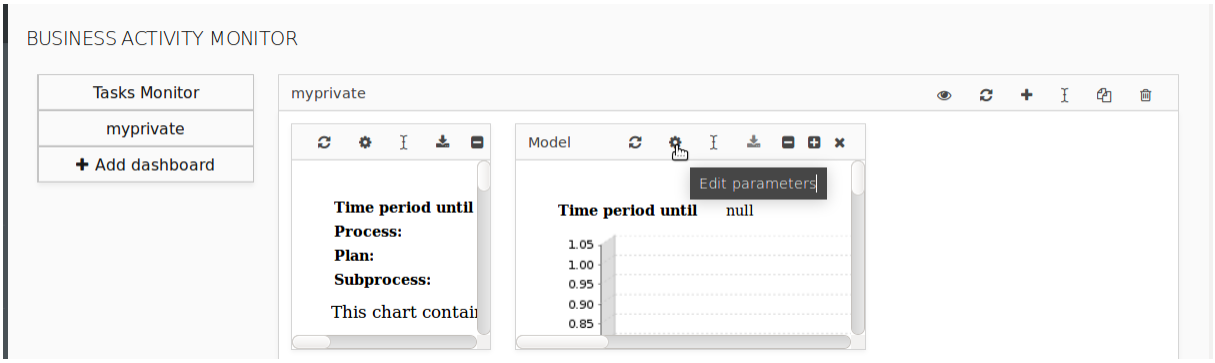


Figure 1.8 Edit parameters button in the widget caption

1.5.2.2 Renaming Widgets

To change the caption of a widget displayed on your dashboard, click the Rename button () in the widget caption.

Note that you cannot edit public dashboards.

1.5.2.3 Exporting Reports to PDF, Word, or Excel

To export the report displayed in a widget, click the export button  in the caption of the widget and select the target format.