



Advanced Integration

EXERCISES

Table of Contents

1. Deploy a simple server	1
1.1. Prerequisites	1
1.2. Objectives	1
1.3. Instructions	1
1.4. Correction	1
1.4.1. Download the bundle from the download page in the customer portal.	1
1.4.2. Extract the bundle.....	1
1.4.3. Install the license file.	2
1.4.4. First start.	3
1.4.5. Check your installation	5
1.4.6. Update configuration / license.....	11
1.4.7. Tips.	12
2. Connect to bonita bpm engine with java	14
2.1. Objective.....	14
2.2. Instructions	14
2.3. Correction	14
2.3.1. Create a maven project on eclipse	14
2.3.2. Create the java class.....	19
2.3.3. Execute the project.....	21
3. Using Bonita REST API	22
3.1. Prerequisites	22
3.2. Objectives.....	22
3.3. Instructions	22
3.4. Correction	22
3.4.1. Download the REST client	22
3.4.2. Connect To The Engine	24
3.4.3. Create a New User	25
3.4.4. Get a List of Users.....	27
3.4.5. Get the List of Installed Processes	28
3.4.6. Create a Case	28
3.4.7. Get the List of Pending Tasks.....	29
3.4.8. Set Case Variable Value	30
3.4.9. Logout	31
4. Vacation Request Use case.....	32
4.1. Use case description	32
4.2. Use case implementation	32
4.3. Preliminary work.....	32
4.3.1. Create a new repository	32

4.3.2. Deploy existing processes	33
4.3.3. Check your BDM data	35
5. Create a page for the Employees	37
5.1. Objectives	37
5.1.1. Overview	37
5.2. Instructions	38
5.3. Correction	38
5.3.1. Create a new empty page	38
5.3.2. Create the data	39
5.3.3. Build your page	41
5.3.4. Page mock-up	41
5.3.5. Choosing and placing your widgets	41
5.3.6. Container widget	43
5.3.7. Link widget as a button	45
5.3.8. Preview on different devices	46
5.3.9. Export your page	48
5.3.10. Create a new application and test your page	48
5.4. Optional exercises	53
5.4.1. Display modify and cancel buttons	53
5.4.2. Display background-color	55
5.4.3. Display a warning text	57
6. Create a new REST API Extention	59
6.1. Prerequisites	59
6.2. Objectives	59
6.3. Instructions	59
6.4. Correction	59
6.4.1. Create new REST API extension	59
6.4.2. Configure the new project	60
6.4.3. Configuration advanced	61
6.4.4. Configuration parameters	62
6.4.5. Screen start	63
6.4.6. Implement the business logic	64
6.4.7. Implement the test	66
6.4.8. Deploy	68
6.4.9. Execute	69
6.4.10. View results on a Page	73
7. Create a custom page for the Managers	77
7.1. Prerequisites	77
7.2. Objectives	77
7.2.1. Overview	77
7.3. Instructions	78

7.4. Correction	78
7.4.1. Create a new empty page	79
7.4.2. Rest API Extension	79
7.4.3. Create the data	80
7.4.4. Build your page.....	81
7.4.5. Preview on different devices.....	86
7.4.6. Export your page	86
7.4.7. Add your page to the Vacation Management application	86
7.5. Optional exercises	89
7.5.1. Display a bar chart.....	89
7.5.2. Display a warning text	90
8. Create a new connector	93
8.1. Objective.....	93
8.2. Prerequisites	93
8.3. Instructions	93
8.4. Correction	93
8.4.1. Download the Development Connector Toolkit from customer portal.	93
8.4.2. Create the definition	95
8.4.3. Create the implementation	98
8.4.4. Generate the java project	99
8.4.5. Launch Eclipse and import the maven project.....	100
8.4.6. Complete your connector implementation.....	102
8.4.7. Generate the connector	103
8.4.8. Import the zip file generated in the previous step in Bonita studio	103
8.4.9. Import "UserCreationDiagram.bos" from provided_files folder.....	104
8.4.10. Test the process	106
9. Create a new Custom Widget	107
9.1. Prerequisites	107
9.2. Objectives.....	107
9.3. Instructions	107
9.3.1. Correction	107
9.3.2. Create the custom widget.....	107
9.3.3. Access the page	107
9.3.4. Properties.....	108
9.3.5. HTML.....	108
9.3.6. Controller	109
9.3.7. How to test	110

Chapter 1. Deploy a simple server

1.1. Prerequisites

You need JAVA SDK 1.7 installed, and JAVA_HOME defined on it.

You must have Java ready for execution.

In command line try "java -version". You should read an answer similar to:

```
java version "1.7.0_80"
Java(TM) SE Runtime Environment (build 1.7.0_80-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.80-b11, mixed mode)
```

1.2. Objectives

The goal of this exercise is to deploy a server for testing and prototypes environments, to quickly and easily deploy Portal and generated applications, including pre-configured application server.

1.3. Instructions

- Download the Tomcat bundle from the download page in the customer portal. (Link provided by email or your trainer)
- Extract the Tomcat bundle.
- Install the license file.
- First start.
- Update configuration and restart.

1.4. Correction

1.4.1. Download the bundle from the download page in the customer portal.

To get the download page, access the customer portal, menu "Download", or ask to your trainer.

- In the section "Deploying Server Components"
- Choose your component, for this exercise we will take the **Tomcat bundle** (i.e BonitaBPMSubscription-<version>-Tomcat-<version>.zip)

1.4.2. Extract the bundle.

Just extract the archive in a folder without spaces or special characters (no accent).

Content of the Tomcat bundle

The Tomcat bundle is based on a standard Tomcat installation with the following additions:

bin/setenv.bat: script to configure JVM system properties for Windows.
bin/setenv.sh: script to configure JVM system properties for Linux.
bonita-start.bat: script to start the bundle on Windows.
bonita-start.sh: script to start the bundle on Linux.
bonita-stop.bat: script to stop the bundle on Windows.
bonita-stop.sh: script to stop the bundle on Linux.
conf/Catalina/localhost/bonita.xml: Tomcat context configuration for Bonita web application. Define data sources used by Bonita Engine.
conf/bitronix-*.properties: configuration files for Bitronix
catalina.properties: modified to include lib/bonita folder to Tomcat classpath.
context.xml: modified to add JTA support using Bitronix library.
logging.properties: modified to create a log file dedicated to Bonita.
server.xml: modified to add listener for Bitronix and h2 (see below for modification needed if you want to switch to another RDBMS).
lib/bonita: extra libraries needed by Bonita. The following libraries are included:
Bitronix JTA Transaction Manager, h2, SLF4J (required by Bitronix).
request_key_utils: folder containing script to generate license request keys (Subscription editions only).
webapps/bonita.war: the Bonita web application.

- Note the path to the extracted folder as "**TOMCAT_HOME**"

1.4.3. Install the license file.

A server needs a license to run, exactly like your Bonita Studio.



If the version (ex 7.x) is the same between the studio and your server, and if they are on the same machine, you can copy the studio license in **TOMCAT_HOME/setup/platform_conf/licenses/**.

Otherwise:

- Go in **TOMCAT_HOME/request_key_utils/**
- Execute the good version of the generator according to your OS and license type (ex : windows, development generateRequestKey.bat)
- Select option 2 - CPU core license
- Select option 1 - Development CPU core license (2 cores allowed).
- Copy the key (everything including the parenthesis)
- Generate a licence from your key (In training just send the key to your trainer)
- Copy the file of the licence in **TOMCAT_HOME/setup/platform_conf/licenses/**. This licence is a **Development licence**, and then you need to follow the Customize chapter.

1.4.4. First start.

Customize startup script

For a server with a **development licence**, you should know that the licence limits the server to only two CPU, so we need to say to the server to use only two CPU. if you copy the licence from the studio, you use this type of licence.

- Go in **TOMCAT_HOME**/
- For Windows:
 - Copy the file bonita-start.bat to a file called **bonita-start_2cores.bat**
 - Change the last line of the file to:

```
start /AFFINITY 3 bonita-start.bat
```

Where 3 is the affinity mask expressed as a hexadecimal number

- For Linux:
 - Copy the file bonita-start.sh to a file called **bonita-start_2cores.sh**.
 - Change the last line of the file to:

```
taskset -c 0,1 bonita-start.sh 0,1
```

Performance Edition specification

If you are installing the Performance Subscription edition, you need to edit some file. Under the **TOMCAT_HOME**, you have a directory named "setup" : this directory is the working folder of the Platform Setup tool. This tool allow you to create bonita database and to manage Bonita configuration stored in that database. (more on this in Update Configuration chapter)

- Edit **setup/platform_conf/initial/platform_init_engine/bonita-platform-init-community-custom.properties**
- Change the value of the activeProfiles key to '**community,performance**'. No change is needed for the Community, Teamwork, or Efficiency edition. Result is

```
activeProfiles=community,performance
```

Run it.

Start the server:

- Open a shell (Linux) / cmd (Windows) prompt and go to **TOMCAT_HOME**

- Run bonita-start_2cores.bat (or .sh) if you specify a 2 core startup, or run bonita-start.bat (or .sh) from command line
- The console finish with the message : "INFO: Server startup in xxxx ms"

```

mager has finished in 238 ms
2016-04-11 15:46:23.268 -0700 org.apache.catalina.startup.HostConfig org.apache.catalina.startup.HostConfig deployDirectory
INFO: Déploiement du répertoire C:\atelier\Tomcat 7.2.2\webapps\ROOT de l'application web
2016-04-11 15:46:23.433 -0700 org.apache.catalina.startup.TldConfig org.apache.catalina.startup.TldConfig execute
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARS that were scanned but no TLDs were found in them. Skipping unneeded JARS during scanning can improve startup time and JSP compilation time.
2016-04-11 15:46:23.442 -0700 org.apache.catalina.startup.HostConfig org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deployment of web application directory C:\atelier\Tomcat 7.2.2\webapps\RO
OT has finished in 174 ms
2016-04-11 15:46:23.448 -0700 org.apache.coyote.http11.Http11Protocol org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
2016-04-11 15:46:23.478 -0700 org.apache.coyote.ajp.AjpProtocol org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
2016-04-11 15:46:23.483 -0700 org.apache.catalina.startup.Catalina org.apache.ca
talina.startup.Catalina start
INFO: Server startup in 89992 ms

```

Figure 1. Server is started

- You can open your browser at the address : <http://localhost:8080/bonita/>

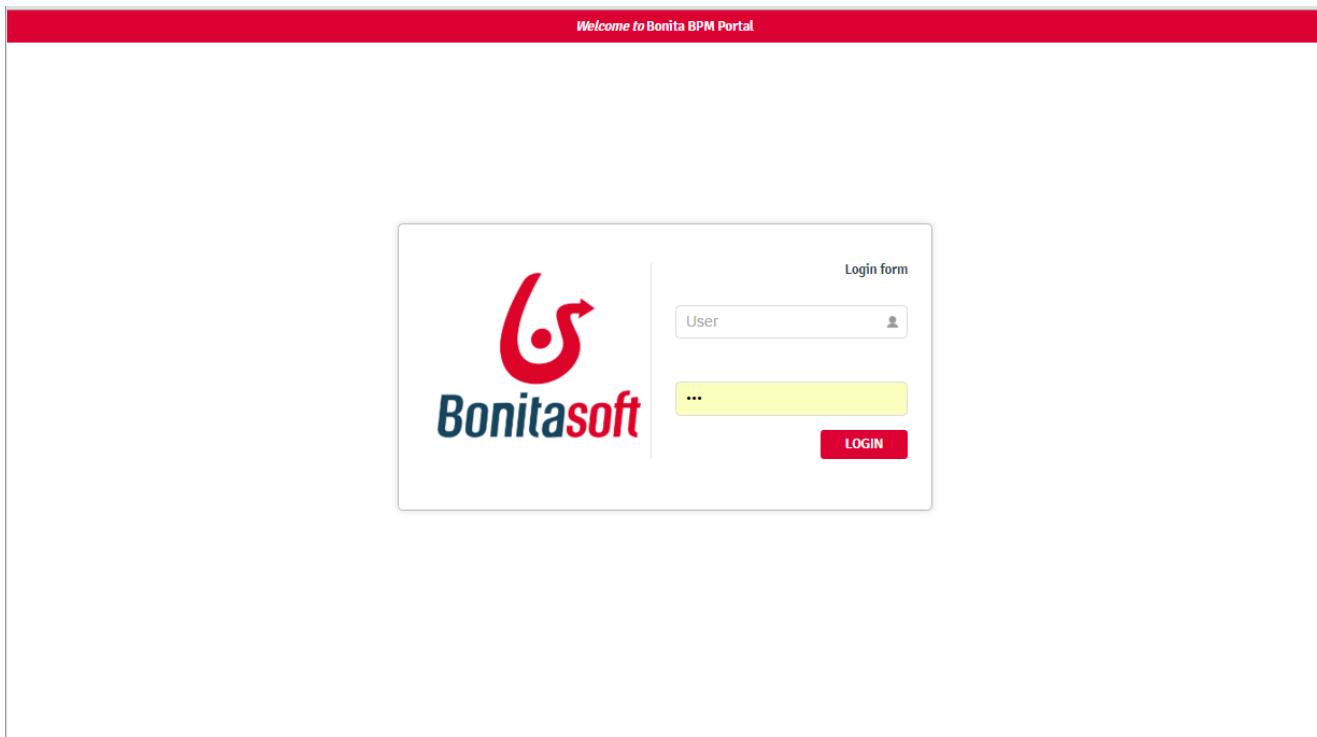


Figure 2. Home page

- Use the default tech user log : **install**, password : **install**

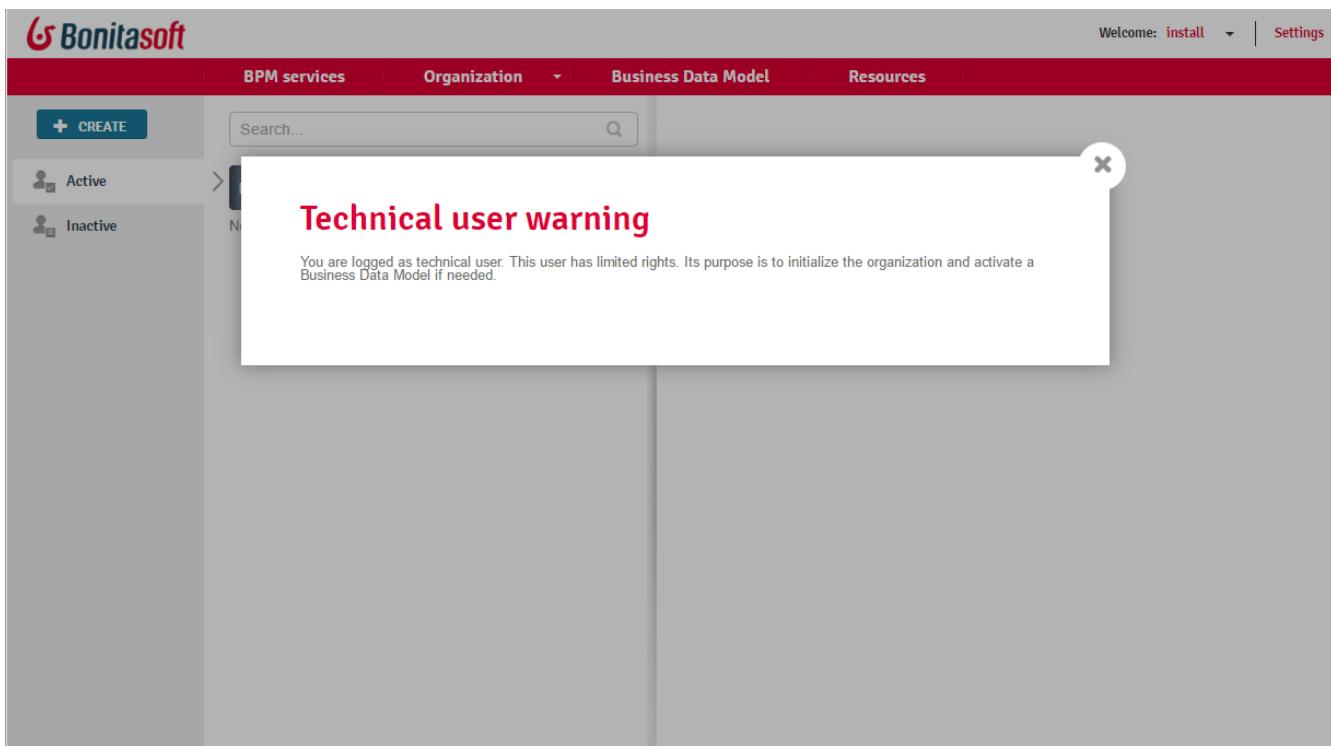


Figure 3. User install is logged

Troubleshooting

If something is not working properly you can check in **TOMCAT_HOME/logs/** for errors. One file name bonita.<Date of the day>.log and catalina.<Date of the day>.log Mains errors:

```
java.net.BindException: Address already in use: JVM_Bind <null>:8080
```

The server try to start on the port 8080, and this port is already used. Maybe the studio is started ? The Studio use the port 8080.

```
org.bonitasoft.engine.exception.CreationException: The licence is not valid: License Error 51  
Unexpected error(s). It might be due an environment issue.  
License Error: No license file found.
```

You forgot to set the license, or the license is not correct.

1.4.5. Check your installation

Prepare organisation and process bar from the studio

- Close the server and open the Studio (they share ports and cannot run at the same time without changing configuration settings)
- Extract the organization from your studio (menu Organization/export)

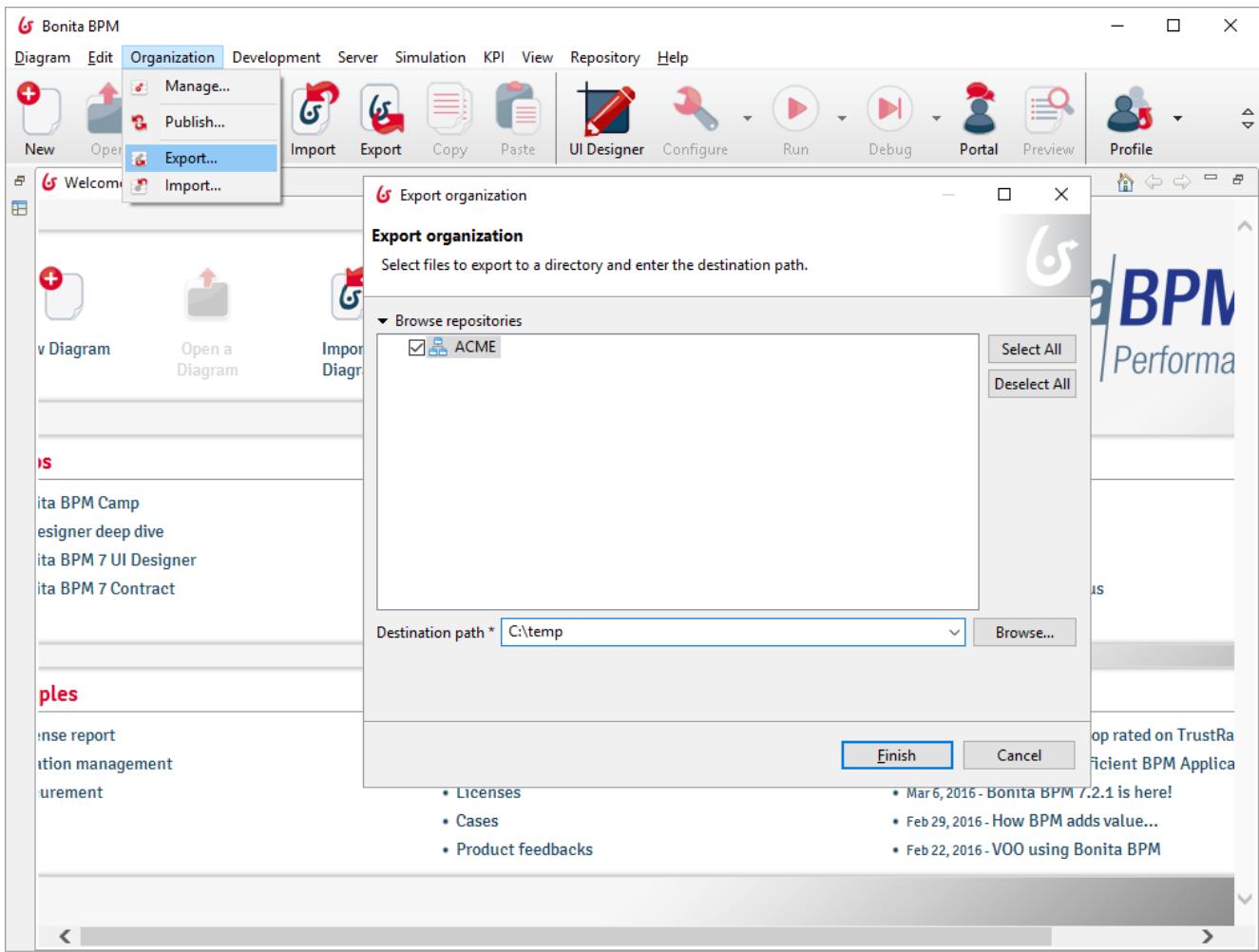


Figure 4. Export the organization

- Open one process developed with the studio and generate a bar file

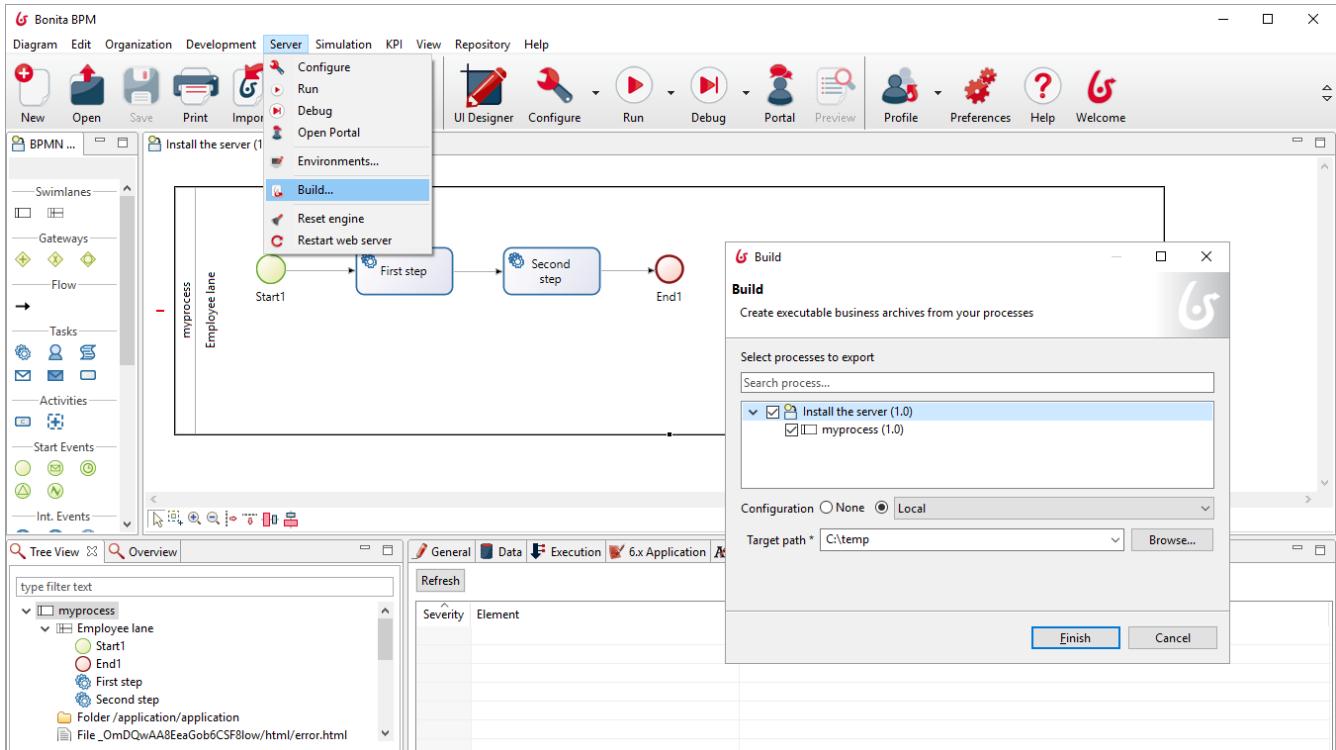


Figure 5. Export a simple process

Note : in this process, there are no human task and no overview, then no form is included.

- Now you can close the studio and start again the server
- Install the organisation
- log as **Install / install**
- From the portal go to the menu **Organization/Import-Export** and import the file extracted from the Studio

The screenshot shows the Bonitasoft Portal homepage. At the top, there is a navigation bar with the Bonitasoft logo, a user dropdown set to "Install", and a "Settings" link. Below the header, there is a sidebar titled "BPM services" featuring a green circular icon with two gears. The main content area has tabs for "Organization", "Business Data Model", and "Resources". The "Organization" tab is active and displays sub-options: "Users", "Groups", "Roles", and "Import / Export". The "Import / Export" option is highlighted with a red background. A note below it states: "You need to pause the BPM services to update to your environment that cannot be done while there are active users. When the services are paused:

- o Only the technical user can log in.
- o Users who are currently logged in, including Administrator users, are automatically logged out.
- o Users who are filling in forms when the services are paused will lose any information that has not been submitted.
- o All processes are automatically paused.

When you resume the services, tell your users that they can log back in to the Portal." A "PAUSE" button is visible at the bottom of this section. At the very bottom of the page, there is a URL bar showing "localhost:8080/bonita/portal/homepage#".

Figure 6. Load the organization

All users must be visible

Welcome: install | Settings

BPM services Organization Business Data Model Resources

CREATE

Active Inactive

First name | Last name | Last login

Giovanna Almeida
Account manager

Daniela Angelo
Vice President of Sales

Walter Bates
Human resources benefits

Isabel Bleasdale
Product marketing manager

Jan Fisher
Infrastructure specialist

Patrick Gardenier
Financial controller

Thorsten Hartmann
Financial planning manager

Joseph Hovell
Engineer

Mrs Giovanna Almeida

Email: giovanna.almeida@acme.com
Manager: Daniela Angelo
Username: giovanna.almeida
Last login: No data

Last update: 04/11/2016 4:43 PM

Profile
No data

Membership
member of latin_america

1 of 1

Figure 7. All users are loaded

- Associate User/Administrator profiles to Users. Go to Organization / Profiles,

Welcome: install | Settings

BPM services Organization Business Data Model Resources

ADD

All

Name

Administrator
The administrator can install a process, manage the organization, and...

Process manager
The Process manager can supervise designated processes, and mana...

User
The user can view and perform tasks and can start a new case of a pr...

EXPORT Updated on

Administrator

The administrator can install a process, manage the organization, and handle some errors (for example, by replaying a task).

Created on: 04/11/2016 3:52 PM
Updated on: 04/11/2016 3:52 PM
Created by: System
Updated by: System

Technical details

Users: 0
Groups: 0
Roles: 0
Memberships: 0

Figure 8. Profiles

Select the profile **Users**, click on **MORE**, and add all users (click on **ADD A USER**)

The screenshot shows the Bonitasoft BPM interface with the 'Business Data Model' tab selected. In the top right corner, there are 'Welcome: install' and 'Settings' buttons. Below the header, there's a 'User' section with a brief description: 'The user can view and perform tasks and can start a new case of a process.' To the right of this description are three timestamped boxes: 'Created on: 04/11/2016 3:52 PM', 'Created by: System', and 'Updated on: 04/11/2016 4:47 PM'. Below these is another box labeled 'Updated by: System'. Under the 'User' section is a 'Users mapping' table listing seven members: William Jobs, Walter Bates, Patrick Gardenier, Virginie Jomphe, Thorsten Hartmann, Jan Fisher, and Isabel Bleasdale, each with a 'remove' action link. To the right is a 'Groups mapping' section with a 'No data' message and a 'ADD A GROUP' button.

Figure 9. Users in the profile

- Go to profile Adminstator, and reference minimum **Walter.Bates** in this profile

The screenshot shows the Bonitasoft BPM interface with the 'Business Data Model' tab selected. In the top right corner, there are 'Welcome: install' and 'Settings' buttons. Below the header, there's an 'Administrator' section with a brief description: 'The administrator can install a process, manage the organization, and handle some errors (for example, by replaying a task.)'. To the right of this description are three timestamped boxes: 'Created on: 04/11/2016 3:52 PM', 'Created by: System', and 'Updated on: 04/11/2016 4:50 PM'. Below these is another box labeled 'Updated by: System'. Under the 'Administrator' section is a 'Users mapping' table listing one member: Walter Bates, with a 'remove' action link. To the right is a 'Groups mapping' section with a 'No data' message and a 'ADD A GROUP' button. At the bottom left of the page is a 'Roles mapping' section with a 'No data' message and a 'ADD A ROLE' button. At the bottom right is a 'Memberships mapping' section with a 'No data' message and a 'ADD A MEMBERSHIP' button.

Figure 10. Users in the profile

- Logout and Login with **walter.bates**/ password **bpm**
- Install one process
- Choose the Administrator profile,

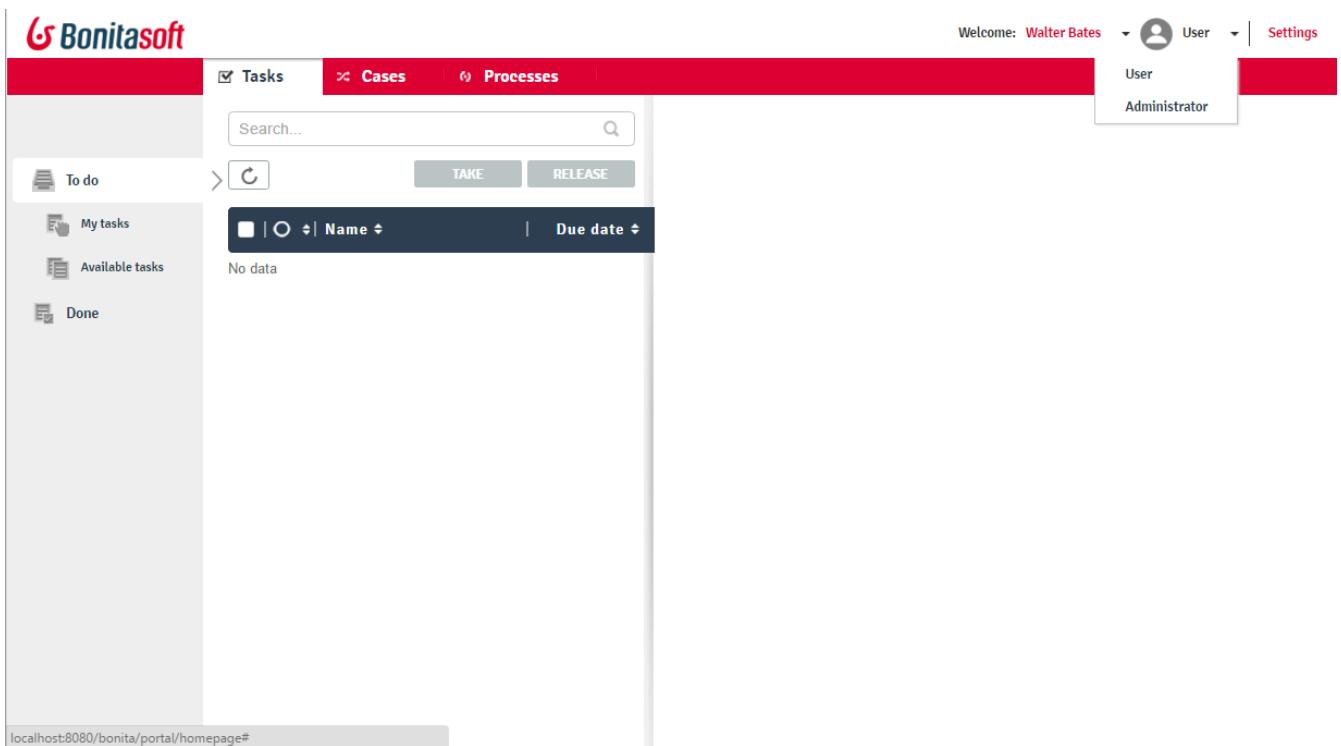


Figure 11. Choose the administrator profile

- Select **BPM/ Processes** tab and click on **INSTALL**

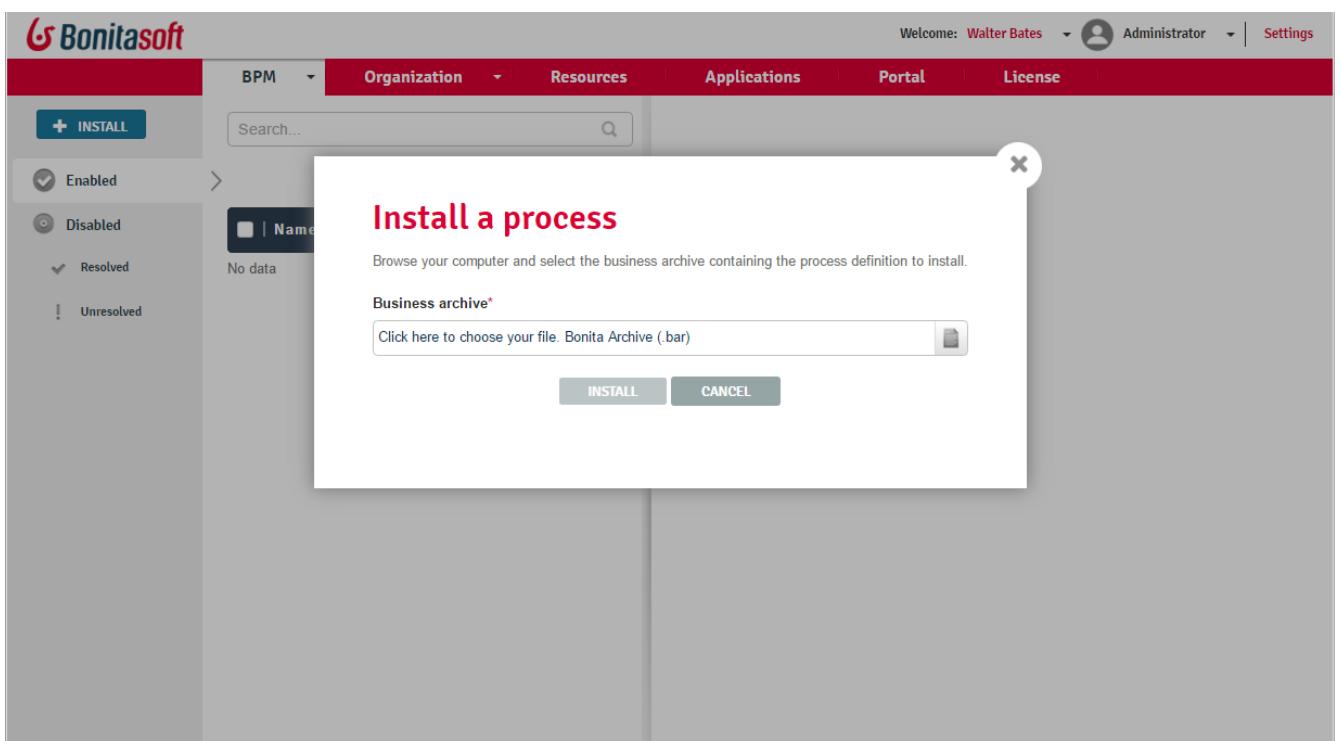


Figure 12. Install a process

- If needed resolve process actor mapping issues. Don't forget to ENABLE it

Welcome: Walter Bates | Administrator | Settings

BACK START FOR DELETE

myprocess (1.0)

General

- Actors
- Parameters
- Connectors
- Forms
- Scripts

Configuration state RESOLVED Activation state **ENABLED** Updated on 04/11/2016 4:56 PM

No description.

Categories

Monitoring status

Cases with failures	Healthy cases	Open cases	View diagram
0	0	0	

Process Manager mapping

Select the entities (users, groups, roles, memberships) who will supervise this process.

Users	Roles	Groups	Memberships

Check that each entity has the relevant Process Manager profile in the Portal. To do so, go to [Profiles](#).

Figure 13. Check the process/ ENABLE it

- Choose the User process, go to Process Run the process and test it

Welcome: Walter Bates | User | Settings

Tasks Cases Processes

Search...

All > Name Version

Name	Version
myprocess	1.0

1 of 1

myprocess (1.0)

No description.

My cases
No data

Cases I worked on
No data

Archives
No data

START

Figure 14. Start a process

1.4.6. Update configuration / license

The Platform setup tool handles the creation of the database schema and the configuration of Bonita BPM Platform.

In Tomcat bundle you can find the tool in the setup folder

It is composed of the following items:

- platform_conf/
- initial/: contains the default configuration of the Bonita BPM Platform, that can be customized and will be pushed when the database is created.
- current/: will contain configuration files after a pull from the database is made.
- licenses/: (Subscriptions only) ⇒ must contain the license file to allow Bonita BPM Platform to start without error.
- sql/: SQL scripts that are used to create the Bonita BPM database tables
- setup.sh: Unix / Mac script to run.
- setup.bat: Windows script to run.
- database.properties: contains properties to connect to the database on which the configuration is managed.

To modify the configuration of an already initialized Bonita BPM Platform, you must use the Platform setup tool as follows:

- Stop Bonita BPM Platform.
- Make sure the database.properties of the Platform setup tool points to the database used by Bonita BPM Platform.
- Run setup.sh pull or setup.bat pull. It will get the current configuration and put it in the platform_conf/current folder.
- Modify the configuration files inside the platform_conf/current folder according to your needs.
- If you are updating your license file, put it inside platform_conf/licenses, along with the existing ones. If some retrieved license files are not valid anymore, you can remove them, to delete them from the database when pushed.
- Run setup.sh push or setup.bat push.
- Start Bonita BPM Platform.

1.4.7. Tips.

- Change the port Run the server and the studio at the same time You can run the server and the studio at the same time if you change the all port in server.xml and in the configuration database. See the Bonitasoft documentation to change that.
- Passwords of technical user As part of the configuration managed by the Platform setup tool; the technical user, install, password install, can be changed. See <http://documentation.bonitasoft.com/bonitabpm/?page=first-steps-after-setup> for more information
- H2 Database access You can consult the data in the BDM by the overview of the case. By default, the server use a H2 database, exactly like the studio. In the folder "TOMCAT_HOME\lib\bonita", you can execute the h2 driver.

Under windows if the JAVA_HOME is correct, just double click on the driver (ex : h2-1.3.170.jar).

Else execute

```
java -jar h2-1.3.170.jar".
```

A page is visible in a browser:

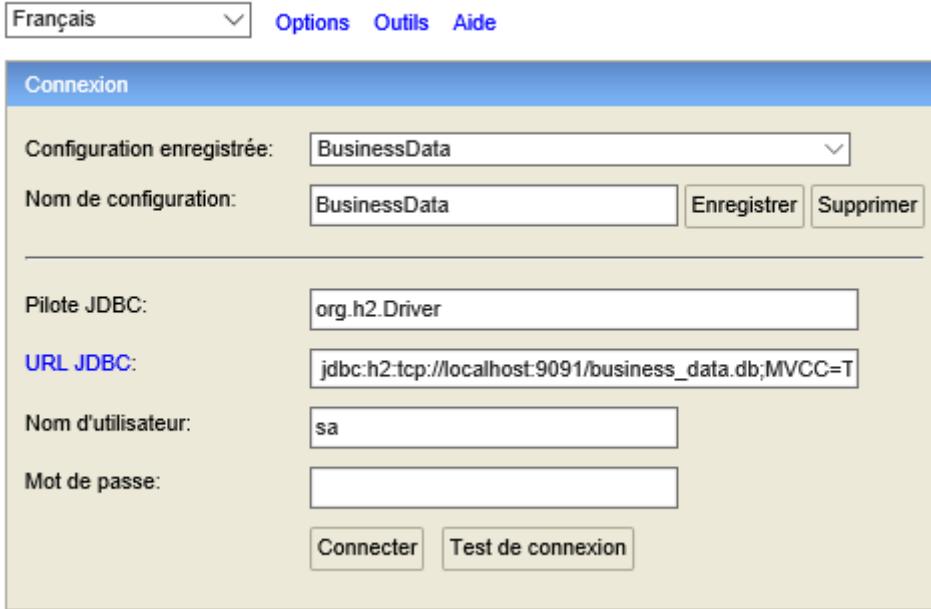


Figure 15. H2 connection

You need to change the URL JDBC by "jdbc:h2:tcp://localhost:9091/business_data.db"

Now you can see directly your data for development.

The screenshot shows the H2 database management interface. At the top, there is a toolbar with various icons. The left sidebar lists databases ('business'), schemas ('INFORMATION_SCHEMA'), and users ('Utilisateurs'). A message at the top right says 'H2 1.3.170 (2012-11-30)'. The main area has a title bar 'Validation automatique (auto commit)' with a checkbox, 'Max lignes: 1000', and other settings. Below the title bar, there are buttons for 'Exécuter (Ctrl+Enter)', 'Effacer', and 'Instruction SQL'. The main content area is currently empty. Below the main area, there is a section titled 'Commandes principales' (Main commands) with a list of options: 'Afficher cette page d'aide', 'Afficher l'historique des commandes', 'Exécuter la commande courante', and 'Déconnexion de la base de données'. There is also a section titled 'Exemple de script SQL' (SQL script example) with a list of commands like 'Effacer une table si elle existe', 'Créer une nouvelle table', etc. At the bottom, there is a section titled 'Ajouter des pilotes de base de données' (Add database drivers) with a note about adding additional drivers.

Figure 16. Display the page

Chapter 2. Connect to bonita bpm engine with java

2.1. Objective

The goal of this exercise is to create a java client and use Bonita BPM API to connect with the Bonita BPM Engine on a separate server and get the list of active users.

2.2. Instructions

- Create a java project.
- Import the Bonita BPM java dependencies.
- Create a java class with a main method where you will:
- Connect to the Bonita BPM Engine.
- Retrieve the list of active users.

2.3. Correction

2.3.1. Create a maven project on eclipse

the correction is based on Eclipse IDE, feel free to use the IDE that you prefer.

1. In Eclipse click on "New → Project → Other" and choose "Maven Project".

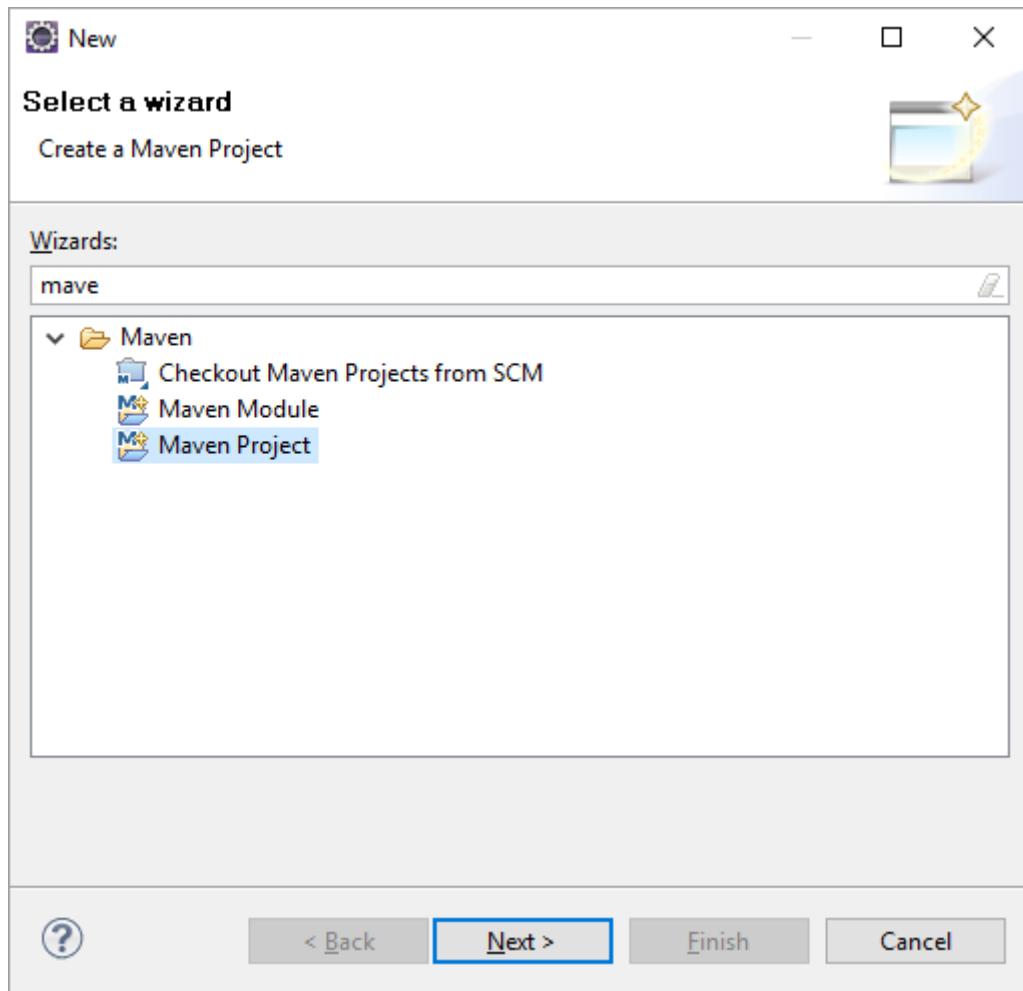


Figure 17. Create the project

1. Select the "Create a simple project (skip archetype selection)"

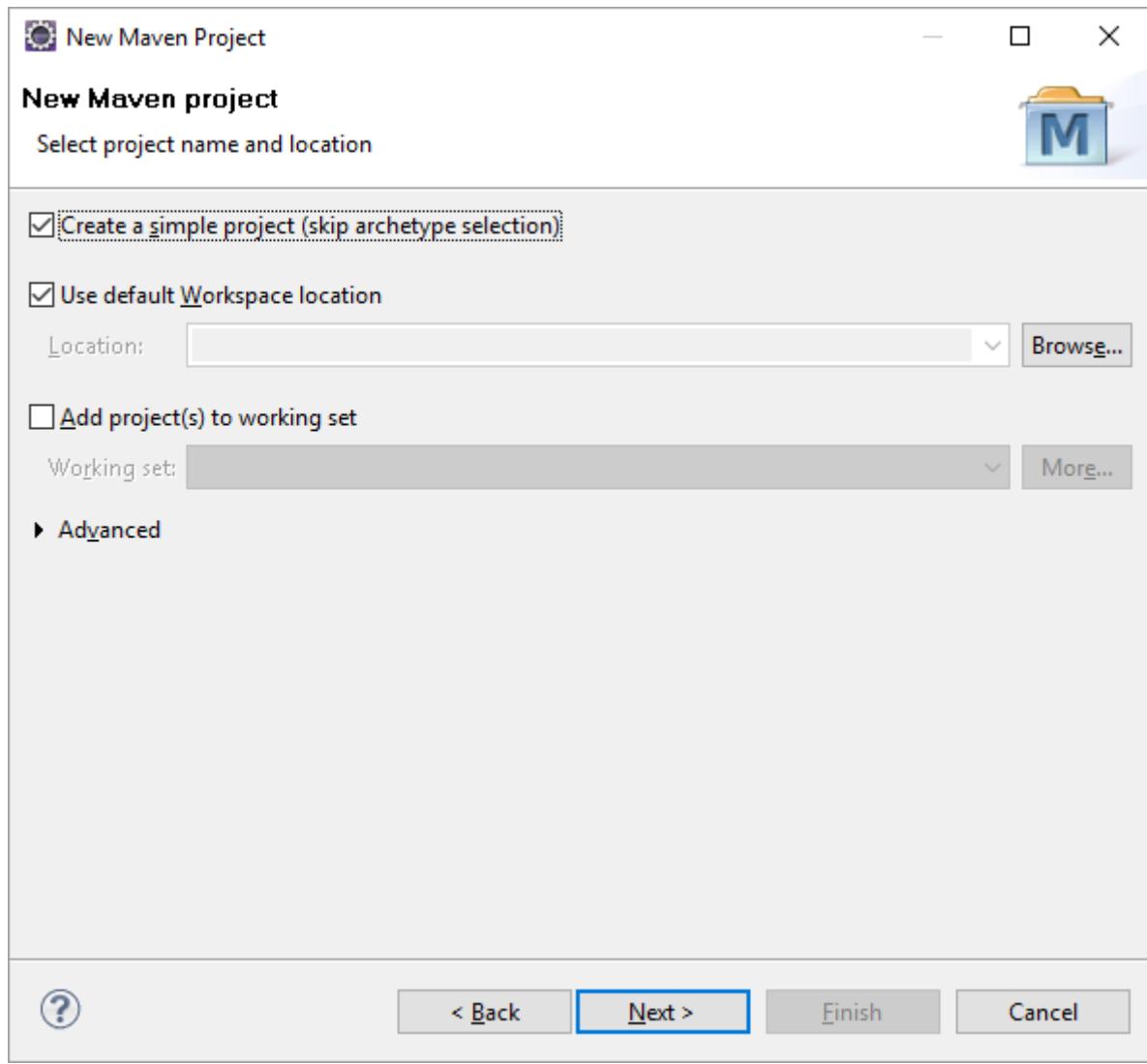


Figure 18. Default Archetype selection

1. Give "org.bonitasoft.training" as groupId, and "bonitaTestApi" as artifactId, click on finish.

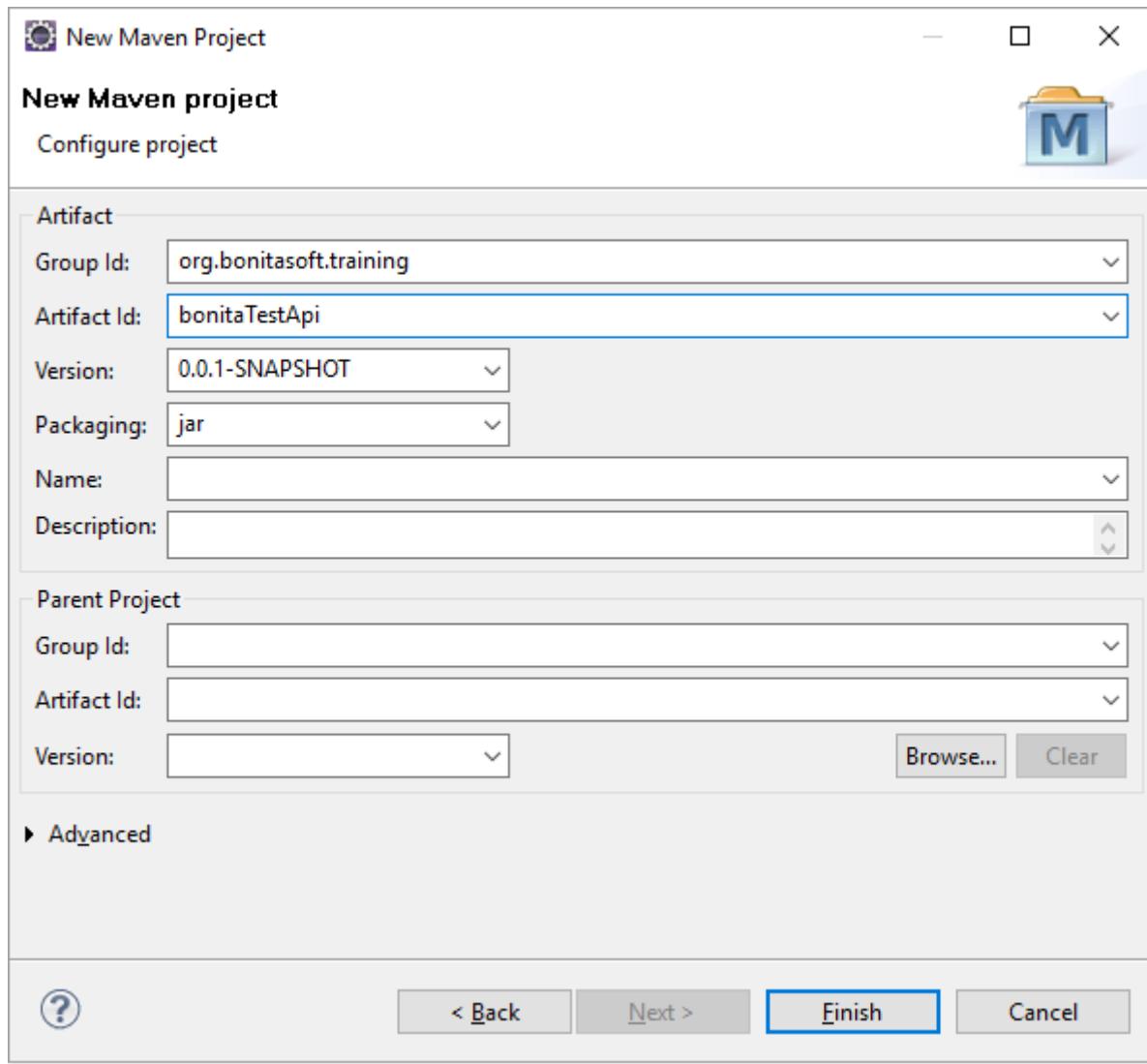


Figure 19. All parameters

- Update the pom.xml, insert the dependencies:

```
<dependencies>
    <dependency>
        <groupId>org.bonitasoft.engine</groupId>
        <artifactId>bonita-client</artifactId>
        <version>Z.X.Y</version>
    </dependency>
</dependencies>
```

Remember to replace the Z.X.Y with your Bonita BPM version

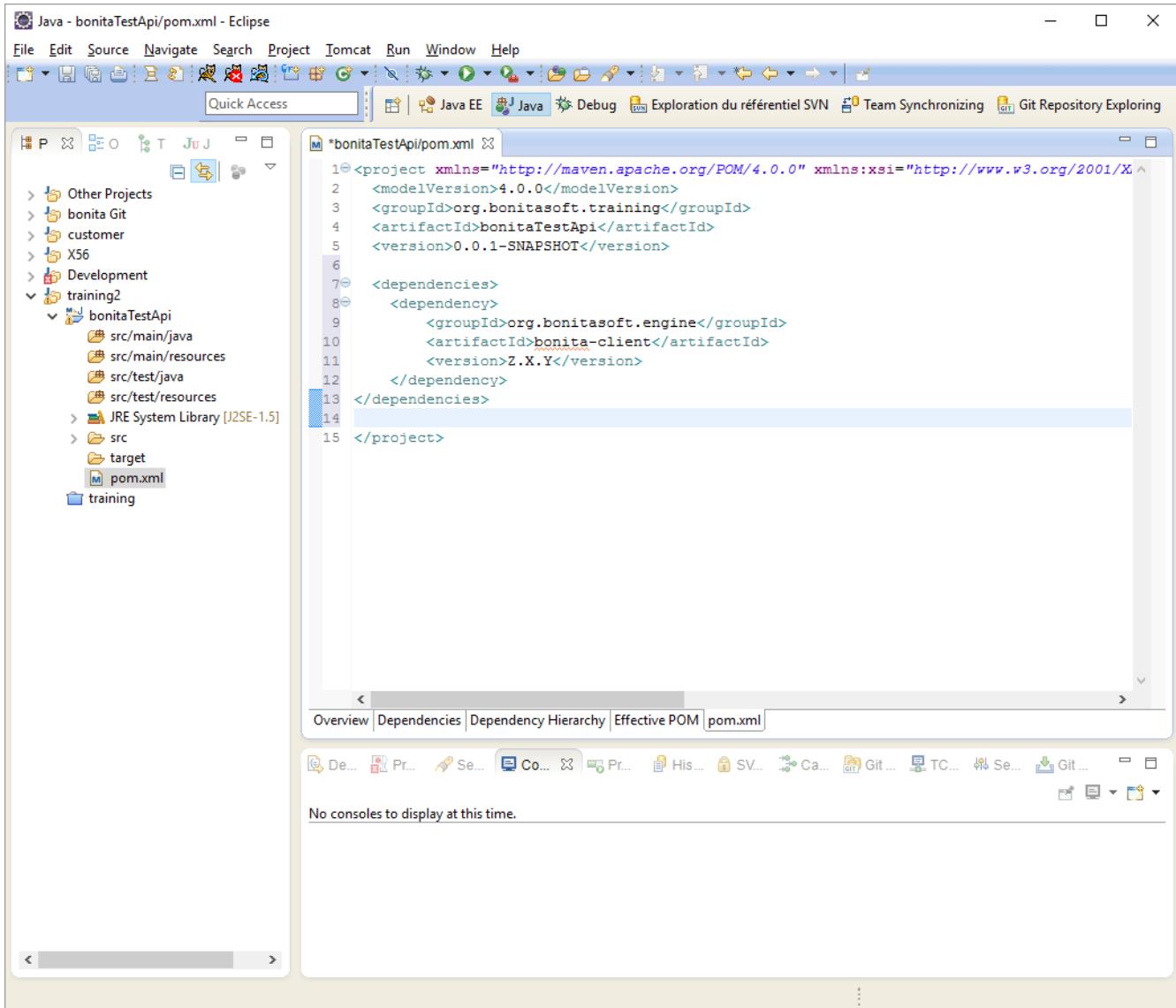


Figure 20. Update the pom.xml

1. Update the Maven project : right click on the project, then select Maven → Update project

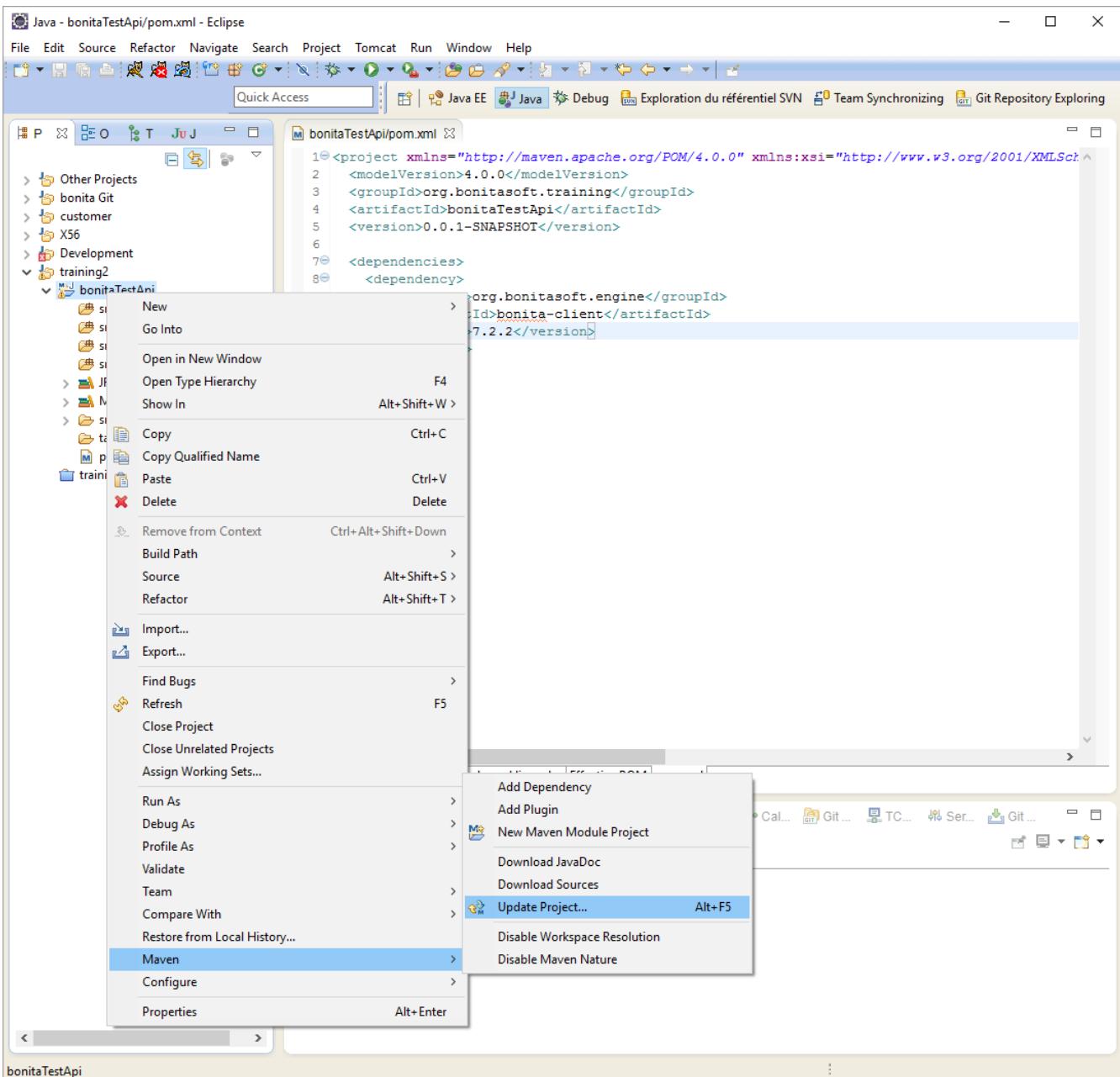


Figure 21. Update the maven project.xml

All the

2.3.2. Create the java class

Create a java class with a static void main method, there write the code to connect to Bonita BPM Engine and retrieve the user list. Create a class TestUserList in the package com.bonitasoft.training



Try not to copy the whole block of code at once, but block by block in order to better understand the single steps

```

package org.bonitasoft.training;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

```

```

import java.util.logging.Level;
import java.util.logging.Logger;

import org.bonitasoft.engine.api.ApiAccessType;
import org.bonitasoft.engine.api.IdentityAPI;
import org.bonitasoft.engine.api.LoginAPI;
import org.bonitasoft.engine.api.TenantAPIAccessor;
import org.bonitasoft.engine.identity.User;
import org.bonitasoft.engine.identity.UserCriterion;
import org.bonitasoft.engine.session.APISession;
import org.bonitasoft.engine.util.APITypeManager;

public class TestUserList {
    public static void main(String args[]){
        Logger logger = Logger.getLogger("TestUserList");
        try{

            /*Block 1: set API type */
            Map<String, String> map = new HashMap<String, String>();
            map.put("server.url", "http://localhost:8080");
            map.put("application.name", "bonita");
            APITypeManager.setAPITypeAndParams(ApiAccessType.HTTP, map);

            /*Block 2: Authentication */
            // Set the username and password
            final String username = "william.jobs";
            final String password = "bpm";
            // get the LoginAPI using the TenantAPIAccessor
            final LoginAPI loginAPI = TenantAPIAccessor.getLoginAPI();
            // log in to the tenant to create a session
            final APISession session = loginAPI.login(username, password);

            /*Block 3: Identity api and list of users */
            // get the identityAPI bound to the session created previously.
            final IdentityAPI identity = TenantAPIAccessor.getIdentityAPI(session);

            // get the list of users
            final List<User> user =
identity.getUsers(0,30,UserCriterion.FIRST_NAME_ASC);

            // display the list of userNames

            /*Block 4: Print the list of users */
            for(int i=0; i< user.size(); i++) {
                System.out.println(user.get(i).getUserName());
            }

            /*Block 5: Logout */
            loginAPI.logout(session);
        }
        catch(Exception e){

```

```

        logger.log(Level.SEVERE, "ERROR RETRIEVING USERS", e);
    }
}
}

```

2.3.3. Execute the project

Before the run, be sure you have a bonita bpm server instance running (standalone server or Bonita BPM Studio).

On the console you should get the user list of Bonita BPM organization.

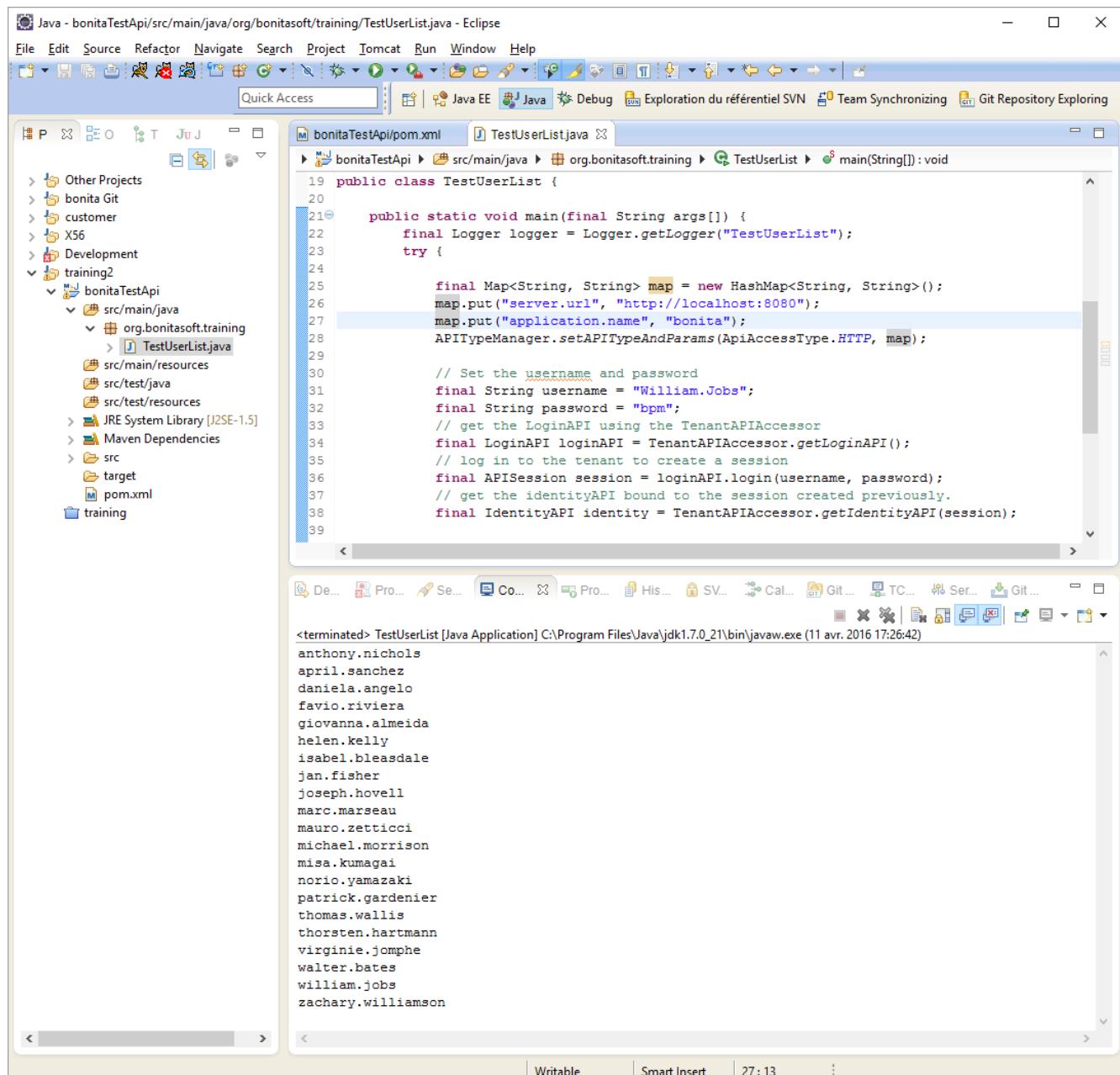


Figure 22. All users are returned.xml

Chapter 3. Using Bonita REST API

3.1. Prerequisites

You must have Bonita BPM Studio SP installed and started.

3.2. Objectives

The goal of this exercise is to become familiar with using REST to communicate and interact with the Bonita Engine.

3.3. Instructions

- Choose a Rest Client
- Connect to Bonita BPM engine
- Create a new user
- Get a list of users
- Get the List of Installed Processes
- Create a case
- Get the List of Pending Tasks
- Set Case Variable Value
- Logout

3.4. Correction

3.4.1. Download the REST client

Use a rest client plugin in your browser. For this exercise we will use the firefox RestClient plugin, but you are free to use any browser based or standalone REST client. (We recommend you do not use REST Easy for Firefox as it had some functional issues when tried with this exercise)

If REST Client is not already installed in your browser, you can install it via the menu (top right) and the Add-ons button.

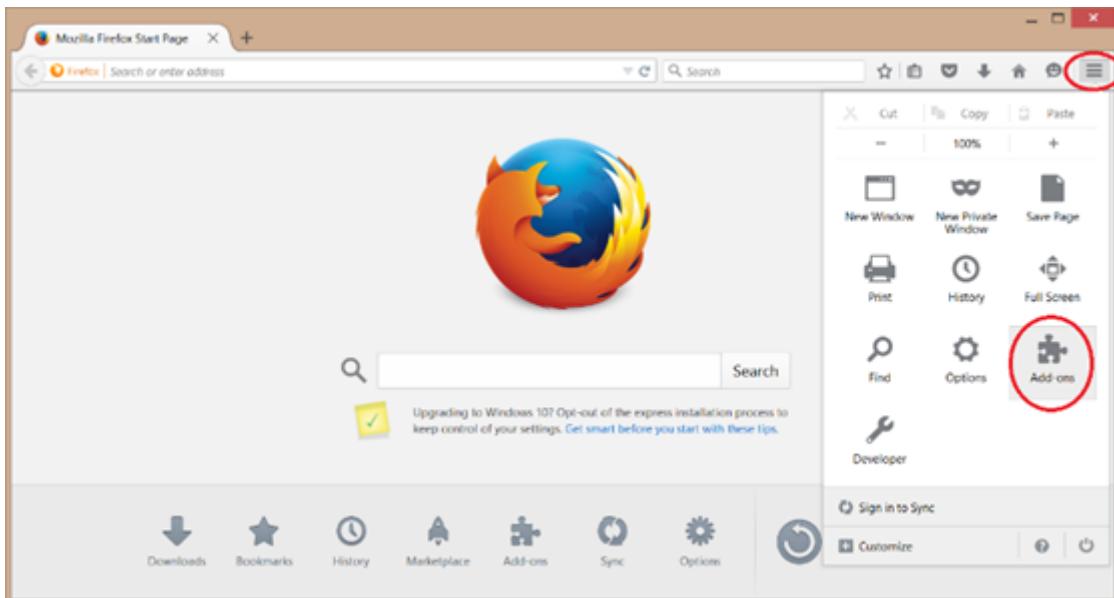


Figure 23. REST Client

Search for and Install REST Client:

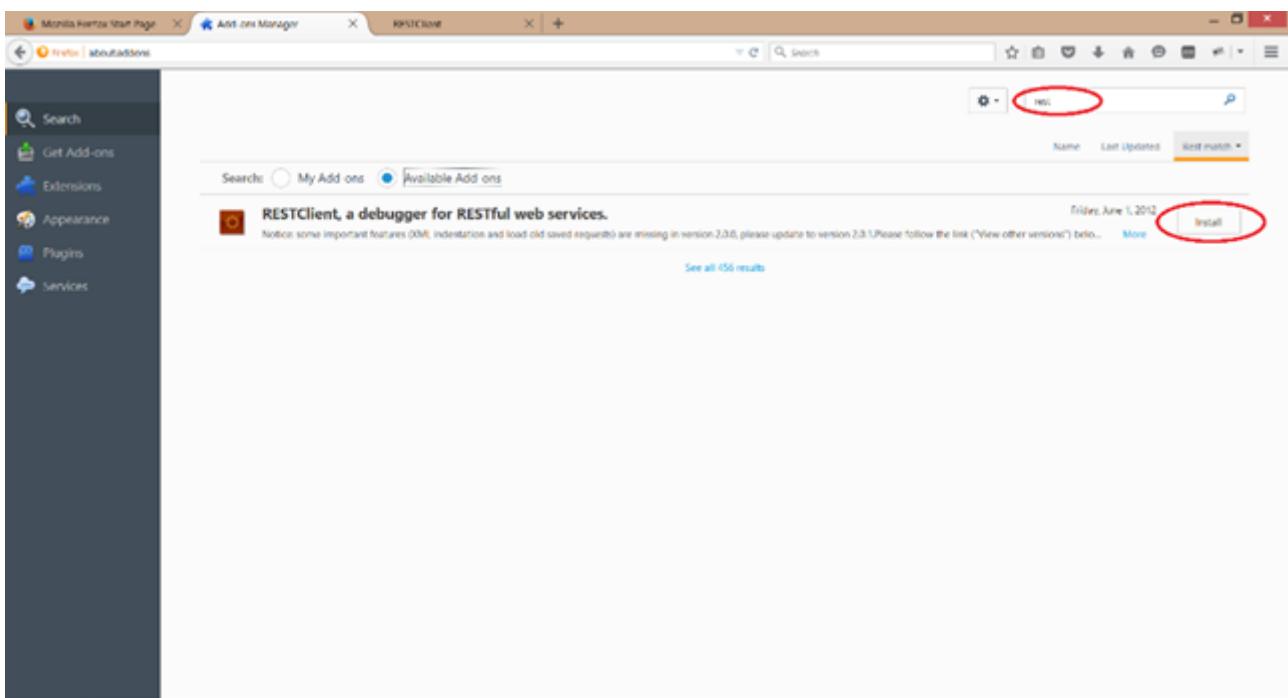


Figure 24. REST Client icon

After you restart Firefox Button to activate REST Client will appear beside the Menu button on your browser :

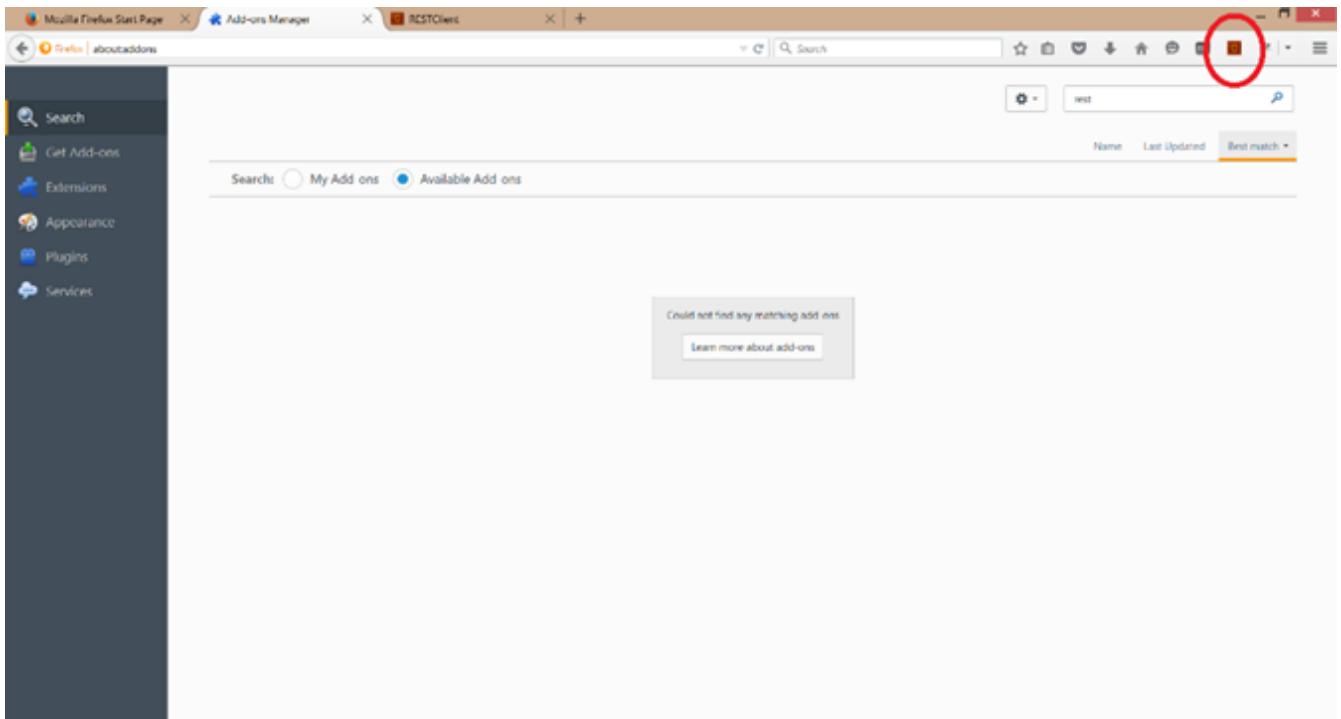


Figure 25. REST Client icon

3.4.2. Connect To The Engine

Click on the REST Client button to open the REST client page. The first operation is to connect to the Bonita Engine. A SessionID is created, and that SessionID will be sent back to the server on each subsequent call. See the documentation WEB REST API on the Bonita Documentation Website for more details. Configure the REST call as follows:

Item	Value
URL	http://localhost:8080/bonita/loginservice?
Method	POST
Content type	application/x-www-form-urlencoded

Note : setting the content type will require you to add a header to the request. Add the header via the Headers menu at the top of the REST Client page.

The screenshot shows the RESTClient application window. At the top, there are tabs for File, Authentication, Headers (which is circled in red), and View. On the right, there are buttons for Favorite Requests, Setting, and RESTClient logo. Below the tabs, there's a request configuration area with Method (POST), URL (http://localhost:8080/bonita/loginservice?), and a SEND button. Under the Headers section, there's a Content-Type header set to application/x-www-form-urlencoded. The Body section contains a form-encoded payload: username=Walter.Bates&password=bpm&redirect=false&redirectUrl=. A 'Remove All' button is also present.

Figure 26. Access the header

And click on the button



Figure 27. Send

Result : You should receive a successful answer: HTTP/1.1 200 OK

The screenshot shows the RESTClient interface after sending the request. The Response tab is selected. The response headers listed are:

- 1. Status Code : 200 OK
- 2. Content-Length : 0
- 3. Date : Thu, 03 Sep 2015 18:03:28 GMT
- 4. Server : Apache-Coyote/1.1
- 5. Set-Cookie : JSESSIONID=6BA52901121E3942AAFA4BB19014D8F4; Path=/bonita/; HttpOnly

Figure 28. Status 200

Extract the API SessionId from the Set-Cookie line in the response header. As long as the Session ID is set as a cookie in the header, the client can access the REST API.

OR

Using a second tab in the browser, just access the Bonita Portal, and connect using Walter.Bates. Then, the cookie is created and call from the REST CLIENT is working (the browser share the cookie between tab).

3.4.3. Create a New User

Using the Identity API, create a user. Use these values with the content type and the cookie set in the header.

Item	Value
------	-------

URL	http://localhost:8080/bonita/API/identity/user
Method	POST
Content type	application/json; charset=UTF-8
Body	{"userName":"john.doe","password":"bpm", "password_confirm":"bpm","firstname":"John","lastname":"Doe"}

The screenshot shows the RESTClient interface. At the top, there are navigation links: File, Authentication, Headers, View, Favorite Requests, Setting, and the title RESTClient. Below this is a section titled [-] Request. It contains fields for Method (POST), URL (http://localhost:8080/bonita/API/identity/user), and a large text area for the Body containing the JSON payload: {"userName":"john.doe","password":"bpm", "password_confirm":"bpm","firstname":"John","lastname":"Doe"}. There are also sections for Headers and Cookies.

Figure 29. Create a user

The result must be HTTP/1.1 200 OK in the response header :

The screenshot shows the RESTClient interface with a section titled [-] Response. It displays the Response Headers for a successful 200 OK response. The headers listed are:

```

1. Status Code      : 200 OK
2. Cache-Control   : no-cache,no-store,no-transform,max-age=0
3. Content-Type    : application/json;charset=UTF-8
4. Date            : Thu, 03 Sep 2015 18:22:31 GMT
5. Expires          : 02 Sep 2015 18:22:30 GMT
6. Pragma           : No-cache
7. Server           : Apache-Coyote/1.1
8. Transfer-Encoding : chunked

```

Figure 30. Result 200

But you may also navigate the response body for more details.

Confirm that the user was created by logging into the portal as an administrator and browsing to the (Inactive) Users in the Organization:

The screenshot shows the BonitaSoft BPM interface. At the top, there are navigation tabs: BPM, Organization, Resources, Applications, and Portal. On the left, there's a sidebar with 'CREATE' and filter options ('Active', 'Inactive'). The main area displays a user profile for 'John Doe'. Key details shown include 'Manager: System', 'Username: john doe', and 'Last login: No data'. A note indicates 'Last update: 09/03/2015 2:22 PM'. Below the profile, sections for 'Profile' and 'Membership' show 'No data'.

Figure 31. User John Doe

3.4.4. Get a List of Users

Send the following API call as a GET message (keep the headers, but the request body is not used for a "Get" call).

Be careful with copy/paste, there are spaces in the URL that you should get rid of

Item	Value
URL	http://localhost:8080/bonita/API/identity/user?p=0&c=10&o=userName%20ASC&f=enabled%3dtrue
Method	GET
Content type	
Body	

The screenshot shows the RESTClient interface. At the top, there are dropdown menus for File, Authentication, Headers, and View, and buttons for Favorite Requests, Setting, and RESTClient logo. Below this, the 'Request' section is titled '[-] Request'. It shows a 'Method' dropdown set to 'GET', a 'URL' input field containing '/localhost:8080/bonita/API/identity/user?p=0&c=10&o=userName%20ASC&f=enabled%3dtrue', and a 'SEND' button. Under the 'Headers' section, there are two fields: 'Content-Type: application/json,c...' and 'Cookie: JSESSIONID=BA88726B4CE5D...'. The 'Body' section is titled 'Body' and contains a text area labeled 'Request Body'.

Figure 32. Get list of users

View the user details in the response body.

Note : the function return a list, so parameter "p" is mandatory. This parameter is the Page number, counting from 0. "c" is the number of item per page, and "o" is used to give a sort key.

3.4.5. Get the List of Installed Processes

Use the GET method on the URL

Item	Value
URL	http://localhost:8080/bonita/API/bpm/process?p=0
Method	GET
Content type	
Body	

And then run your request: you get the list of all processes in JSON format

Result is:

The screenshot shows the RESTClient interface. In the 'Request' section, the method is set to 'GET', the URL is 'http://localhost:8080/bonita/API/bpm/process?p=0', and the body is empty. In the 'Headers' section, there are two entries: 'Content-Type: application/json; charset=UTF-8' and 'Cookie: JSESSIONID=BA88726B4CE5D...'. In the 'Response' section, the 'Response Body (Raw)' tab is selected, displaying the following JSON array:

```
[{"id": "6807560759317448659", "icon": "", "displayDescription": "", "deploymentDate": "2015-09-03 14:43:55.428", "description": "", "activationState": "ENABLED", "name": "A Sample Process", "deployedBy": "4", "displayName": "A Sample Process", "actorInitiatorId": "1", "lastUpdateDate": "2015-09-03 14:43:56.795", "configurationState": "RESOLVED", "version": "1.0"}]
```

Figure 33. List of processes

3.4.6. Create a Case

To start a new case you must specify the URL, you must use the POST method. As content type, you must use: application/json; charset=UTF-8. In the body you must add the process definition identifier as parameter, you can take it from the process definition ID in response in Step 5

Item	Value
------	-------

URL	http://localhost:8080/bonita/API/bpm/case
Method	POST
Content type	application/json; charset=UTF-8
Body	{"processDefinitionId":7708373484763319816}

Result is

The screenshot shows the RESTClient interface. In the 'Request' section, the method is set to POST, the URL is http://localhost:8080/bonita/API/bpm/case, and the body contains the JSON {"processDefinitionId":7708373484763319816}. In the 'Response' section, the status is 200 OK, and the response body is a JSON object with fields like id, end_date, startedBySubstitute, start, state, rootCaseId, started_by, processDefinitionId, and last_update_date.

Response Headers	Response Body (Raw)	Response Body (Highlight)	Response Body (Preview)
------------------	---------------------	---------------------------	-------------------------

```
{"id": "1", "end_date": "", "startedBySubstitute": "4", "start": "2015-09-03 14:50:48.690", "state": "started", "rootCaseId": "1", "started_by": "4", "processDefinitionId": "6807560759317448659", "last_update_date": "2015-09-03 14:50:48.690"}
```

Figure 34. Create a case

The result must be HTTP/1.1 200 OK

3.4.7. Get the List of Pending Tasks

To know the ID of a user, access the portal as an administrator, and click on a user. The URL contains the user's ID.

Example : when clicking on Walter Bates

http://localhost:8080/bonita/console/homepage#?id=_4&p=userlistingadmin&pf=2&f=allusersfilter

We note that the Id is 4. The URL specifies the user_id.

Item	Value
------	-------

URL	http://localhost:8080/bonita/API/bpm/humanTask?p=0&c=10&f=state%3dready&f=user_id%3d4
Method	GET
Content type	
Body	

Result is

The screenshot shows the RESTClient interface. In the top header, there are tabs for File, Authentication, Headers, and View, along with buttons for Favorite Requests and Setting. The title bar says "RESTClient". Below the header, the "Request" section is titled "[-] Request". It contains a form with "Method" set to "GET", "URL" set to "http://localhost:8080/bonita/API/bpm/humanTask?p=0&c=10&f=state%3dready&f=user_id%3d4", and a "SEND" button. Under the "Headers" section, there are two entries: "Content-Type: application/json; charset=UTF-8" and "Cookie: JSESSIONID=BA88726B4CE5D...". The "Body" section is titled "Body" and contains a text area labeled "Request Body". In the "Response" section, which is titled "[-] Response", there are four tabs: Response Headers, Response Body (Raw), Response Body (Highlight), and Response Body (Preview). The "Response Body (Raw)" tab is selected and displays the JSON response: [{"displayDescription": "", "executedBySubstitute": "0", "processId": "6807560759317448659", "parentCaseId": "1", "state": "ready", "rootContainerId": "1", "type": "USER_TASK", "assigned_id": "", "assigned_date": "", "id": "2", "executedBy": "0", "caseId": "1", "priority": "normal", "actorId": "1", "description": "", "name": "Step1", "reached_state_date": "2015-09-03 14:50:48.847", "rootCaseId": "1", "displayName": "Step1", "dueDate": "2015-09-03 15:50:48.809", "last_update_date": "2015-09-03 14:50:48.847"}].

Figure 35. List of pending tasks

3.4.8. Set Case Variable Value

The URL contains the activity identifier. In this example, you will set the value of aSampleVariable variable to a new "updatedValue". The used method is PUT. In this example, the case Id = 4 and the variable name is "aSampleVariable"

Item	Value
URL	http://localhost:8080/bonita/API/bpm/caseVariable/4/aSampleVariable
Method	POST
Content type	application/json; charset=UTF-8

Body	{"type": "java.lang.String", "value": "updatedValue"}
------	---

Access the Bonitasoft portal and verify if the value is updated

Case variables

Name	Type	Value	Actions
aSampleVar	java.lang.String	updatedValue	Edit

1 of 1

Figure 36. Case variable value

3.4.9. Logout

Item	Value
URL	http://localhost:8080/bonita/logoutservice
Method	GET
Content type	
Body	

After this call, any request will fail with the following response

[-] Response

Response Headers Response Body (Raw) Response Body (Highlight) Response Body (Preview)

```

1. Status Code      : 401 Unauthorized
2. Content-Length   : 0
3. Date            : Thu, 03 Sep 2015 19:55:45 GMT
4. Server           : Apache-Coyote/1.1

```

Figure 37. Logout

Chapter 4. Vacation Request Use case

For next exercises we will use an example application for managing payed vacation.

Helen Kelly, HR manager of the ACME corporation, wants an application that employees can use to track their paid vacation allowance and request days off.

4.1. Use case description

This is how the application works:

- When an employee log in to the application, he wants to view the number of days available for him.
- Then he can fill a form to request vacation
- The request is sent to the employee manager who reviews it and approves or refuse it.
- At any time the employee can cancel or update an existing request.
- Every manager will have an overall status of his employees vacations, and of course, he will be able to validate/refuse requests.

4.2. Use case implementation

As we know, the application logic is managed by processes. In the described use case we can imagine to have several processes, one for each vacation operation: Create a new Request, Delete it and update it.

These processes are provided, you can check them to see how they work, but you don't need to modify them, unless you want to apply customizations to understand them better.

In the following exercises you will work on creating the front end application that drives the processes. In fact the processes are not intended to be started from Bonita Studio or Bonita Portal, but from this external application that we are going to create.

4.3. Preliminary work

For this exercise we will use Bonita BPM Studio.

4.3.1. Create a new repository

In order to not mess up with previous works you can create a new separate local repository:

in the Bonita BPM Studio use the menu **Repository** → **Local** → **Create new repository**.

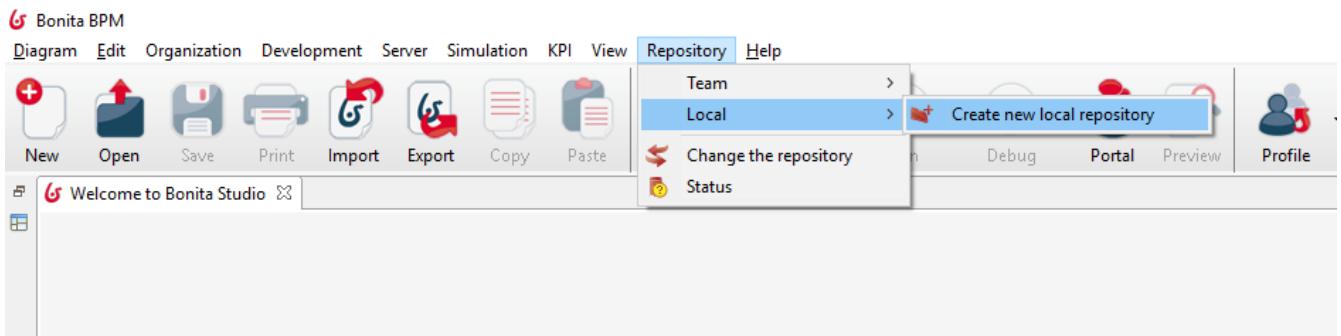


Figure 38. Create a new repository

4.3.2. Deploy existing processes

Download and import vacation Request processes: **VacationRequest-1.0.bos**. You can notice that one bdm is imported too, and it contains two Business Objects: **VacationAvailable** and **VacationRequest**

- From the Studio, click on Import button from the Cool bar and choose the provided **VacationRequest-1.0.bos**

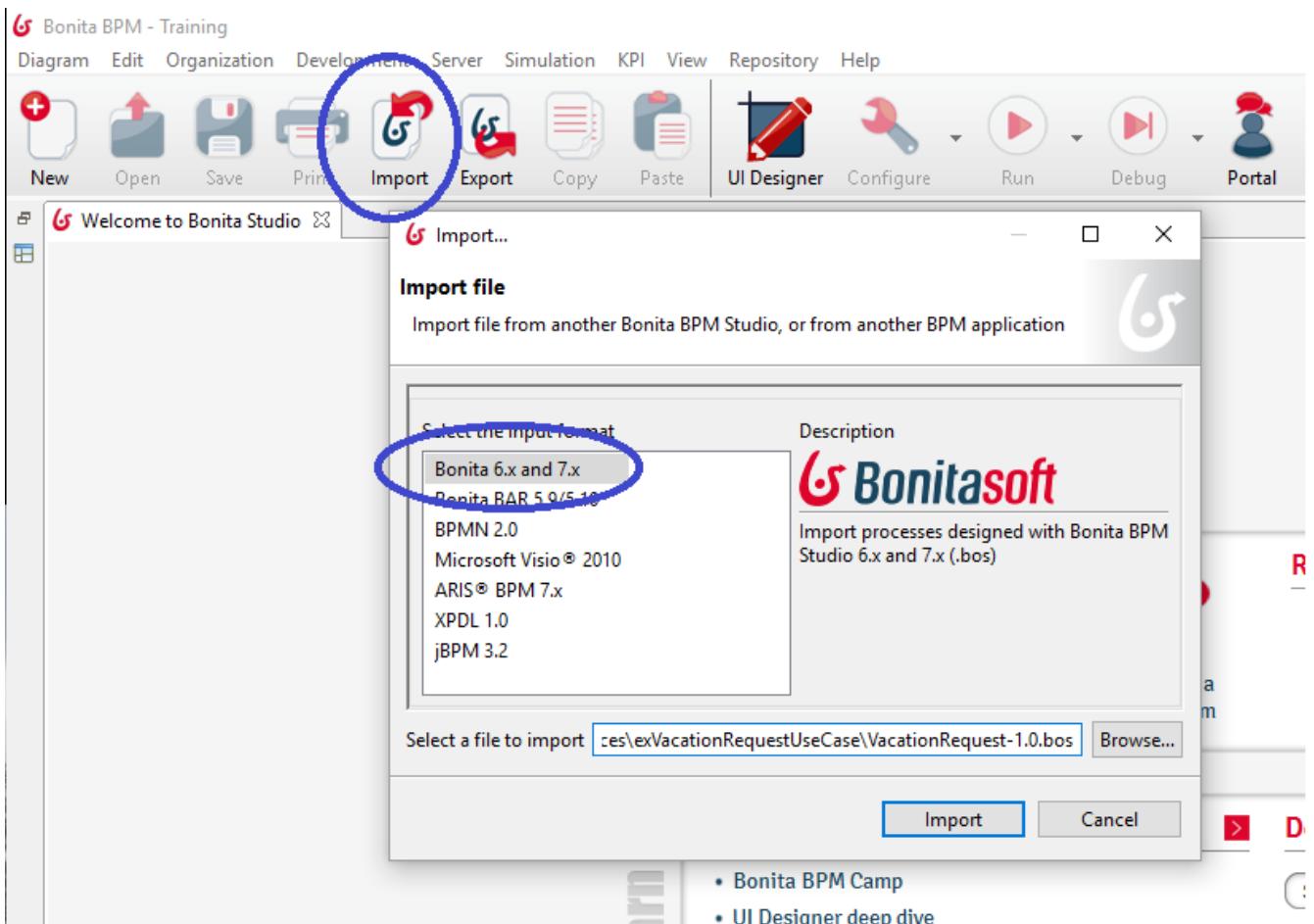


Figure 39. Import diagrams

Then, we need to deploy the processes using the Run button.

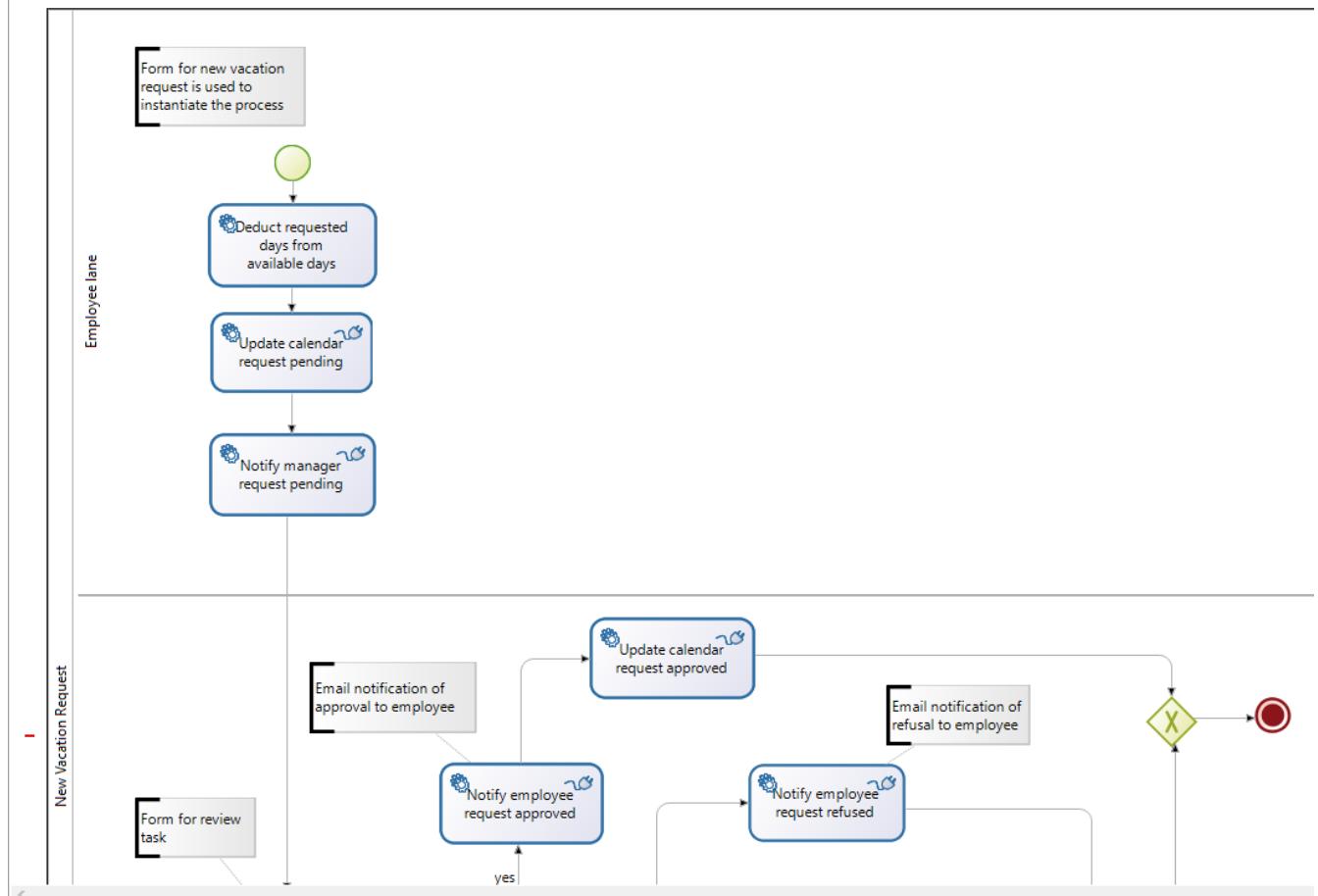
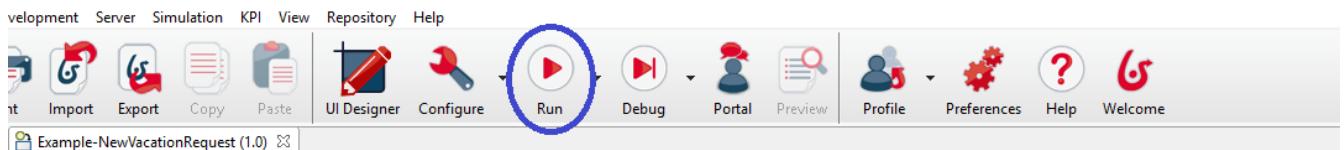


Figure 40. Run button



The button "Run" makes two actions: deploy and execute, but for our needs we just use it to deploy.

- Run **Example-InitiateVacationAvailable** process - this will add some initial data into the BDM. A browser page is opened, click on "Initialize vacation available" button and close the page.

Click on the button below to initialize users' individual vacation counters.

[Initialize vacation available](#)

VacationAvailable business data successfully initialized!

Figure 41. Click to run the process

Now we have 10 vacation days available for each user in our Organization.

- Run **Example-CancelVacationRequest**, **Example-NewVacationRequest** **Example-ModifyPendingVacationRequest** processes and close the pages (just click RUN to deploy the processes)

4.3.3. Check your BDM data

Once you do it go and check the database to better understand your model.

Do you remember how to connect to the h2 database embedded in your studio?

Go to **BONITA_STUDIO\workspace\tomcat\lib\bonita**

and double click on the jar file that starts with h2.

Alternatively open a command prompt on that directory and execute the command:

```
java -jar h2-1.3.170.jar
```

A browser with the h2 console will be opened.

The screenshot shows a 'Login' interface for the H2 database. The 'Saved Settings' dropdown is set to 'external database'. The 'Setting Name' field contains 'external database' with 'Save' and 'Remove' buttons. The 'Driver Class' field contains 'org.h2.Driver'. The 'JDBC URL' field contains 'jdbc:h2:tcp://localhost:9091/business_data.db;MVCC=T'. The 'User Name' field contains 'sa'. The 'Password' field is empty. At the bottom are 'Connect' and 'Test Connection' buttons.

Figure 42. H2 Connection

In the jdbc URL put:

```
jdbc:h2:tcp://localhost:9091/business_data.db
```

leave the other fields unchanged and click on Connect. This will lead to the main dashboard from where you can execute query like:

*SELECT * FROM VACATIONAVAILABLE*

to show vacations available for each user.

The screenshot shows the H2 Database Browser interface. On the left, there is a tree view of database objects:

- jdbc:h2:tcp://localhost:9091/busi
- VACATIONAVAILABLE
- VACATIONREQUEST
- INFORMATION_SCHEMA
- Sequences
- Users
- H2 1.3.170 (2012-11-30)

On the right, there are two panes. The top pane contains a SQL statement:

```
SELECT * FROM VACATIONAVAILABLE
```

The bottom pane displays the results of the query as a table:

PERSISTENCEID	BONITABPMID	DAYSAVAILABLECOUNTER	PERSISTENCEVERSION
1	18	10	0
2	2	10	0
3	17	10	0
4	11	10	0
5	21	10	0
6	3	10	0
7	10	10	0

Figure 43. H2 Browser information

bonitaBpmId is the user Id

Now we have processes and data we are ready to create the application and its pages, that's the goal of next exercises.

Chapter 5. Create a page for the Employees

5.1. Objectives

In the previous exercise we created the basic environment in order to start creating our Leaving Request application. The goal of this exercise is to create a page for the employees to be used in this application. This "custom" page will be created using the UI designer as seen for the form in previous chapters. It will:

- show the **logged user** name
- show the number of **available days**
- show a **list** of his vacation requests
- give the possibility to **create a new one** through a button link.

Optional

- Create buttons to modify/cancel the vacation requests
- Display a different background-color due to the state's request
- Display a text only if there are no requests
- Add your Company logo

5.1.1. Overview

Tahiti Vacation Management

Welcome to the Tahiti Vacation Management Page. {{session.user_name}}. You have {{myVacationAvailable[0].daysAvailableCounter}} vacation days available.

+ New vacation

⚠ There are no vacation requests yet. Use the button to create a new request.

The screenshot shows a modal dialog box with a light blue border. Inside, there are four input fields arranged in a grid-like layout. Top-left: 'Start date' with a text input 'data:\$item.startDate' and a calendar icon. Top-right: 'Return date' with a text input 'data:\$item.returnDate' and a calendar icon. Bottom-left: 'Numer of days' with a text input 'data:\$item.numberOfDays'. Bottom-right: 'Status' with a text input 'data:\$item.status'. At the bottom left is an orange 'Modify' button, and at the bottom right is a red 'Cancel' button.

Figure 44. Overview

5.2. Instructions

- Create a new page
- Create all the data needed for your page (API variables to load the BDM,...)
- Drag and Drop the widgets used on your page, you will need Text, Title, Containers, Datepickers, Inputs and Link widgets
- Bind all your data to your widgets
- Test your page
- Create an application that contains your page
- Test your application page.

5.3. Correction

This section provides details about steps to follow to create a page using best practices

5.3.1. Create a new empty page

- Run the UI Designer either by clicking on UI Designer icon on the Studio image::exPageEmployee/exPageEmployee_02.png[title=UI Designer]
- On the UI Designer home page, click on "+ Create" button and be sure that the Type "Application page" is checked. Name your page **myVacationRequest** and click on Create

button.

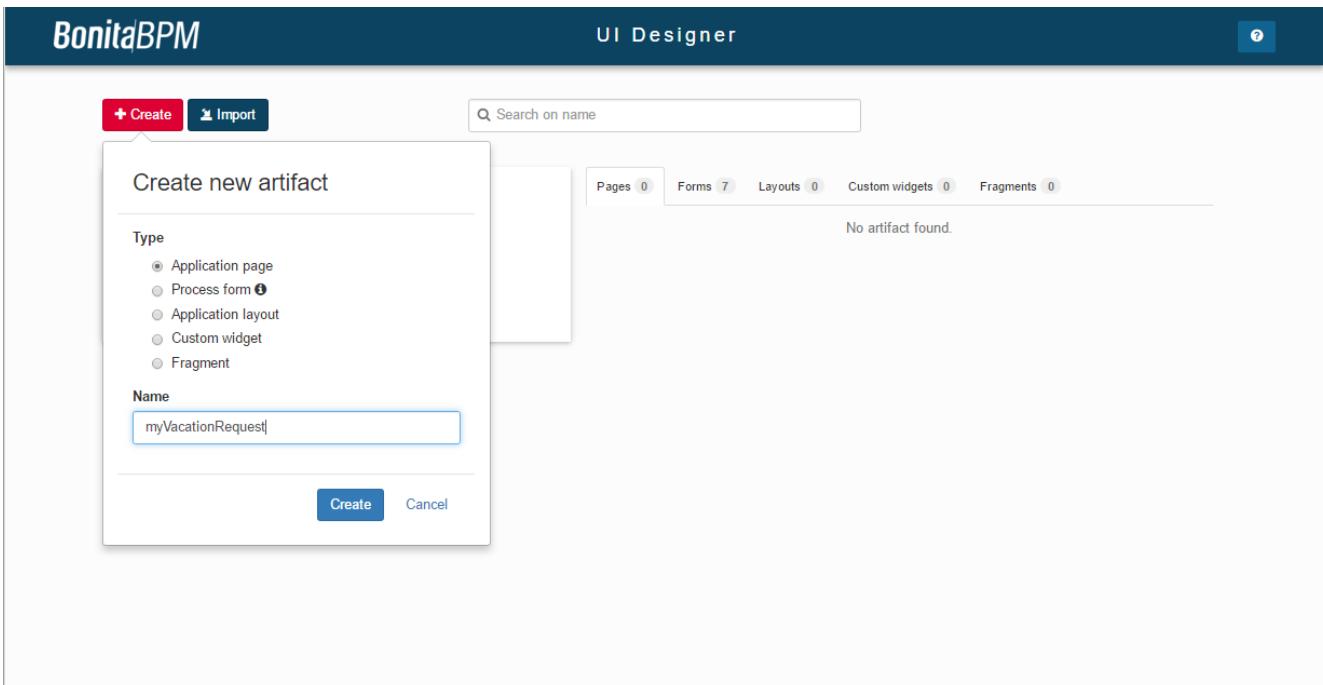


Figure 45. Create your page

- Your page is created

5.3.2. Create the data

For your page communication and logic you will need some data. Below the list of data you need to create with the detailed use of each one.

Data creation

In the page, search the button "Create a new variable" and for each line, create a variable.

The screenshot shows the Bonita BPM Page Editor. The top navigation bar includes "PAGE EDITOR", a search bar with "myVacationRequest", "Save", "Preview", and other icons. On the left, a toolbar contains icons for "CONTAINER", "H1", "INPUT", "TEXT AREA", "SELECT", "CHECKLIST", "RADIO BUTTONS", "CHECKBOX", "DATE PICKER", "UPLOAD", "TEXT", "DATA TABLE", "LINK", and "BUTTON". The main workspace is labeled "This page is empty. To add content, drag and drop components from the palette to the whiteboard above." In the bottom left corner of the workspace, there's a "VARIABLES" tab with a red circle around the "Create a new variable" button. A tooltip on the right says "Select an element on the whiteboard, then set its properties here". The "VARIABLES" table shows one entry: "myVacation" with a value of ".../API/bdm/businessData/com.company.model.VacationRequest?q..." and a type of "External API".

Figure 46. Create variables

Name	Type	Content
myVacation	External API	.. /API/bdm/businessData/com.company.model.VacationRequest? q=findByRequesterBonitaBPMId&p=0&c=100&f=requesterBonitaBPMId={{session.user_id}}
myVacationAvailable	External API	.. /API/bdm/businessData/com.company.model.VacationAvailable?q=findByBonitaBPMId&p=0&c=1&f=bonitaBPMId={{session.user_id}}
processCreate	External API	.. /API/bpm/process?s>New%20Vacation%20Request&p=0&c=10&o=version&f=activationState=ENABLED
session	External API	.. /API/system/session/unusedid
urlStartCreateProcess	Javascript expression	<pre> if (\$data.processCreate && \$data.processCreate.length > 0) { return "/bonita/portal/resource/process/ + \$data.processCreate[\$data.processCreate.length-1].name+ + \$data.processCreate[\$data.processCreate.length-1].version+ "/content/?id="+ \$data.processCreate[\$data.processCreate.length-1].id; } else { return null; }</pre>

More about created data

myVacation

This API call is used to load the Vacation Request business data using the requester id (f parameter) and the user_id provided by the session.

myVacationAvailable

This API call is used to load the Vacation Available business data using the current logged user.

processCreate

This API call is used to gather information of the "New Vacation Request" process.

session

This API call returns the current logged user info.

urlStartCreateProcess

This JS expression return the URL to start "New Vacation Request" process using the processCreate data.

Note 1: to access the another variable: use \$data.<name>.

Note 2 : the javascript test first if the \$data.processCreate exist, because the UI Designer will call the JavaScript first at begining, and when any REST API result arrive on the browser.

5.3.3. Build your page

Now we can start to design "**graphically**" our page. We are going to add widgets, configure them and bind them to related data. This will be the last step of your page creation.

5.3.4. Page mock-up

Before getting started, you should design your page mock-up. As you have all the variables inside and outside your page already created, you can know which widgets you will use. You can also take the opportunity to think about the widgets placement and the global look'n'feel of your page.

5.3.5. Choosing and placing your widgets

We are going to need:

- a **Title widget** to display the name of our page
- a **Text widget** to display the current user name and his available vacation days
- a **Container widget** to iterate the vacation requests with
 - a **Datepicker widget** to show the Start date
 - a **Datepicker widget** to show the Return date
 - an **Input widget** to show the Number of days
 - an **Input widget** to show the Status of the request
- a **Link widget** as a button to call the create a new instance of the NewVacationRequest process

Title of our Custom Page

Add a Title widget to the whiteboard and fill in their properties.

Property name	Property value
Text	Tahiti Vacation Management
Title level	Level 1
Alignment	left

Figure 47. Title widget

Text widget

Add a Text widget below the Title widget and fill in their properties.

Property name	Property value
Text	Welcome to the Tahiti Vacation Management Page, {{session.user_name}} . You have {{myVacationAvailable[0].daysAvailableCounter}} vacation days available.
Width	6
Alignment	left

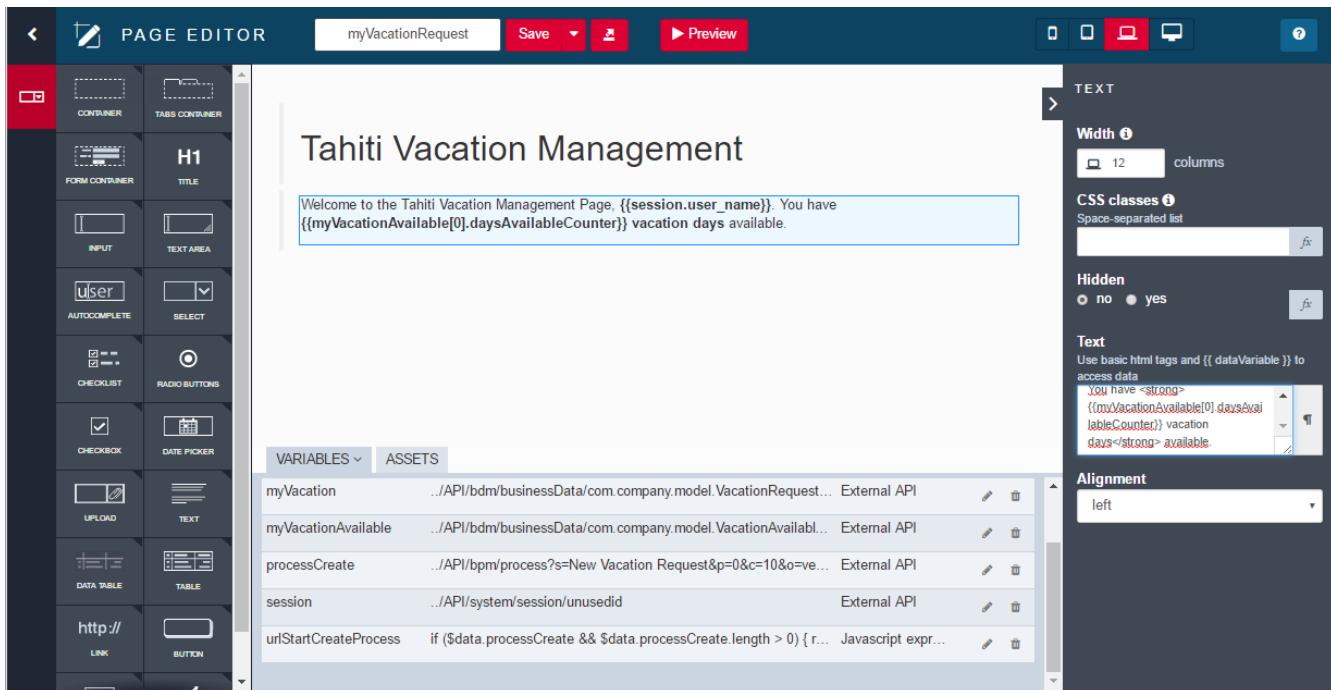


Figure 48. Create variables

Text property is using an Interpolation Type of data (thanks to AngularJS), so we can use basic html tags and **{{ dataVariable }}** to access data.

We are using **session** variable, that returns a JSON like this:

```
session: { "user_name": "helen.kelly", "session_id": "8247337820069982753", "is_technical_user": "false", "user_id": "3", "conf": "[\"976A80D4E3EC6F1D2D1 ... 2ACA9B5EA2B562E\"]", "copyright": "Bonitasoft © 2016", "version": "7.2.0" }
```

We want to display the **user_name** attribute so we use: **{{session.user_name}}**

Instead, **myVacationAvailable** variable returns a JSON list like this:

```
myVacationAvailable: [ { "persistenceId": 60, "persistenceId_string": "60", "persistenceVersion": 0, "persistenceVersion_string": "0", "bonitaBPMId": 3, "bonitaBPMId_string": "3", "daysAvailableCounter": 10 } ]
```

We need to select the first (and unique) element of the list, and display the **daysAvailableCounter** attribute, so we use: **{{myVacationAvailable[0].daysAvailableCounter}}**

5.3.6. Container widget

Drag and drop a Container widget below the Text widget and use these properties:

Property name	Property value
Collection	myVacation

Figure 49. Collection

This Collection parameter is expecting an Array, so we are using **myVacation** variable, which is returning a JSON list, to iterate the content of the list using **\$item** inside for each iteration.

Drag and drop 2 Datepicker widgets and 2 Input widgets inside the Container widget, and configure their properties this way:

Start date - Datepicker

Property name	Property value
	Width
6 Read-only (<i>bound to an expression or variable</i>)	\$item.status != "pending"
Label	Start date
Value	\$item.startDate

Number of days - Input

Property name	Property value
Width	6
Read-only (<i>bound to an expression or variable</i>)	\$item.status != "pending"
Label	Number of days
Value	\$item.numberOfDays

Return date - Datepicker

Property name	Property value
Width	6
Read-only (<i>bound to an expression or variable</i>)	\$item.status != "pending"
Label	Return date
Value	\$item.returnDate

Status - Input

Property name	Property value
Width	6
Read-only	yes
Label	Status
Value	\$item.status

Figure 50. All sub widgets

We want to activate the Read-only property only when vacation request status is not "pending". Using **\$item** inside the container allows us access the current list element into the list iteration.

5.3.7. Link widget as a button

Drag and drop a Link widget between the Text widget and the Container widget and configure its properties:

Property name	Property value
Text	+ New vacation
URL (<i>bound to an expression or variable</i>)	urlStartCreateProcess
Alignment	right
Style	primary

The screenshot shows the SAP Fiori UI Designer interface. On the left is a library of UI elements. In the center, there's a preview area titled "Tahiti Vacation Management". Below the title, there's a welcome message: "Welcome to the Tahiti Vacation Management Page, {{session.user_name}}. You have {{myVacationAvailable[0].daysAvailableCounter}} vacation days available." To the right of the message is a blue button labeled "+ New vacation". On the far right, there's a configuration panel for a "LINK" element. The "URL" property is set to "urlStartCreateProcess". Other properties like "Width", "CSS classes", "Hidden", "Text", "Frame", "Alignment", and "Style" are also visible.

Figure 51. Link widget

Link URL property is **Dynamic Value** type. You can use a dynamic value field to specify a constant (by default) or an expression. Click on the expression icon 'fx' to switch from constant to expression and start typing the name of your variable (in this case **urlStartCreateProcess**) and select it from the autocomplete selector.

5.3.8. Preview on different devices

Save your changes and use the **PREVIEW** button to check how your page is going to look like.

In order to access the information, your browser must be logged. So, in a first tab on your browser, connect as Walter.Bates. Then, access the UI Designer and click on Preview.
image::exPageEmployee/exPageEmployee_10.png[title=Connect to the portal]

On preview, you can see the the information about Walter.Bates are displayed:

Tahiti Vacation Management

Welcome to the Tahiti Vacation Management Page, **walter.bates**. You have 10 vacation days available.

[+ New vacation](#)

Figure 52. Preview as Walter Bates

Click on +New vacation, and add a vacation (no worry about the error 404 you get when you create it : you are in the preview). So, information are updated:

Tahiti Vacation Management

Welcome to the Tahiti Vacation Management Page, **walter.bates**. You have -5 vacation days available.

[+ New vacation](#)

start date

Return date

Number of days

Status

Figure 53. Preview

Use the different devices view button to test how widgets will be shown.

Tips

Each device view is directly related with each widget Width property. The value of this property can be from 1 to 12. This is based on Bootstrap, where a total of 12 cols divide the screen for each different device (extra-small, small, medium, large).

As an example, a Datepicker widget inside the container could have this width properties:

Device	Width property value
Large (width >= 1200px)	3
Medium (width >= 992px)	3
Small (width >= 768px)	12
Extra small (width < 768px)	12

Play with the different widget Width values that fits your page design for the different devices using the PREVIEW button.

5.3.9. Export your page

Once everything is ready on your page, click on the Export button. It will create a .zip file that we can use to import it as a resource into the Bonita Portal.

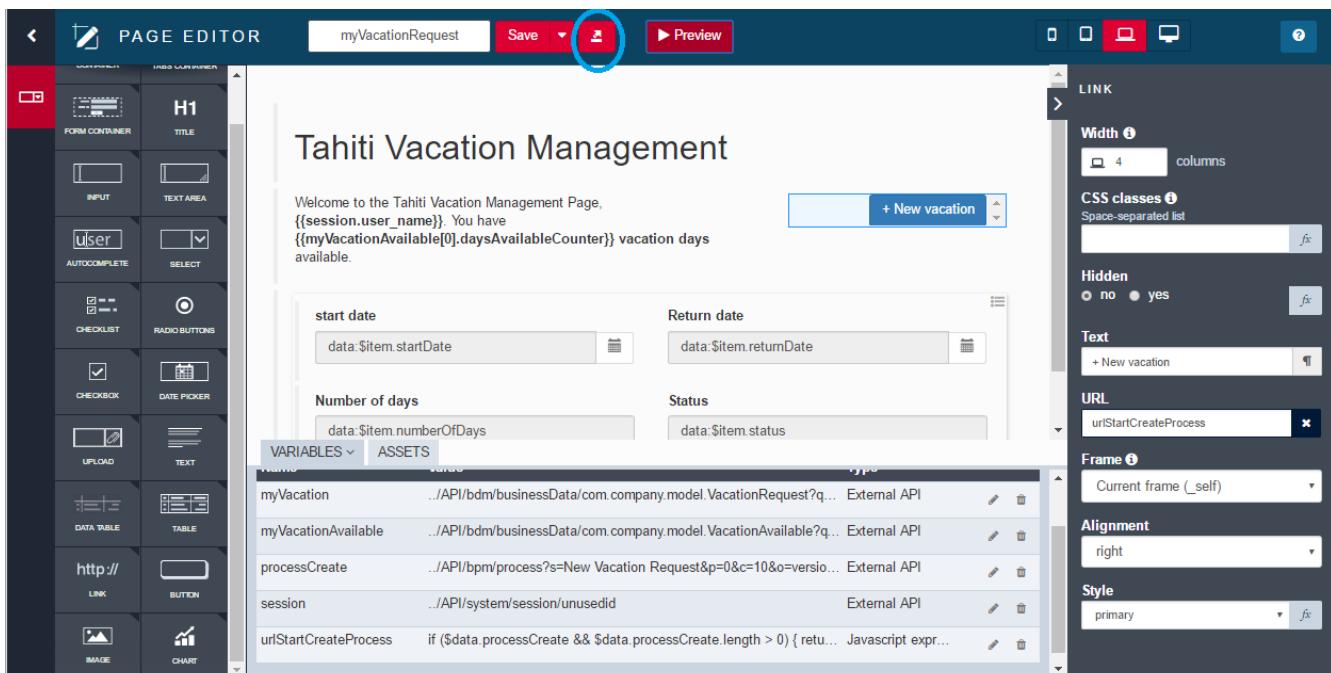


Figure 54. Export button

5.3.10. Create a new application and test your page

New application

- Go to the Bonita Portal using the **Administrator** profile, **Resources** section.

Welcome: Walter Bates - Administrator | Settings

Applications | **Portal** | **License**

API extension viewer page

API extension viewer page generated with Bonita BPM UI designer

Name for the URL
custompage_apiExtensionViewer

More information

A resource is imported as a zip archive containing a page.properties file and a resources folder. The resources folder must contain an index.groovy class or an index.html file and optionally can contain some additional resources.

The content type is defined in page.properties.

If you create a resource with the UI designer, the exported zip automatically has the correct format.

Page:Content type is 'page'.
To view an imported page, add it to an application page list and navigation.

Visible to
No data

localhost:8080/bonita/portal/homepage#

Autogenerated Task Form 2 hr ago

page-myVacationReq....zip

Figure 55. Import the page in the portal

- Import your **page-myVacationRequest.zip** file clicking on "+ ADD" button on the top left side.
- Go to **Applications** section and click on New button.

Welcome: Walter Bates - Administrator | Settings

BPM | **Organization** | **Resources** | **Applications** | **Portal**

Application list

NEW **IMPORT**

No application available.

To create an application, click New or Import.
An application is a customized environment for a specific user profile, in which users interact with business data and business processes in the most efficient way.

Figure 56. Application list

- Use this values:

Input name	Input value
Display name	Vacation Management
URL	tahiti

Input name	Input value
Version	1.0
Profile	User
Description	Vacation management for employees and managers

- Click on Create button.

The screenshot shows the Bonitasoft interface with a red header bar. In the center, a modal dialog box is open with the title 'Create an application'. The dialog contains several input fields: 'Display name *' with the value 'Vacation Management', 'URL *' with the value '..../apps/ tahiti', 'Version *' with the value '1.0', 'Profile *' set to 'User', and a 'Description' field containing 'Vacation management for employees and managers'. At the bottom right of the dialog are two buttons: 'CREATE' and 'CANCEL'.

Figure 57. Create the application

Your Vacation Management application is created. You can click on the URL link to access your application. There are no pages or menu elements included yet, just by default page/menu so, **let's add your page into your application!**

Configure application

- Click on the "three dots" action button to configure your application.

Name	Version	URL	Profile	Updated on	Actions
Vacation Management	1.0	./apps/tahiti	User	a few seconds...	... X trash

Figure 58. Access the application

We can change the Look & Feel selecting a different Layout or Theme and we can add more adding them into Resources section. Let's keep it simple for the moment but feel free to change the Look & Feel later.

Add a page

Let's add your Page and a new Navigation Menu for your application.

- Click on Pages "ADD" button and select **custompage_myVacationRequest - myVacationRequest page**. Also set "index" into the URL, finish with click on "ADD".

Add page

Page *
custompage_myVacationRequest - myVacationRequest page

URL *
..../apps/tahiti/index

ADD **CANCEL**

Pages

ADD home - Application home page (house icon)

Navigation

ADD

Specify the menu structure.
You must add a page before you can reference it in a menu.

Figure 59. Add the Vacation request page

The green house icon of the default added page shows that every time you access to your

application URL, that page will be shown first, as the Home page. Let's change it.

- Set your page as your application Home page making active the green house icon.

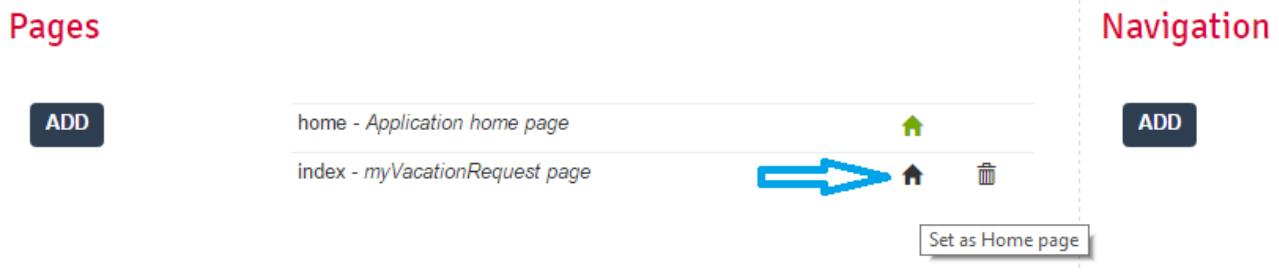


Figure 60. Set the home page

- Delete the default page using the trash icon

Every time end-users access to our application, myVacationRequest page will be shown as Home page.

Add a Menu element

We can add a Navigation element because in the future we will add another page into this application.

- Click on Navigation "ADD" button, use the Name "Index" and select your page. Click on "ADD" button.

The screenshot shows the BonitaSoft application management interface after configuration. At the top, the navigation bar includes 'BPM', 'Organization', 'Resources', 'Applications', 'Portal', and 'License'. The 'Resources' tab is selected. In the center, the 'Look & Feel' section shows 'Default layout' and 'Bootstrap default theme'. Below this are the 'Pages' and 'Navigation' sections. The 'Pages' section lists 'index - myVacationRequest page' with a green house icon. The 'Navigation' section lists 'Index' with a green house icon. There are 'ADD' buttons in both sections.

Figure 61. Final result

Test application

Now we can access our application clicking on the URL link or accessing to <http://localhost:8080/bonita/apps/tahiti/index/>

Tahiti Vacation Management

Welcome to the Tahiti Vacation Management Page, **walter.bates**. You have **-5** vacation days available.

[+ New vacation](#)

start date

06/19/2016

Return date

07/05/2016

Number of days

15

Status

pending

Figure 62. Application

Test the "+ New vacation" button and create a new vacation request.

5.4. Optional exercises

We can improve our Page adding some valuable features due to enhance the end-users experience.

Remember

- **Rename** your page using the Save as button.
- **Export** your new page.
- **Import** your new page directly in Bonita Portal.
- **Resources** section, selecting the old page and clicking on **Edit** button. It will overwrite the old page with the new improved one and you could test it on your application directly.

5.4.1. Display modify and cancel buttons

Create buttons to modify/cancel the vacation requests.

Data creation

We are going to need some new variables to call the **Modify Pending Vacation Request** and **Cancel Vacation Request** processes:

Name	Type	Content
processModify	External API	..//API/bpm/process?s=Modify Pending Vacation Request &p=0&c=10&o=version&f=activationState=ENABLED

Name	Type	Content
urlStartModifyProcess	Javascript expression	<pre>if (\$data.processModify && \$data.processModify.length > 0) { return "../API/bpm/process/" + \$data.processModify[\$data.processModify.length-1].id + "/instantiation"; } else { return null; }</pre>
processCancel	External API	<pre>../API/bpm/process?s=Cancel Vacation Request&p=0&c=10&o=version&f=activationState=ENABLED</pre>
urlStartCancelProcess	Javascript expression	<pre>if (\$data.processCancel && \$data.processCancel.length > 0) { return "../API/bpm/process/" + \$data.processCancel[\$data.processCancel.length-1].id + "/instantiation"; } else { return null; }</pre>

processModify and processCancel

These External APIs gather all the information related to the enable processes using a by name search parameter.

urlStartModifyProcess and urlStartCancelProcess

These JS expressions return the builded URL to call into the buttons in order to instanciate their processes.

Modify - Button widget

Drag and drop a Button widget into the **Container widget**, close to Status Input widget, and use this properties:

Property name	Property value
Width	6
Hidden (<i>bound to an expression or variable</i>)	\$item.status!="pending"
Label	Modify
Alignment	left
Style	warning
Action	POST
URL to call	<code>{{urlStartModifyProcess}}</code>

Property name	Property value
Data sent on click (<i>bound to an expression or variable</i>)	{"vacationRequestIdContract": \$item.persistenceId_string, "startDateContract": \$item.startDate, "returnDateContract": \$item.returnDate, "numberOfDaysContract": \$item.numberOfDays}
Target URL on success	/bonita/apps/tahiti/index/

Cancel - Button widget

Drag and drop a Button widget into the Container widget, close to Modify button widget, and use this properties:

Property name	Property value
Width	6
Hidden (<i>bound to an expression or variable</i>)	\$item.status!="pending"
Label	Cancel
Alignment	right
Style	danger
Action	POST
URL to call	{{urlStartCancelProcess}}
Data sent on click (<i>bound to an expression or variable</i>)	{"vacationRequestIdContract": \$item.persistenceId_string}
Target URL on success	/bonita/apps/tahiti/index/

The page is now: image::exPageEmployee/exPageEmployee_20.png[title=With modify and Cancel]

5.4.2. Display background-color

Display a different background-color due to the state's request

We are going to use nested-containers to achieve this exercise. We will add classes into the Vacation Requests container widget evaluating some expressions.

Main Container widget

- Add an extra **Container widget** above the current Container widget.
- Drag and drop the old Container widget (with all the widgets inside) into the new one.
- Configure the Main Container widget properties:

Property name	Property value
Collection	myVacation

Old Container widget

We need to remove `myVacation` variable from Collection widget property and add some expressions to evaluate whenever the status change within each iteration.

- Configure the sub **Container widget** properties this way:

Property name	Property value
CSS classes (<i>bound to an expression or variable</i>)	<code>(\$item.status == "refused") ? "alert alert-danger" : (\$item.status == "approved") ? "alert alert-success" : (\$item.status == "pending") ? "alert alert-warning" : (\$item.status == "cancelled") ? "alert alert-danger" : "alert"</code>
Collection	

The two container embeded image::exPageEmployee/exPageEmployee_21.png[title=Two containers]

and the result in preview (if you add a new Vacation request, and then cancel it) :

The screenshot shows a web application titled "Tahiti Vacation Management". At the top, it says "Welcome to the Tahiti Vacation Management Page, walter.bates. You have -4 vacation days available." There are two vacation requests listed:

- Vacation Request 1 (Orange Row):** Start date: 06/20/2016, Return date: 07/04/2016, Number of days: 14, Status: pending. It includes "Modify" and "Cancel" buttons.
- Vacation Request 2 (Pink Row):** Start date: 04/19/2016, Return date: 04/28/2016, Number of days: 5, Status: cancelled.

Figure 63. Color change

The first container do the iteration, when the second control the color.

5.4.3. Display a warning text

Display a text only if there are no vacation requests.

Text widget

- Add a **Text widget** above the Link widget + New vacation and configure its properties:

Property name	Property value
CSS classes	alert alert-danger
Hidden (<i>bound to an expression or variable</i>)	myVacation!=0
Text	 There are no vacation requests yet. Use the button to create a new request.
Alignment	left

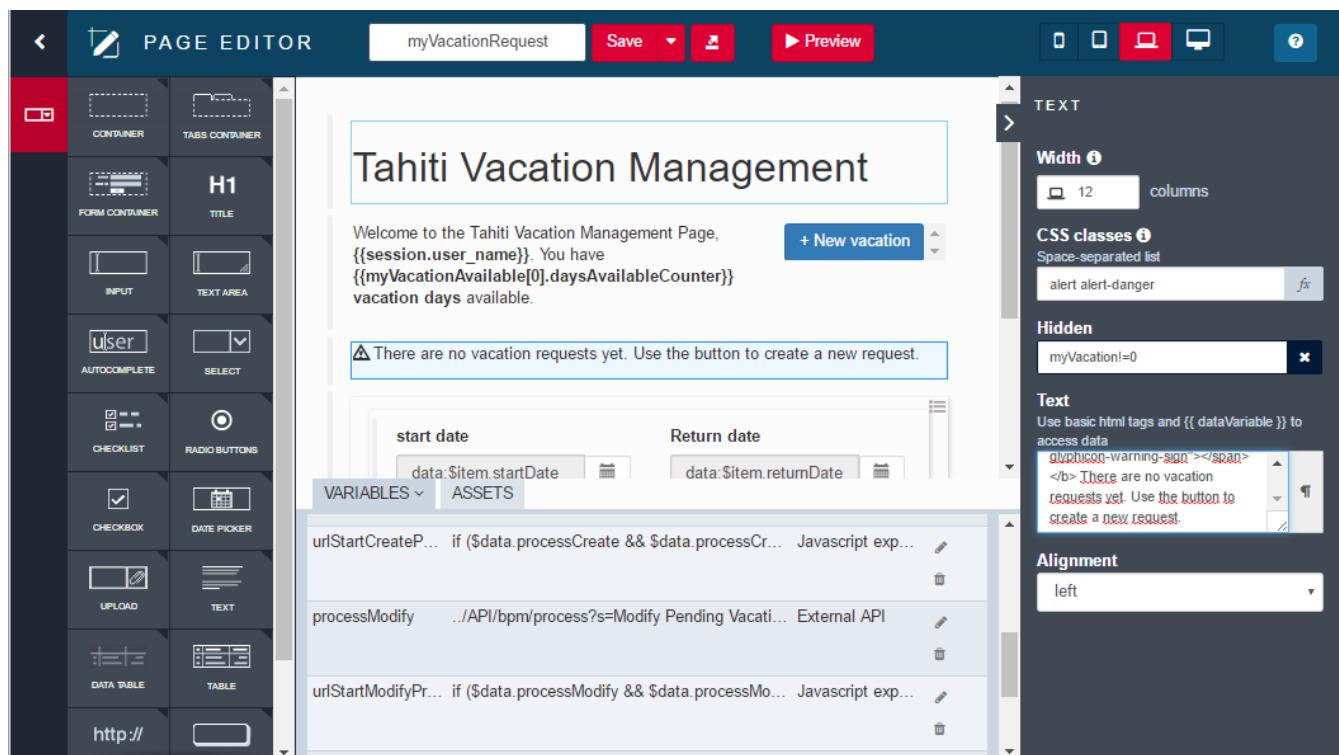


Figure 64. Text warning

Main Container widget

- Configure the **Main Container widget** properties this way:

Property name	Property value
Hidden (<i>bound to an expression or variable</i>)	!myVacation
Collection	myVacation

- Now, when there are no employee vacation requests we warn the end-users they need to use the

Link widget to create a new vacation request.

Chapter 6. Create a new REST API Extension

6.1. Prerequisites

You must have Bonita BPM Studio SP installed and started. An Internet connection is mandatory.

6.2. Objectives

The goal of this exercise is to become familiar with developing REST API Extensions with Bonita BPM Studio. It will permit a manager to get the list of her team subordinates vacations.

6.3. Instructions

In this exercise we will create an extension that will consolidate the information for future util view manager.

- Create a new REST API extension
- Implement the business logic
- Implement test
- Deploy
- Execute
- Create a page to use the extension
- Add the page to the application

Optional

- Advanced tests

6.4. Correction

6.4.1. Create new REST API extension

You can create a new REST API extension from the studio menu **Development / REST API Extension / New ...** as name put Tahiti REST API extension

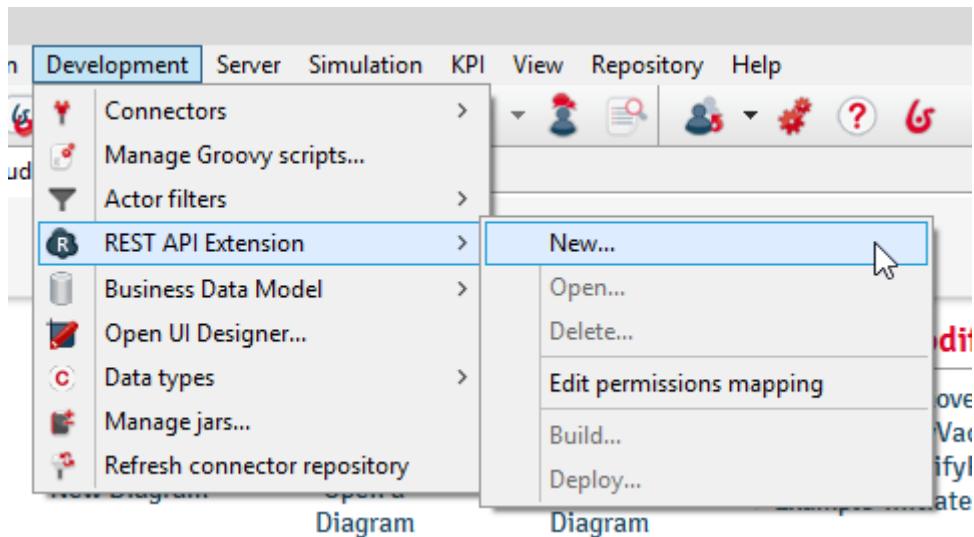


Figure 65. REST API Extension menu



For the first creation, you must have an Internet connection. The studio use Maven to install all additional component

6.4.2. Configure the new project

Your new REST API Extension needs to be configured.

Please indicate the following parameters.

Field	Value
Name	Tahiti REST API extension
Description	REST API extension that provide API to list users that are managed by a manager
Package	com.tahiti.rest.api
Project name	tahitiUsersOfManagerExtension
Version	1.0.0-SNAPSHOT

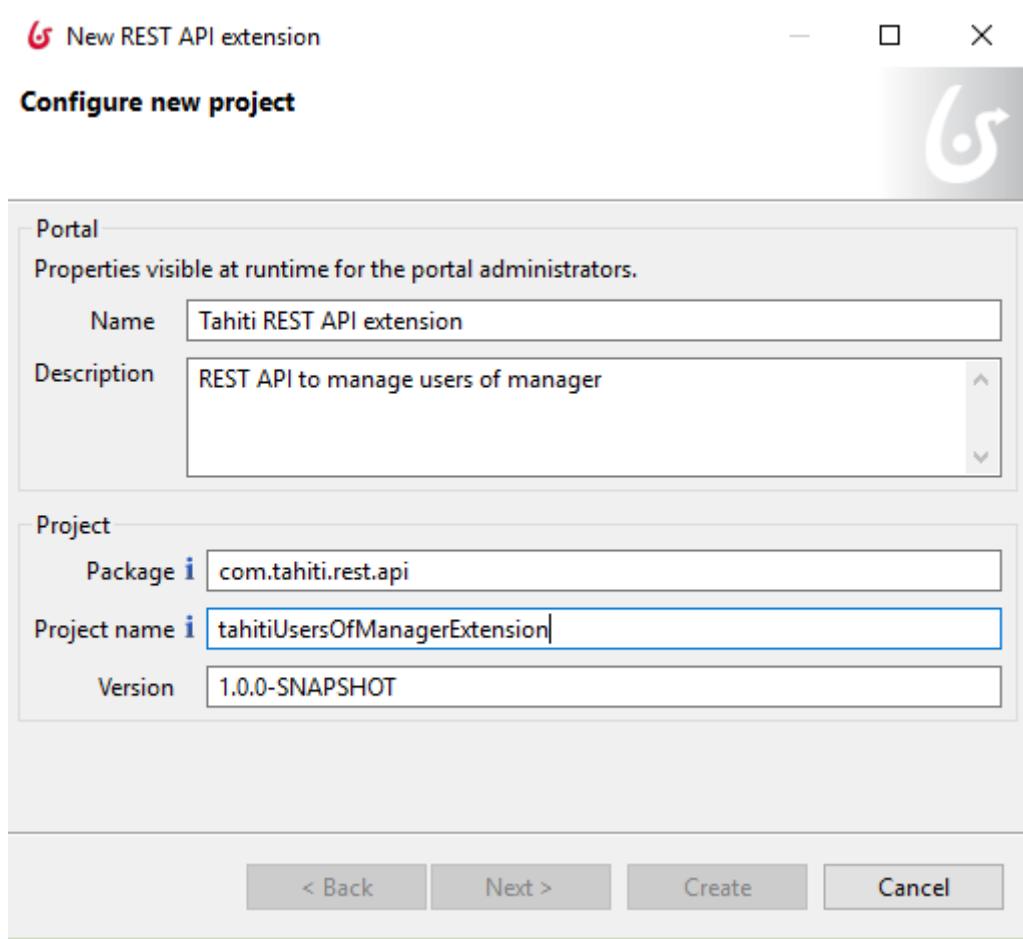


Figure 66. Description

Click on **Next** when you finish.

6.4.3. Configuration advanced

Indicate the following parameters.

Field	Value
Path template	tahitiUsersOfManager
Add BDM dependencies	false
Permissions	task_visualization

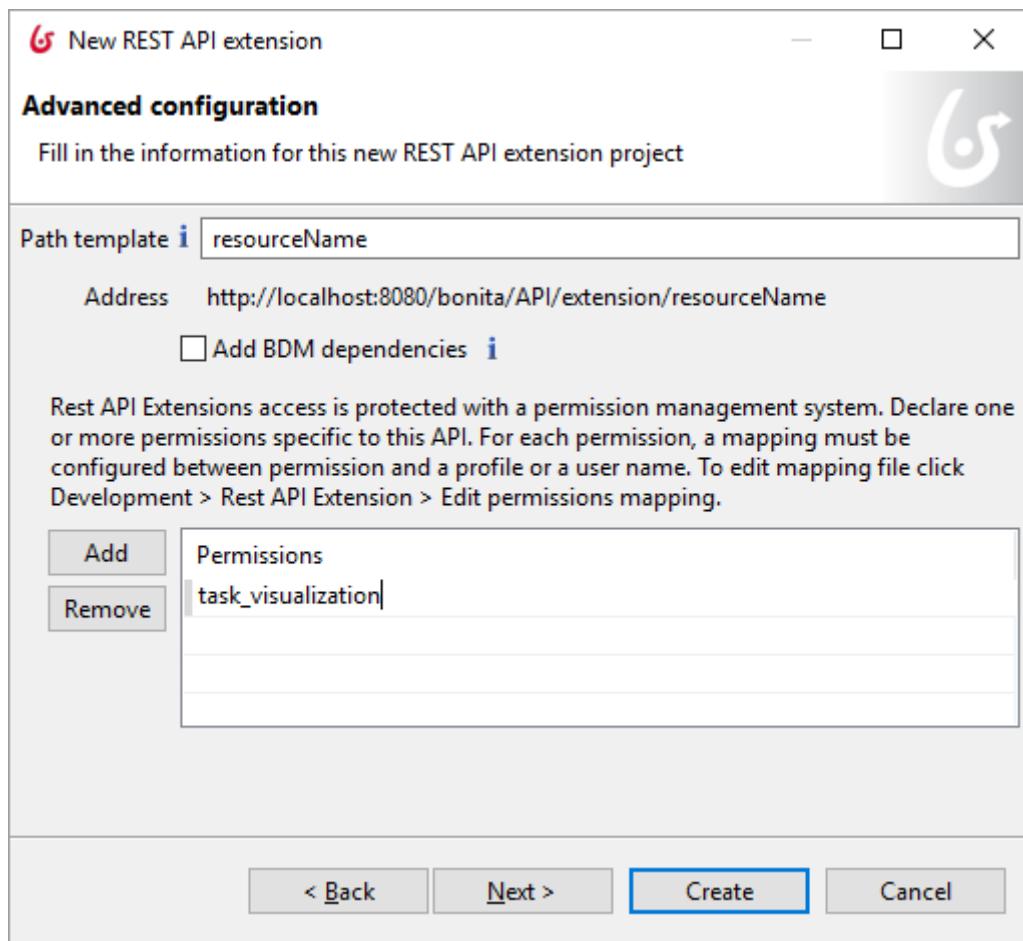


Figure 67. Permission

Click on **Next** when you finish.

6.4.4. Configuration parameters

For the purpose of the exercise we just keep the parameters p and c

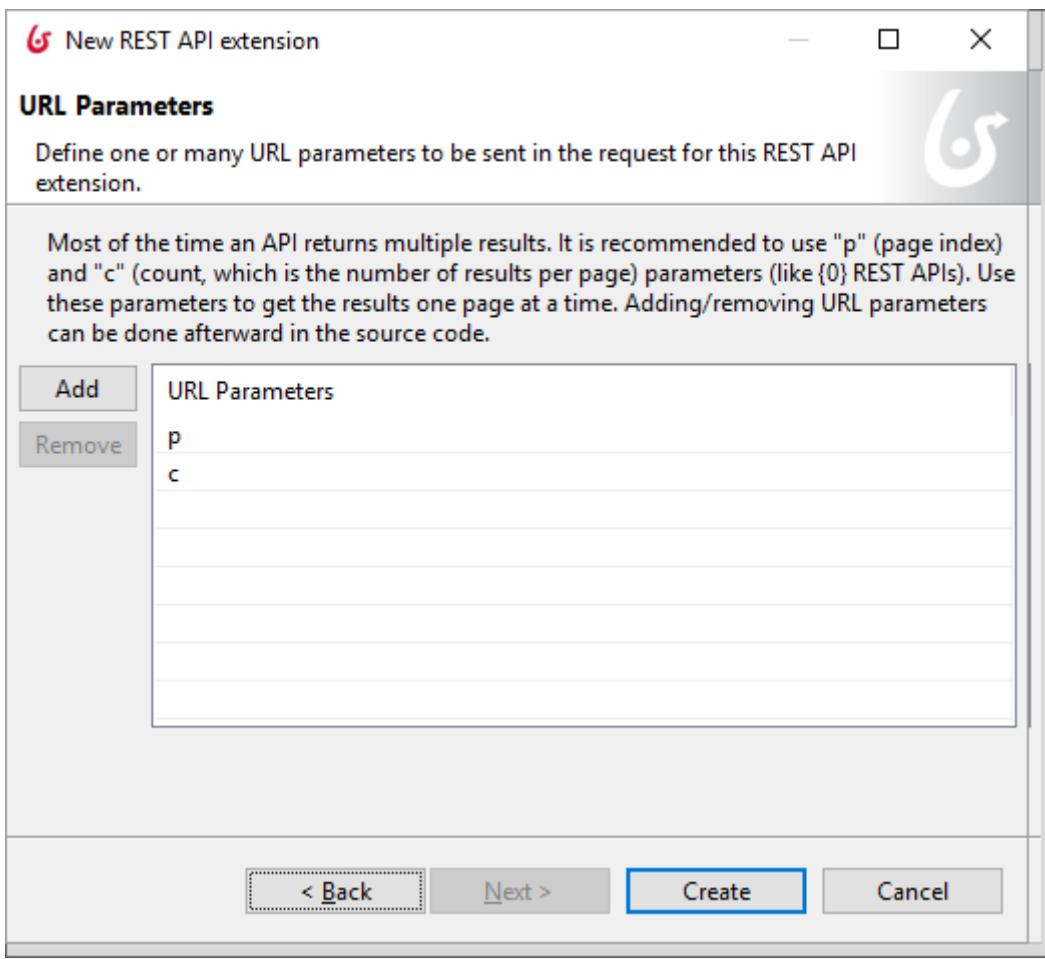


Figure 68. Rest parameters

Click on **Create** when you finish.

6.4.5. Screen start

When the configuration is finished, the studio will open the "Rest API development perspective" on your project REST API extension.

On the left panel you find the tree of your project, it's been auto-generated by the studio.

REST API Extensions

```

REST API Extensions
  tahitiUsersOfManagerExtension
    src/main/groovy
      com.tahiti.rest.api
        Index.groovy
    src/main/resources
      configuration.properties
      page.properties
    src/test/groovy
    src/test/resources
    JRE System Library [JavaSE-1.7]
    Maven Dependencies
    Referenced Libraries
    Groovy DSL Support (Disabled)
    .settings
    src
    target
    .classpath
    project
    content.xml
    pom.xml

Index.groovy
#The technical name of the REST API extension
#Must be URL compliant (alpha-numeric characters with no whitespace) and be prefixed by "custompage_"
name=custompage_tahitiUsersOfManagerExtension

#Name displayed in the Portal
displayName=Tahiti usersOfManager REST API

#Description displayed in the Portal
description=REST API to manage resourceName

#Must be apiExtension for a REST API extension
contentType=apiExtension

#Declare at least one API extension here (comma-separated list)
apiExtensions=tahitiUsersOfManagerExtension

#For each declared API extension, specify the
#following properties: method,pathTemplate,classFileName and permissions
|
#Specify one HTTP verb from GET|POST|PUT|PATCH|DELETE|HEAD|OPTIONS|TRACE
#GET is the recommended value for a REST API extension.
#Write operations should be performed by a process.
tahitiUsersOfManagerExtension.method=GET

#Define the URL path template
#Resulting URL: ../API/extension/resourceName
tahitiUsersOfManagerExtension.pathTemplate=tahitiUsersOfManager

#Declare the associated RestAPIController Groovy file
tahitiUsersOfManagerExtension.classFileName=com/tahiti/rest/api/Index.groovy

#Declare the permissions list (comma-separated list)
#For each permission declared, you must map it either to a profile (for example User, Administrator, or a custom profile) or to a specific user.
#Edit ${bonita.home}/client/tenants/${tenant_id}/conf/custom-permissions-mapping.properties.
#For example: user|john=[task_visualization] or profile|user=[task_visualization]
#In production, restart the web server to apply the changes.
#In the studio, you just have to logout/login to apply the changes.
tahitiUsersOfManagerExtension.permissions=task_visualization

```

Figure 69. Configuration file

We don't need to modify the file "page.properties" you can close it.

6.4.6. Implement the business logic

An important part of the work is of course to implement the logic of our REST API ext, it's on the file "Index.groovy".

Here you are the default page created by the Studio that we are going to modify.

```
class Index implements RestApiController {  
    private static final Logger LOGGER = LoggerFactory.getLogger(Index.class)  
  
    @Override  
    RestApiResponse doHandle(HttpServletRequest request, RestApiResponseBuilder responseBuilder, RestAPIContext context) {  
        // To retrieve query parameters use the request.getParameter(..) method.  
        // Be careful, parameter values are always returned as String values  
  
        // Retrieve p parameter  
        def p = request.getParameter "p"  
        if (p == null) {  
            return buildResponse(responseBuilder, HttpServletResponse.SC_BAD_REQUEST, """{"error" : "the parameter p is missing"}""")  
        }  
  
        // Retrieve c parameter  
        def c = request.getParameter "c"  
        if (c == null) {  
            return buildResponse(responseBuilder, HttpServletResponse.SC_BAD_REQUEST, """{"error" : "the parameter c is missing"}""")  
        }  
  
        // Here is an example of how you can retrieve configuration parameters from a properties file  
        // It is safe to remove this if no configuration is required  
        Properties props = loadProperties("configuration.properties", context.resourceProvider)  
        String paramValue = props["myParameterKey"]  
  
        /*  
         * Execute business logic here  
         *  
         * Your code goes here  
         *  
         */  
  
        // Prepare the result  
        def result = [ "p" : p , "c" : c , "myParameterKey" : paramValue ]  
  
        // Send the result as a JSON representation  
        // You may use buildPagedResponse if your result is multiple  
        return buildResponse(responseBuilder, HttpServletResponse.SC_OK, new JsonBuilder(result).toPrettyString())  
    }  
}
```

Figure 70. Index.groovy

The idea is to return the list of subordinates of a specific user. The extension will return a json with the users (list of subordinates).

You can remove the content of the doHandle method.

In order to better understand, we will gradually introduce code blocks step by step.

Retrieve input parameters

We have three input parameters:

pagination parameters p (page number) and c (number of elements to return) and the id of the manager

We retrieve these parameters in this way:

```

int p = 0;
int c = 10;

try{
    p = Integer.parseInt(request.getParameter("p"));
    c = Integer.parseInt(request.getParameter("c"));
}
catch(Exception e){
    LOGGER.info("Request parameters p and c should be numbers");
}

```

As you see in the code if p and c are not numbers we just log it and we use default values.

The id of the manager, managerUserId is retrieved from the session:

```

def apiSession = context.getApiSession();
def managerUserId = apiSession.getUserId();

```

Retrieve user subordinates

This is the core of the Rest extension. First of all we retrieve the IdentityApi then we use the search Bonita api classes and methods to retrieve our users. They are stored in the variable users



While coding, in order to get the import done, use autocompletion clicking CTRL + SPACE to have the list of suggestions.

```

def contextApiClient = context.getApiClient();
// Get a reference to IdentityAPI to search for users managed by current user
def identityAPI = contextApiClient.getIdentityAPI();

def searchBuilder = new SearchOptionsBuilder(p*c, c);

//We filter the user for MANAGER_USER_ID
searchBuilder.filter(UserSearchDescriptor.MANAGER_USER_ID, managerUserId)

def users = identityAPI.searchUsers(searchBuilder.done()).getResult()

LOGGER.info "Users managed by: " + managerUserId + " are: " + users
// If current user manage at least one user he/she is a manager

def userListNumber = 0;
if(!users.empty)
    userListNumber = users.size();

```

Result creation and pagination

Now we can build the paged response in this way:

```
return buildPagedResponse(responseBuilder, new JsonBuilder(users).toPrettyString(),p,c, new Long(userListNumber))
```

Check the imports

The index.groovy class is terminated, check that the class import are correctly done. You should have the following import list :

```
import groovy.json.JsonBuilder

import javax.servlet.http.HttpServletRequest
import javax.servlet.http.HttpServletResponse

import org.apache.http.HttpHeaders
import org.bonitasoft.web.extension.ResourceProvider
import org.bonitasoft.web.extension.rest.RestApiResponse
import org.bonitasoft.web.extension.rest.RestApiResponseBuilder
import org.slf4j.Logger
import org.slf4j.LoggerFactory

import com.bonitasoft.web.extension.rest.RestAPIContext
import com.bonitasoft.web.extension.rest.RestApiController

import org.bonitasoft.engine.session.APISession
import org.bonitasoft.engine.search.SearchOptionsBuilder
import org.bonitasoft.engine.api.TenantAPIAccessor
import org.bonitasoft.engine.identity.UserSearchDescriptor
```

6.4.7. Implement the test

This environment includes test features, so by default a test has been delivered. We provide here an example of test that should be included in the IndexTest.groovy file. If you are familiar with tests, this should sound logic to you.

When business logic changes, present test should change too.

```
package com.tahiti.rest.api;

import groovy.json.JsonSlurper
import groovy.mock.interceptor.MockFor;

import javax.servlet.http.HttpServletRequest

import org.bonitasoft.engine.identity.User
```

```

import org.bonitasoft.engine.identity.UserSearchDescriptor;
import org.bonitasoft.engine.search.SearchOptionsBuilder
import org.bonitasoft.engine.search.impl.SearchResultImpl;
import org.bonitasoft.engine.session.APISession;
import org.bonitasoft.web.extension.ResourceProvider
import org.bonitasoft.web.extension.rest.RestApiResponseBuilder

import spock.lang.Specification

import com.bonitasoft.engine.api.APIClient;
import com.bonitasoft.engine.api.IdentityAPI;
import com.bonitasoft.web.extension.rest.RestAPIContext

/**
 * @see http://spockframework.github.io/spock/docs/1.0/index.html
 */
class IndexTest extends Specification {

    // Declare mocks here
    // Mocks are used to simulate external dependencies behavior
    def HttpServletRequest httpRequest = Mock()
    def ResourceProvider resourceProvider = Mock()
    def RestAPIContext context = Mock()
    def APIClient apiClient = Mock();
    def APISession apiSession = Mock();
    def IdentityAPI identityAPI = Mock();

    /**
     * You can configure mocks before each tests in the setup method
     */
    def setup(){
        // Simulate access to configuration.properties resource
        context.resourceProvider >> resourceProvider
        resourceProvider.getResourceAsStream("configuration.properties") >>
IndexTest.class.getClassLoader.getResourceAsStream("testConfiguration.properties")
        context.apiClient >> apiClient
        context.apiSession >> apiSession
        apiSession.userId >> 1L
        apiClient.identityAPI >> identityAPI
    }

    public void should_return_an_empty_userlist_when_logged_user_is_not_manager() {
        given: "a RestAPIController"
        def index = new Index()
        httpRequest.getParameter("p") >> 0
        httpRequest.getParameter("c") >> 100
        identityAPI.searchUsers(new SearchOptionsBuilder(0, 100)
            .filter(UserSearchDescriptor.MANAGER_USER_ID, apiSession.userId).done())
        >> new SearchResultImpl<User>(0,[])
    }
}

```

```

when: "Invoking the REST API"
def apiResponse = index.doHandle(httpRequest, new RestApiResponseBuilder(),
context)

then: "A JSON representation is returned in response body"
def jsonResponse = new JsonSlurper().parseText(apiResponse.response)
// Validate returned response
apiResponse.HttpStatus == 200
jsonResponse == []
}

public void should_return_a_userlist_when_logged_user_is_a_manager() {
given: "a RestAPIController"
def index = new Index()
httpRequest.getParameter("p") >> 0
httpRequest.getParameter("c") >> 100
identityAPI.searchUsers(new SearchOptionsBuilder(0, 100)
    .filter(UserSearchDescriptor.MANAGER_USER_ID, apiSession.userId).done())
>> new SearchResultImpl<User>(2,[Mock(User),Mock(User)])

when: "Invoking the REST API"
def apiResponse = index.doHandle(httpRequest, new RestApiResponseBuilder(),
context)

then: "A JSON representation is returned in response body"
def jsonResponse = new JsonSlurper().parseText(apiResponse.response)
// Validate returned response
apiResponse.HttpStatus == 200
jsonResponse.size() == 2
}
}

```

6.4.8. Deploy

To test the interface with the embedded platform, we have the possibility to deploy directly from the Bonita BPM Studio.

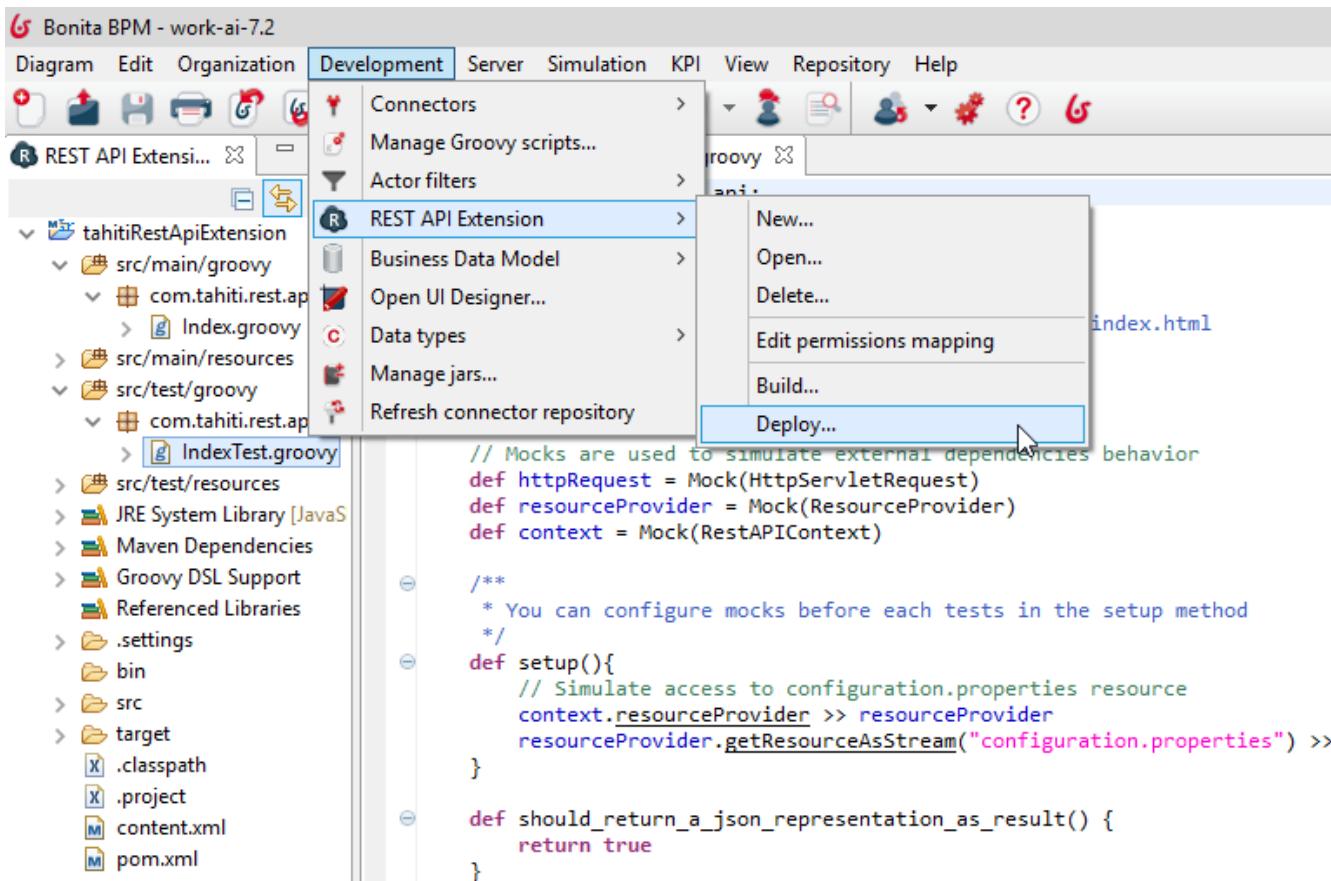


Figure 71. Deploy



If by any chance you have a failure at this stage, we must start from the beginning.

6.4.9. Execute

As a reminder the purpose of the REST API EXT is to get users of the manager. We must login in the portal with a manager.

Helen Kelly Login: helen.kelly Password: bpm

We can take the opportunity to change the profile to "Administrator" and see the resources in our REST API is present Ext.

Welcome: Helen Kelly | Administrator |

BPM | Organization | Resources | Applications | Portal | License |

DELETE | **EDIT** | **EXP**

Tahiti REST API extension

REST API extension that provide API to list vacation request submitted by employee managed by provided manager id

Created on: 02/23/2016 12:05 PM
Updated on: 02/23/2016 1:57 PM
Created by: Walter Bates
Updated by: Walter Bates

Name for the URL
custompage_tahitiRestApiExtension

Figure 72. Resource is deployed

As a reminder we said our REST API Ext has the following path: **tahitiUsersOfManager**, so the url to provide is: <http://localhost:8080/bonita/API/extension/tahitiUsersOfManager?p=0&c=10>

First, log in the browser as Helen.Kelly

Welcome to Bonita BPM Portal

Login form

helen.kelly

...

LOGIN

Figure 73. Log as Helen.Kelly

Then, using the REST CLIENT, check the call

The screenshot shows the RESTClient interface. At the top, there are tabs for File, Authentication, Headers, View, Favorite Requests, Setting, and a title bar with 'RESTClient'. Below the header, the URL is set to `http://localhost:8080/bonita/API/extension/tahitiUsersOfManager?p=0&c=10`. The method is set to GET. On the right, there is a red 'SEND' button. A 'Body' section labeled 'Request Body' contains the text 'Request Body'. Below the request area, a 'Response' section is expanded. It has tabs for Response Headers, Response Body (Raw), Response Body (Highlight) (which is selected), and Response Body (Preview). The response body is a JSON array with one element:

```
[{"password": "", "userName": "april.sanchez", "jobTitle": "Compensation specialist", "title": "Mrs", "managerUserName": null, "createdBy": -1, "creationDate": "2016-04-12T18:58:14+0000", "managerUserId": 3, "firstName": "April", "id": 2, "iconName": null, "lastName": "Sanchez"}]
```

Figure 74. Run the REST CLIENT

Just copy and paste this url on a new tab of your browser.

```
[  
  {  
    "password": "",  
    "userName": "april.sanchez",  
    "jobTitle": "Compensation specialist",  
    "title": "Mrs",  
    "managerUserName": null,  
    "createdBy": -1,  
    "creationDate": "2016-03-22T15:51:54+0000",  
    "managerUserId": 147,  
    "firstName": "April",  
    "id": 146,  
    "iconName": null,  
    "lastName": "Sanchez",  
    "enabled": true,  
    "iconPath": null,  
    "lastUpdate": "2016-03-22T15:51:54+0000",  
    "lastConnection": null  
  },  
  {  
    "password": "",  
    "userName": "walter.bates",  
    "jobTitle": "Human resources benefits",  
    "title": "Mr",  
    "managerUserName": null,  
    "createdBy": -1,  
    "creationDate": "2016-03-22T15:51:54+0000",  
    "managerUserId": 147,  
    "firstName": "Walter",  
    "id": 148,  
    "iconName": null,  
    "lastName": "Bates",  
    "enabled": true,  
    "iconPath": null,  
    "lastUpdate": "2016-03-22T15:51:54+0000",  
    "lastConnection": "2016-03-25T08:53:35+0000"  
  }  
]
```

Result is



The screenshot shows a browser window with the URL `localhost:8080/bonita/API/extension/vacationRequest`. The page displays a JSON response from a REST API. The JSON object contains two main sections: `"employeesVacationRequests"` and `"employeesVacationAvailable"`.

```
{  
  "isManager": true,  
  "employeesVacationRequests": [  
    {  
      "firstName": "Walter",  
      "lastName": "Bates",  
      "startDate": "2016-02-22T00:00:00+0000",  
      "returnDate": "2016-02-23T00:00:00+0000",  
      "numberOfDays": 1,  
      "status": "pending",  
      "taskId": 15  
    },  
    {  
      "firstName": "Walter",  
      "lastName": "Bates",  
      "startDate": "2016-02-22T00:00:00+0000",  
      "returnDate": "2016-02-23T00:00:00+0000",  
      "numberOfDays": 1,  
      "status": "pending",  
      "taskId": 23  
    }  
  "employeesVacationAvailable": [  
    {  
      "firstName": "April",  
      "lastName": "Sanchez",  
      "daysAvailableCounter": 10  
    },  
    {  
      "firstName": "Walter",  
      "lastName": "Bates",  
      "daysAvailableCounter": 7  
    }  
}
```

Figure 75. Execute the REST

6.4.10. View results on a Page

In order to see the results of our extension we need to create a simple **application page**. It will have just two widgets:

- a **Title widget** to display the name of our page: Team members
- a **Data table** to list all the manager team members

Title of our Page

Create a page and name it **Tahiti Page**.

Create new artifact

Type

- Application page
- Process form ?
- Application layout
- Custom widget
- Fragment

Name

Create **Cancel**

Figure 76. Create the page

Add a Title widget to the whiteboard and fill in their properties.

Property name	Property value
Text	Team members
Title level	Level 1
Alignment	left

Add a Data table widget

Property name	Property value
Headers	firstName, lastName, jobTitle
Data source	Bonita API
URL	./API/extension/tahitiUsersOfManager
Columns key	firstName, lastName, jobTitle

Page are

Figure 77. Content of the Tahiti page

You can preview your page with the button "Preview" and check that everything works fine. Attention : first log as Helen.Kelly in the browser;

firstName	lastName	jobTitle
April	Sanchez	Compensation specialist
Walter	Bates	Human resources benefits

Figure 78. List of persons of the team

Every time a manager login he will see his team members, if you login with a user that is not a manager you will not see any users.

Once the page is created you can add it to the application. Do you remember how to do it?

If not, just check the previous exercise.

Chapter 7. Create a custom page for the Managers

7.1. Prerequisites

Having done the previous exercises

7.2. Objectives

The goal of this exercise is to create a custom page for the Managers to be used in the Vacation Management application. The custom page will be created using the UI Designer as seen in the previous exercise. This custom page will:

- show the **vacation available** from all manager employees
- show the **vacation requests** from all manager employees
- display a **review link** to access selected vacation request validation form

Optional

- Display a bar chart with manager employees and their available vacation days
- Display a warning text only when logged user is not a manager

7.2.1. Overview

Team Vacation



⚠ You need to be a manager to access this Page.

Vacation available	Vacation requests	+			
First name	Last name	Start date	Number of days	Return date	Status

[Review selected vacation request](#)

Figure 79. Overview

7.3. Instructions

- Create a new page
- Create all the data needed for your page (API variables to load the BDM,...)
- Drag and Drop the widgets used on your page, you will need Title, Tabs Container, Table and Button widgets
- Bind all your data to your widgets
- Test your page
- Add your page into the Vacation Management application
- Test your application page.

7.4. Correction

This section provides details about steps to follow to create a page using best practices.

7.4.1. Create a new empty page

- Run the UI Designer either by clicking on UI Designer icon on the Studio
- On the UI Designer home page, click on "+ Create" button and be sure that the Type "Application page" is checked. Name your page **myVacationManager** and click on Create button.

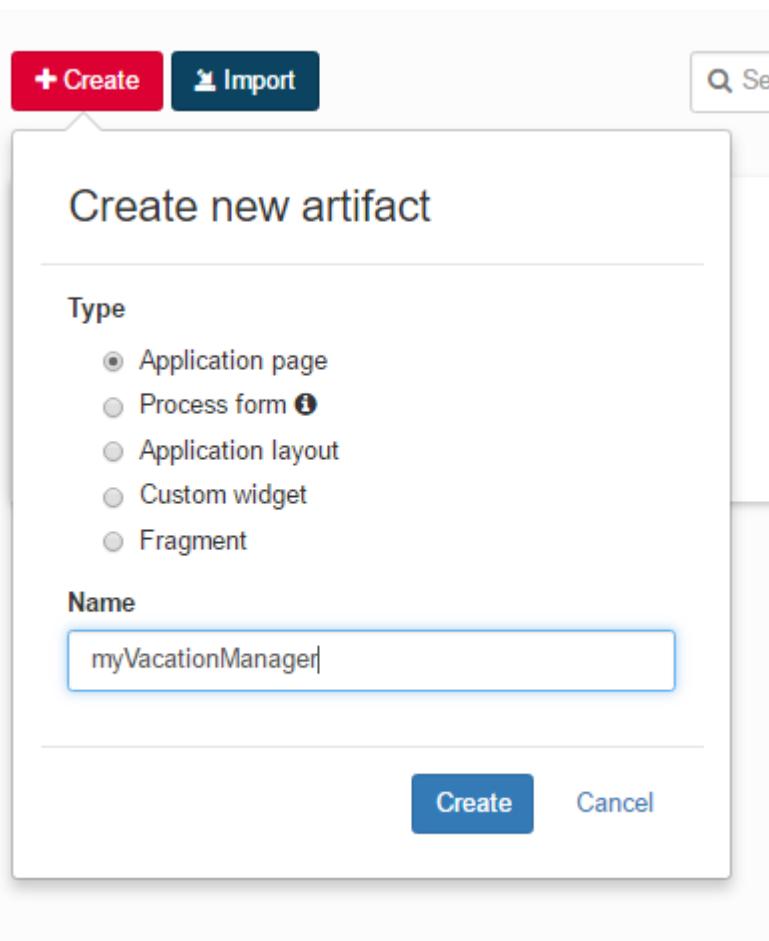


Figure 80. Create the page

- Your page is created

7.4.2. Rest API Extension

We need to retrieve the information of the vacation days available for each team member. We will use a provided Rest API Extension for that, called `tahitiRestApiExtension`.

Download it from provided files and import the `.bos` file as it were a new process.

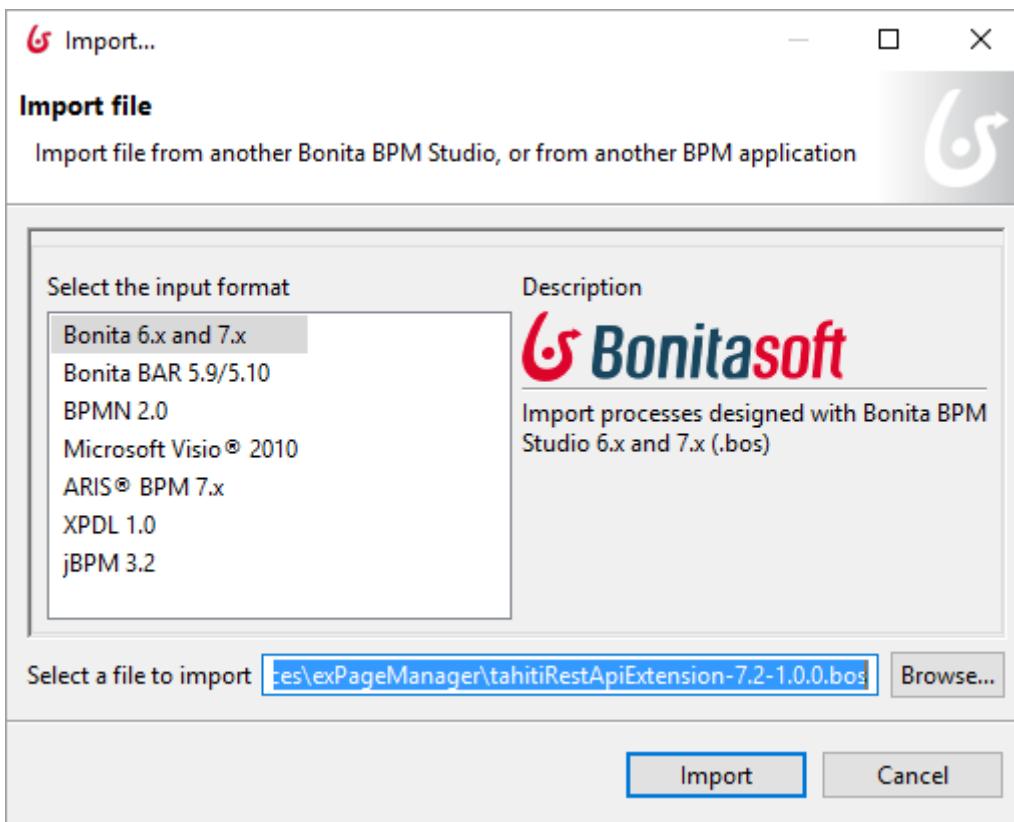


Figure 81. Import the REST API

You can then deploy it on the portal:

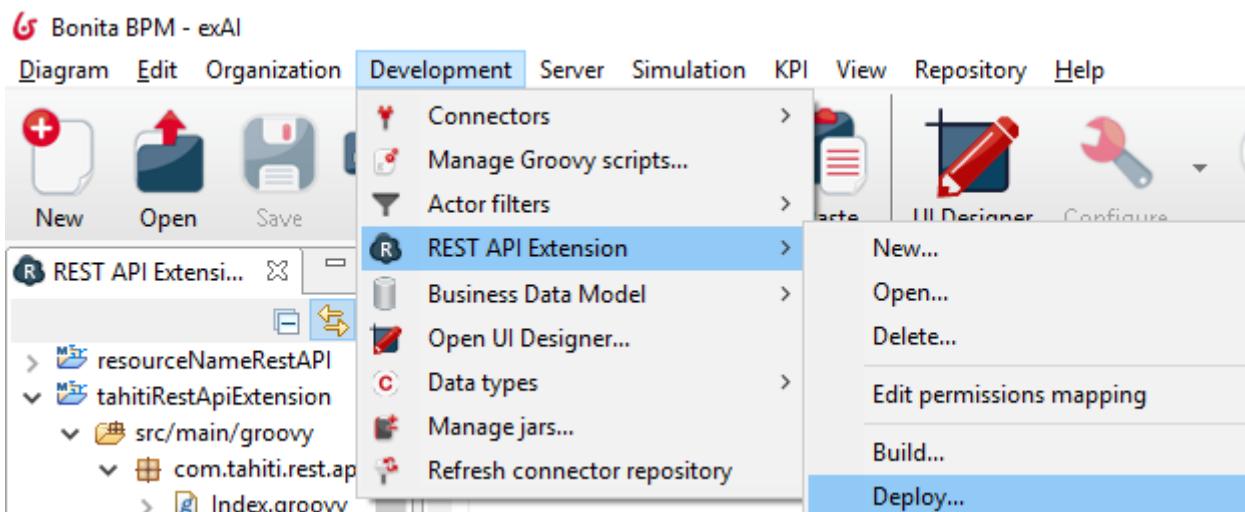


Figure 82. Deploy the RESTAPI

7.4.3. Create the data

For your page communication and logic you will need some data. Below the list of data you need to create.

Name	Type	Content
managerTeamInfo	External API/API/extension/vacationRequest
teamSelectedRow	JSON	{}

Result is

VARIABLES		ASSETS
Create a new variable		?
Name	Value	Type
managerTeamInfo	..//API/extension/vacationRequest	External API
teamSelectedRow	{}	JSON

Figure 83. Variables created

More about created data

managerTeamInfo

This API call is used to load all the Vacation Request business data

teamSelectedRow

This empty JSON will be useful to take data from the selected row in the Table widget

7.4.4. Build your page

Now we can start to design "graphically" our page. We are going to add widgets, configure them and bind them to related data. This will be the last step of your page creation.

Before getting started, you should design your page mock-up. As you have all the variables inside and outside your page already created, you can know which widgets you will use. You can also take the opportunity to think about the widgets placement and the global look'n'feel of your page.

We are going to need:

- a **Title widget** to display the name of our page
- a **Tabs container widget** to show Vacation available tab and Vacation requests tab
 - a **Table widget** to show the Vacation available of the manager employees
 - a **Table widget** to show the Vacation requests of the manager employees
 - a **Link widget** as a button to call the task "Review request" from the NewVacationRequest process, once selected a request in the previous Table.

Widget Title

Add a Title widget to the whiteboard and fill in their properties.

Property name	Property value
Text	Team Vacation

Property name	Property value
Title level	Level 1
Alignment	left

Tabs container widget

Add a Tabs container widget below the Title widget. Click on "+" on the top right corner of the Tabs container widget to add a new tab. Fill in their properties.

Property name	Property value
Title (on the first tab)	Vacation available
Title (on the second tab)	Vacation requests

Once you select a tab, you can drag and drop widgets inside.

Drag and drop 2 Table widgets, one for each tab, a Link widget into the second tab and configure their properties this way:

Vacation available tab - Table widget

Drag and drop a Table widget inside the "Vacation available" tab and use these properties:

Property name	Property value
Headers	First name, Last name, Days available
Content (<i>bound to an expression or variable</i>)	managerTeamInfo.employeesVacationAvailable
Column keys	firstName, lastName, daysAvailableCounter

Headers property display the name of the Table widget headers.

Column keys property belong to the Content attributes.

Content property is a **Dynamic Value** type. You can use a dynamic value field to specify a constant (by default) or an expression. Click on the expression icon 'fx' to switch from constant to expression and start typing the name of your variable (in this case **managerTeamInfo**) and select it from the autocomplete selector. We want to display the **employeesVacationAvailable** attribute that is a JSON list If you run the REST API, the result is this one:

```

managerTeamInfo: {
    "isManager": true,
    "employeesVacationRequests": [
        { "firstName": "Walter", "lastName": "Bates", "startDate": "2016-02-12T00:00:00+0000", "returnDate": "2016-02-13T00:00:00+0000", "numberOfDays": 1, "status": "processing cancellation", "taskId": null }, { "firstName": "Walter", "lastName": "Bates", "startDate": "2016-02-12T00:00:00+0000", "returnDate": "2016-02-14T00:00:00+0000", "numberOfDays": 2, "status": "approved", "taskId": null }, { "firstName": "Walter", "lastName": "Bates", "startDate": "2016-02-15T00:00:00+0000", "returnDate": "2016-02-17T00:00:00+0000", "numberOfDays": 2, "status": "cancelled", "taskId": null }
    ],
    "employeesVacationAvailable": [
        { "firstName": "April", "lastName": "Sanchez", "daysAvailableCounter": 10 },
        { "firstName": "Walter", "lastName": "Bates", "daysAvailableCounter": 7 }
    ]
}

```

So we use **managerTeamInfo.employeesVacationAvailable** JSON list for iteration.

Result of the tab is

Figure 84. Vacation available tab

Vacation request tab - Table widget

Drag and drop a Table widget inside the "Vacation request" tab and use these properties:

Property name	Property value
Headers	First name, Last name, Start date, Number of days, Return date, Status
Content (<i>bound to an expression or variable</i>)	managerTeamInfo.employeesVacationRequests
Column keys	firstName, lastName, startDate date, numberOfDays, returnDate date, status
Selected Row	teamSelectedRow

Column keys property is using a list of attributes that match the Content list property. Attributes startDate and returnDate are using an AngularJS filter to show formated dates like this: **startDate | date**.

SelectedRow property is using a variable to store the value of the selected row in the Table widget.

Content property is a **Dynamic Value** type. You can use a dynamic value field to specify a constant (by default) or an expression. Click on the expression icon 'fx' to switch from constant to expression and start typing the name of your variable (in this case **managerTeamInfo**) and select it from the autocomplete selector. We want to display the **employeesVacationRequests** attribute that is a JSON list:

```
managerTeamInfo: {
    "isManager": true,
    "employeesVacationRequests": [
        { "firstName": "Walter", "lastName": "Bates", "startDate": "2016-02-12T00:00:00+0000", "returnDate": "2016-02-13T00:00:00+0000", "numberOfDays": 1, "status": "processing cancellation", "taskId": null },
        { "firstName": "Walter", "lastName": "Bates", "startDate": "2016-02-12T00:00:00+0000", "returnDate": "2016-02-14T00:00:00+0000", "numberOfDays": 2, "status": "approved", "taskId": null },
        { "firstName": "Walter", "lastName": "Bates", "startDate": "2016-02-15T00:00:00+0000", "returnDate": "2016-02-17T00:00:00+0000", "numberOfDays": 2, "status": "cancelled", "taskId": null }
    ],
    "employeesVacationAvailable": [
        { "firstName": "April", "lastName": "Sanchez", "daysAvailableCounter": 10 },
        { "firstName": "Walter", "lastName": "Bates", "daysAvailableCounter": 7 }
    ]
}
```

So we use **managerTeamInfo.employeesVacationRequests** JSON list for iteration.

Vacation request tab - Link widget

Drag and drop a Link widgets inside the "Vacation request" tab, below the Table widget, and configure their properties this way:

Property name	Property value
Hidden (<i>bound to an expression or variable</i>)	teamSelectedRow.status!="pending"
Text	Review selected vacation request
URL (<i>bound to an expression or variable</i>)	"/bonita/portal/form/taskInstance/" + teamSelectedRow.taskId
Frame	Current frame (_self)
Alignment	center
Style	primary

Hidden property is a **Dynamic Value** type and we want to evaluate an expression so, it needs to be bound to an expression or variable. We want to hide the Link whenever the status of the selected Table row is not "pending".

URL property is a **Dynamic Value** type. You can use a dynamic value field to specify a constant (by default) or an expression. Click on the expression icon '**fx**' to switch from constant to expression. In this case we want to display the URL String path + the selected row taskId. We can take the **taskId** from the selected row bound to **teamSelectedRow** data:

```
teamSelectedRow: { "firstName": "Helen", "lastName": "Kelly", "startDate": "2016-02-17T00:00:00+0000", "returnDate": "2016-02-18T00:00:00+0000", "numberOfDays": 1, "status": "pending", "taskId": 80005 }
```

So we use the path as a String "/bonita/portal/form/taskInstance/" and we add the teamSelectedRow.taskId

Result is

Figure 85. Tab Vacation request

7.4.5. Preview on different devices

Save your changes and use the Preview button to check how your page is going to look like. Use the different devices view button to test how widgets will be shown.

Tips

Each device view is directly related with each widget Width property. The value of this property can be from 1 to 12. This is based on Bootstrap, where a total of 12 cols divide the screen for each different device (extra-small, small, medium, large).

Play with the different widget Width values that fits your page design for the different devices using the Preview button.

7.4.6. Export your page

Once everything is ready on your page, click on the Export button. It will create a .zip file that we can use to import it as a resource into the Bonita Portal.

7.4.7. Add your page to the Vacation Management application

- Go to the Bonita Portal using the Administrator profile, **Resources** section.
- Import your page-myVacationManager.zip file clicking on "+ ADD" button on the top left side.

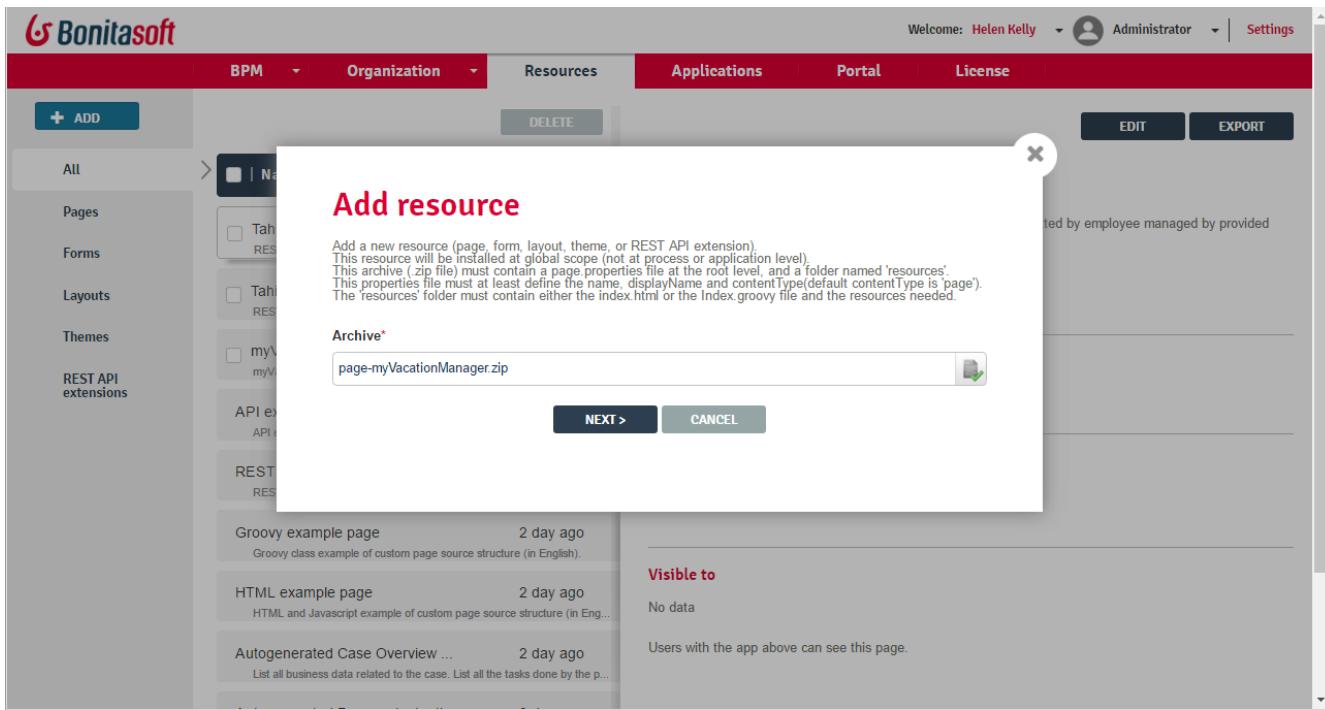


Figure 86. Add the page

- Also, you will need to import the REST API extension provided.
- Go to **Applications** section and click on the three dots icon action button of the Vacation Management application.

Let's add your Page and a new Navigation Menu for your application.

- Click on Pages "ADD" button and select **custompage_myVacationManager - myVacationManager page**. Also type **management** into the URL field.

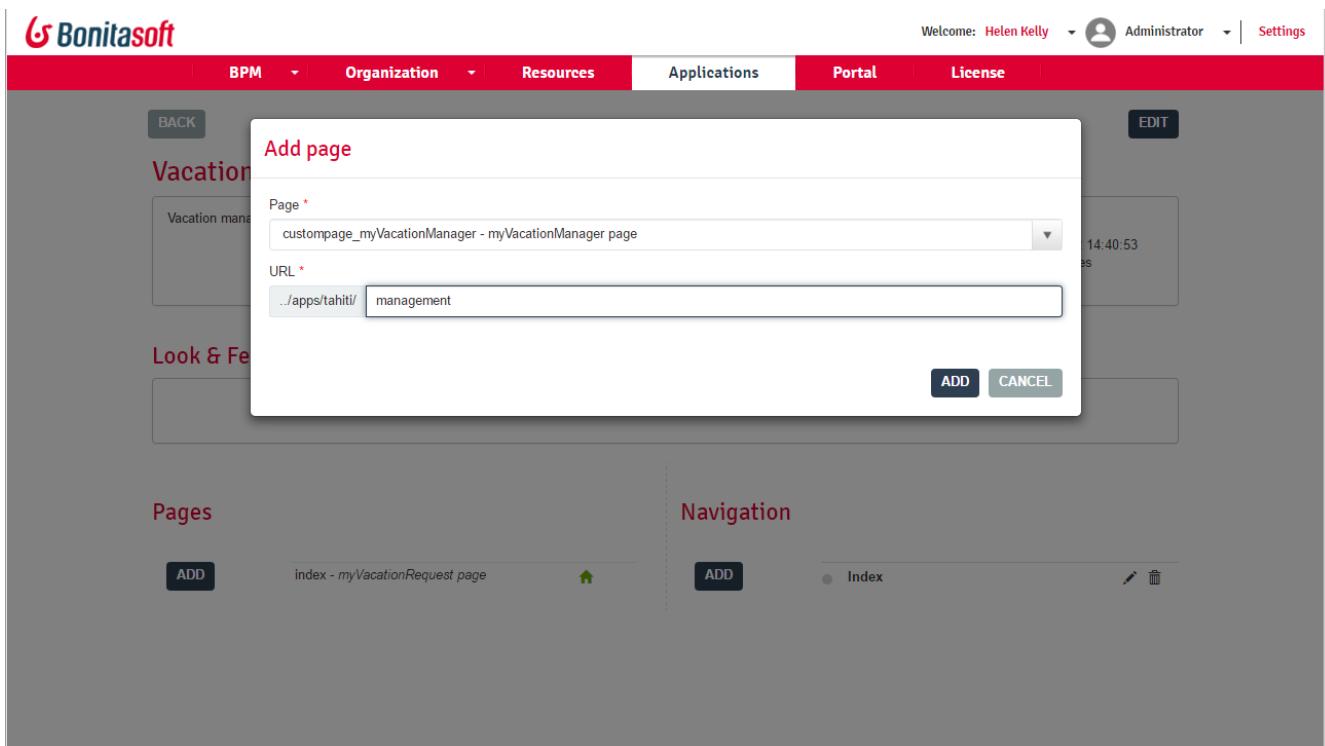


Figure 87. Add the page

- Click on Navigation "ADD" button, use the Name **Management** and select your page. Click on "ADD" button.

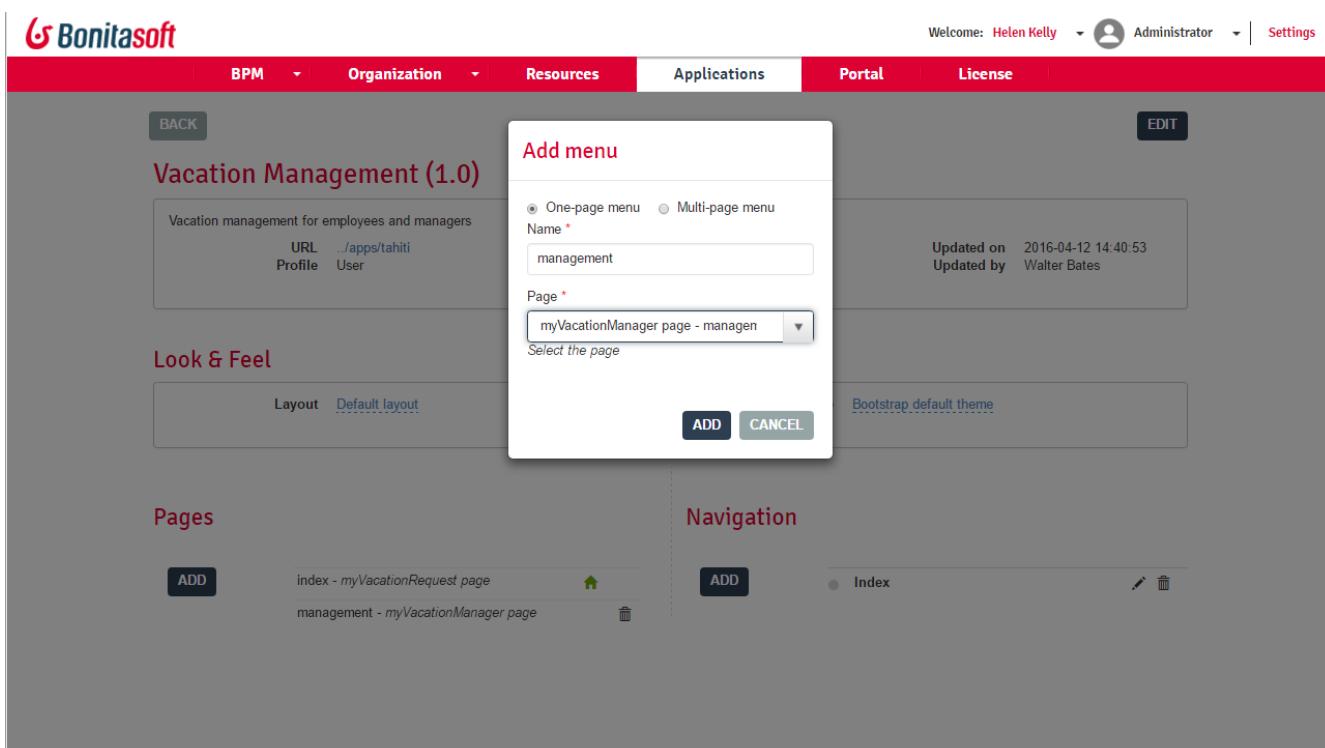


Figure 88. Add the menu

Now we can access our application clicking on the URL link or directly access to <http://localhost:8080/bonita/apps/tahiti/index/>

Be sure that you are logged in as a manager, for example as helen.kelly, manager of the HR group in ACME organization.

First name	Last name	Days available
April	Sanchez	10
Walter	Bates	-4

Figure 89. Vacation application

- Test the page Management.

7.5. Optional exercises

We can improve our Page adding some valuable features due to enhance the end-users experience.

Remember

- **Rename** your page using the Save as button
- **Export** your new page.
- **Import** your new page directly in Bonita Portal - **Resources** section, selecting the old page and clicking on **Edit** button. It will overwrite the old page with the new improved one and you could test it on your application directly.

7.5.1. Display a bar chart

Display a bar chart with manager employees and their available vacation days.

Data creation

Add these new variables:

Name	Type	Content
datas	Javascript expression	<pre>if(\$data.managerTeamInfo){ return \$data.managerTeamInfo.employeesVacationAvailable.map(function(selector){ return selector.daysAvailableCounter; }); }</pre>
labels	Javascript expression	<pre>if(\$data.managerTeamInfo){ return \$data.managerTeamInfo.employeesVacationAvailable.map(function(selector){ return selector.firstName + " " + selector.lastName; }); }</pre>

datas

this JS expression iterates managerTeamInfo.employeesVacationAvailable and returns a list of daysAvailableCounter.

labels

this JS expression iterates managerTeamInfo.employeesVacationAvailable and returns a list of firstName and lastname.

Chart widget

Drag and drop a Chart widget above title and use this properties:

Property name	Property value
Hidden (<i>bound to an expression or variable</i>)	!datas && !labels
Type	Bar
Data (<i>bound to an expression or variable</i>)	datas
Labels (<i>bound to an expression or variable</i>)	labels
Set labels	Vacation days left
Legend hidden	no

7.5.2. Display a warning text

Display a warning text only when logged user is not a manager.

One managerTeamInfo attribute is **isManager** of boolean type, so we can check it using an expression into a Hidden widget property. Remember that we need to switch to **bind an expressions or variables** clicking on the fx button close to yes/no radio buttons.

- Add a **Text widget** above the Title widget and configure its properties:

Property name	Property value
CSS classes	alert alert-danger
Hidden (<i>bound to an expression or variable</i>)	managerTeamInfo.isManager
Text	 You need to be a manager to access this Page.
Alignment	left

Container widget

- Add a Container widget below the Text widget and drag and drop the Title, Chart and Tab container widgets inside.
- Configure the **Container widget** properties this way:

Property name	Property value
Hidden (<i>bound to an expression or variable</i>)	!managerTeamInfo.isManager

- Now, when an employee user access the page Management will see the warning text and only a manager could access to see the content.

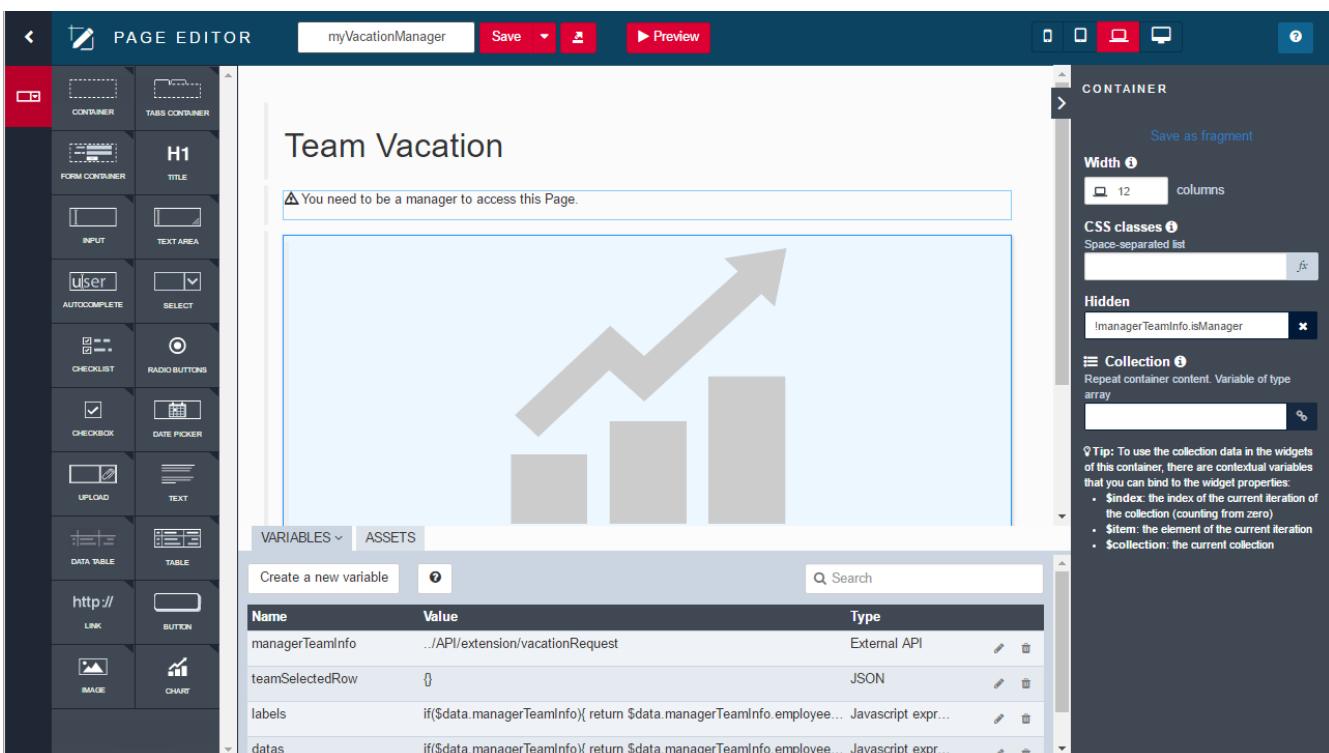


Figure 90. Optional design

Run the preview:

Team Vacation



Vacation available

Vacation requests

First name	Last name	Days available
April	Sanchez	10
Walter	Bates	6

Figure 91. Preview the application

Chapter 8. Create a new connector

8.1. Objective

The goal of this exercise is to build a new connector using the Connector Development Toolkit.

8.2. Prerequisites

To develop and test a Bonita BPM connector with the toolkit, you need the following software installed:

- Java 7 or later
- Maven 3.2.x or later

8.3. Instructions

- Download and install the development connector toolkit.
- Create connector definition and implementation.
- Use and test the new connector in a process.

8.4. Correction

8.4.1. Download the Development Connector Toolkit from customer portal.

Access the Customer portal, Downloads and the choose the Connector-Development-Toolkit

CUSTOMER PORTAL CASES DOWNLOADS LICENSES CONTENT PRODUCT FEEDBACK

Search All Bonitasoft Websites Search

Downloads

Request A New Download

Select a version on the list below and click on Access Download Page to access to the page with all resources for the chosen version.

Choose a product *

- Connector-Development-Toolkit
- BonitaBPM-Migration-Tool
- BonitaBPM-Subscription-7.0.0
- BonitaBPM-Subscription-6.5.3
- BonitaBPM-Subscription-6.5.2
- BonitaBPM-Subscription-6.5.1
- BonitaBPM-Subscription-6.5.0
- BonitaBPM-Subscription-6.4.2
- BonitaBPM-Subscription-6.4.1
- BonitaBPM-Subscription-6.4.0
- BonitaBPM-Subscription-6.3.9
- BonitaBPM-Subscription-6.3.8
- BonitaBPM-Subscription-6.3.7
- BonitaBPM-Subscription-6.3.6
- BonitaBPM-Subscription-6.3.5
- BonitaBPM-Subscription-6.3.4
- BonitaBPM-Subscription-6.3.3
- BonitaBPM-Subscription-6.3.2
- BonitaBPM-Subscription-6.3.1
- BonitaBPM-Subscription-6.3.0
- BonitaBPM-Subscription-6.2.6
- BonitaBPM-Subscription-6.2.5
- BonitaBPM-Subscription-6.2.4
- BonitaBPM-Subscription-6.2.3
- BonitaBPM-Subscription-6.2.2
- BonitaBPM-Subscription-6.2.1
- BonitaBPM-Subscription-6.2.0
- BonitaBPM-Subscription-6.1.2
- BonitaBPM-Subscription-6.1.1

© 2015 Bonitasoft, Inc. All rights reserved

[Twitter](#) [Facebook](#) [LinkedIn](#) [RSS](#) [Google+](#) [YouTube](#)

Figure 92. Download the toolkit

Unzip the toolkit file into a temporary installation folder, for example **c:\atelier supports**. Using a Command, do

```
cd C:\atelier supports\bonita-connector-development-toolkit-1.0  
install.bat
```

```

C:\WINDOWS\system32\cmd.exe - install
C:\atelier supports\bonita-connector-development-toolkit-1.0>install
C:\atelier supports\bonita-connector-development-toolkit-1.0>call mvn install:install-file -Dfile=bonita-connector-definition-maven-plugin-1.0-jar-with-dependencies.jar -DgroupId=com.bonitasoft.connector.definition -DartifactId=bonita-connector-definition-maven-plugin -Dversion=1.0 -DgeneratePom=true -Dpackaging=jar
[INFO] Scanning for projects...
[INFO]
[INFO] Building Maven Stub Project <No POM> 1
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
[INFO] Installing C:\atelier supports\bonita-connector-development-toolkit-1.0\bonita-connector-definition-maven-plugin-1.0-jar-with-dependencies.jar to C:\Users\pierre-yves\.m2\repository\com\bonitasoft\connector\definition\bonita-connector-definition-maven-plugin\1.0\bonita-connector-definition-maven-plugin-1.0.jar
[INFO] Installing C:\Users\PIERRE\Temp\mninstall13225927671942964541.pom to C:\Users\pierre-yves\.m2\repository\com\bonitasoft\connector\definition\bonita-connector-definition-maven-plugin\1.0\bonita-connector-definition-maven-plugin-1.0.pom
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 5.964s
[INFO] Finished at: Fri Apr 15 13:08:42 PDT 2016
[INFO] Final Memory: 6M/122M
[INFO]
[INFO] Scanning for projects...
[INFO]
[INFO] Building Maven Stub Project <No POM> 1
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
[INFO] Installing C:\atelier supports\bonita-connector-development-toolkit-1.0\bonita-connector-implementation-maven-plugin-1.0-jar-with-dependencies.jar to C:\Users\pierre-yves\.m2\repository\com\bonitasoft\connector\implementation\bonita-connector-implementation-maven-plugin\1.0\bonita-connector-implementation-maven-plugin-1.0.jar
[INFO] Installing C:\Users\PIERRE\Temp\mninstall17680141395119441446.pom to C:\Users\pierre-yves\.m2\repository\com\bonitasoft\connector\implementation\bonita-connector-implementation-maven-plugin\1.0\bonita-connector-implementation-maven-plugin-1.0.pom
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 2.728s
[INFO] Finished at: Fri Apr 15 13:08:52 PDT 2016
[INFO] Final Memory: 6M/122M
[INFO]
[INFO] Scanning for projects...
[INFO]
[INFO] Building Maven Stub Project <No POM> 1
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
[INFO] Installing C:\atelier supports\bonita-connector-development-toolkit-1.0\bonita-connector-definition-archetype-1.0.jar to C:\Users\pierre-yves\.m2\repository\com\bonitasoft\connector\definition\bonita-connector-definition-archetype\1.0\bonita-connector-definition-archetype-1.0.jar
[INFO] Installing C:\Users\PIERRE\Temp\mninstall16401969728710525662.pom to C:\Users\pierre-yves\.m2\repository\com\bonitasoft\connector\definition\bonita-connector-definition-archetype\1.0\bonita-connector-definition-archetype-1.0.pom
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 2.076s
[INFO] Finished at: Fri Apr 15 13:08:59 PDT 2016
[INFO] Final Memory: 6M/122M
[INFO]
[INFO] Scanning for projects...
[INFO]
[INFO] Building Maven Stub Project <No POM> 1
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
[INFO] Installing C:\atelier supports\bonita-connector-development-toolkit-1.0\bonita-connector-implementation-archetype-1.0.jar to C:\Users\pierre-yves\.m2\repository\com\bonitasoft\connector\implementation\bonita-connector-implementation-archetype\1.0\bonita-connector-implementation-archetype-1.0.jar
[INFO] Installing C:\Users\PIERRE\Temp\mninstall1864969888309888285.pom to C:\Users\pierre-yves\.m2\repository\com\bonitasoft\connector\implementation\bonita-connector-implementation-archetype\1.0\bonita-connector-implementation-archetype-1.0.pom
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 2.619s
[INFO] Finished at: Fri Apr 15 13:09:07 PDT 2016
[INFO] Final Memory: 6M/122M

```

Figure 93. Download the toolkit

8.4.2. Create the definition

Open a command window and navigate to the folder that will hold your connector definition. For exemple, we create a folder UserCreation under c:\atelier supports

```
cd "C:\atelier supports"
```

and then Run the following command:

```
mvn archetype:generate -DinteractiveMode=false
-DarchetypeGroupId=com.bonitasoft.connector.definition -DarchetypeArtifactId=bonita
-connector-definition-archetype -DarchetypeVersion=1.0
-DgroupId=com.bonitasoft.connectors -DartifactId=userCreation -Dversion=1.0.0
```

If the result is success, a new directory named UserCreation should be created.

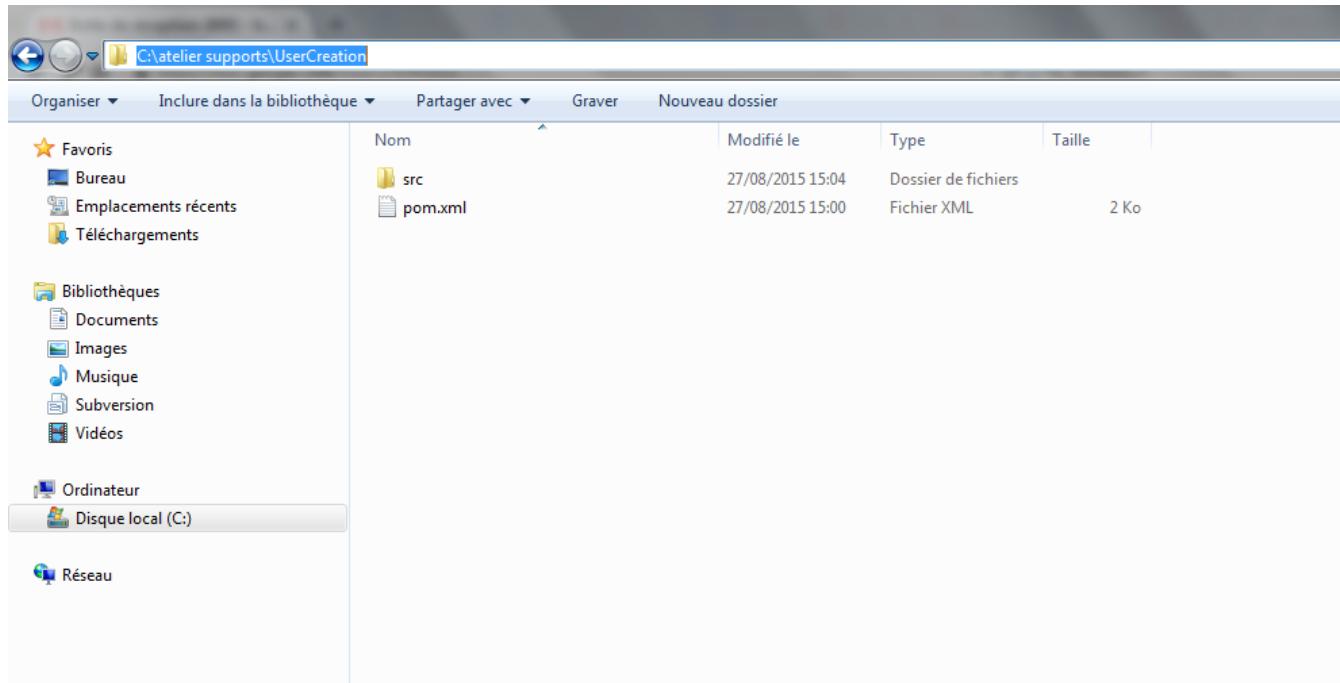


Figure 94. Create the definition

Edit the file pom.xml under **c:\atelier support\userCreation** and change the Bonita.version

```
<properties>
    <bonita.version>7.2.0</bonita.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

Edit the file userCreation/src/main/resources/connector_definition.xml. Change the content of this file as the following:

WARNING

change the BonitaVersion to the version you use at this moment (7.0.2 in the example)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ConnectorDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <_package>com.bonitasoft.connectors</_package>
    <id>UserCreation</id>
    <version>1.0.0</version>
    <label>user creation connector</label>
    <description>UserCreation 1.0.0</description>
    <icon>connector-icon.png</icon>
```

```

<category>
    <id>MyCategory</id>
    <icon>category-icon.png</icon>
</category>

<!-- Configure the version of your Bonita Engine -->
<bonitaVersion>7.0.2</bonitaVersion>

<!-- connector Wizard pages configuration -->
<pages>
    <page>
        <id>userData</id>
        <title>User Data</title>
        <description>User Data description</description>
        <!-- an ordered set of one or more widget tags contained inside a page -->
        <widgets>
            <widget>
                <id>Username</id>
                <label>Username</label>
                <!-- widget types: [Text, Password, TextArea, Checkbox,
RadioGroup,
Select, Array, ScriptEditor, List, Group] -->
                <widgetType>Text</widgetType>
                <javaType>java.lang.String</javaType>
                <mandatory>true</mandatory>
                <defaultValue/>
                <description>The username</description>
            </widget>
            <widget>
                <id>Password</id>
                <label>Password</label>
                <!-- widget types: [Text, Password, TextArea, Checkbox,
RadioGroup,
Select, Array, ScriptEditor, List, Group] -->
                <widgetType>Password</widgetType>
                <javaType>java.lang.String</javaType>
                <mandatory>true</mandatory>
                <defaultValue/>
                <description>The password</description>
            </widget>
            <widget>
                <id>FirstName</id>
                <label>First Name</label>
                <widgetType>Text</widgetType>
                <javaType>java.lang.String</javaType>
                <mandatory>true</mandatory>
                <defaultValue/>
                <description>The FirstName</description>
            </widget>
            <widget>
                <id>LastName</id>

```

```

<label>Last Name</label>
<widgetType>Text</widgetType>
<javaType>java.lang.String</javaType>
<mandatory>true</mandatory>
<defaultValue/>
<description>The LastName</description>
</widget>
<widget>
<id>ProfileId</id>
<label>ProfileId</label>
<widgetType>Text</widgetType>
<javaType>java.lang.String</javaType>
<mandatory>true</mandatory>
<defaultValue/>
<description>The Profile ID</description>
</widget>
</widgets>
</page>
</pages>
<!-- the 'outputs' configuration tag is optional and responsible for defining --&gt;
<!-- connector outputs --&gt;
&lt;outputs&gt;
&lt;output&gt;
&lt;id&gt;result&lt;/id&gt;
&lt;javaType&gt;java.lang.String&lt;/javaType&gt;
&lt;/output&gt;
&lt;/outputs&gt;

&lt;/ConnectorDefinition&gt;
</pre>

```

8.4.3. Create the implementation

Open a command windows, navigate to UserCreation folder and run the following command:

```

cd "c:\atelier supports\userCreation"

mvn archetype:generate -DinteractiveMode=false
-DarchetypeGroupId=com.bonitasoft.connector.implementation
-DarchetypeArtifactId=bonita-connector-implementation-archetype -DarchetypeVersion=1.0
-DgroupId=com.bonitasoft.connectors -DartifactId=userCreationImpl -Dversion=1.0.0
-DdefinitionId=userCreation -DdefinitionVersion=1.0.0

```

If the result of build is success, a directory named "UserCreationImpl" containing the connector implementation project will be created.

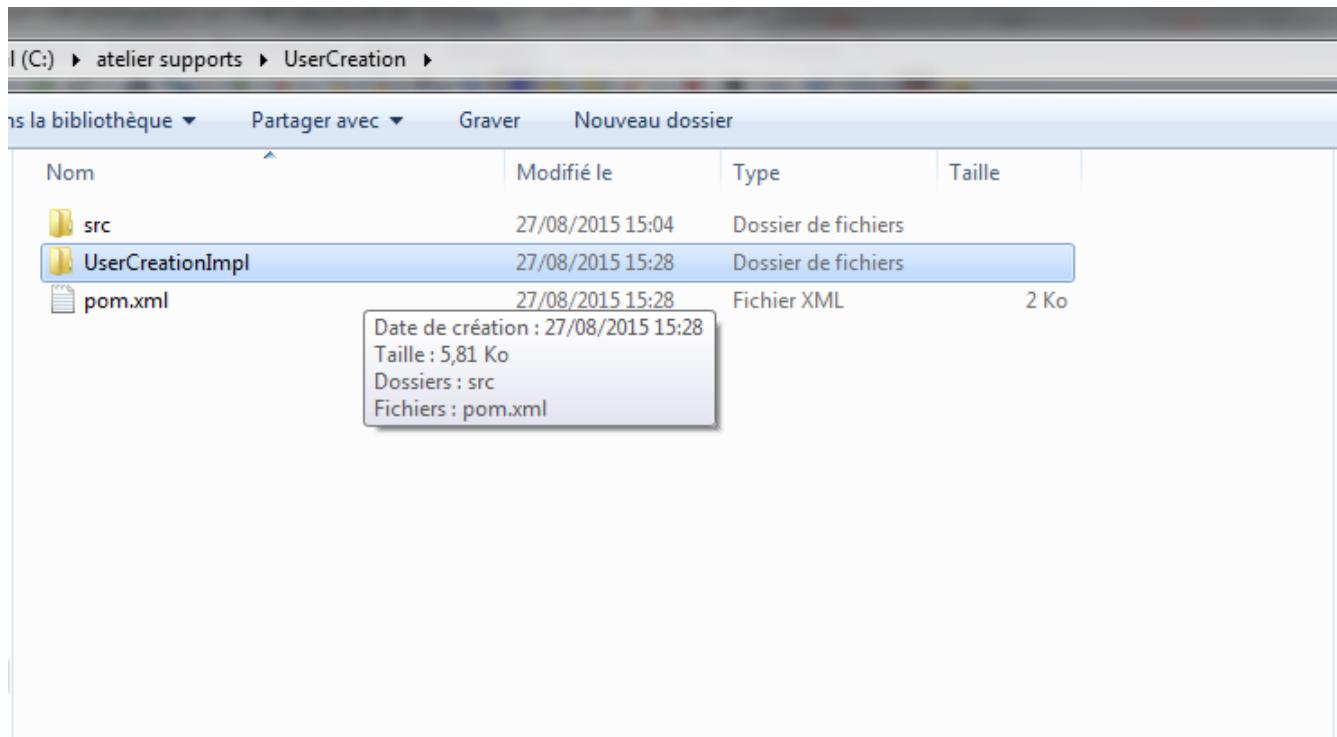


Figure 95. Check the creation

8.4.4. Generate the java project

run the following command:

```
cd "c:\atelier supports\userCreation\userCreationImpl"  
mvn bonita-connector-definition:generate  
mvn bonita-connector-implementation:generate
```

After these commands, an abstract class named "AbstractUserCreation.java" and a stub of the connector implementation named "UserCreationImpl.java" will be created.

al (C:) ▶ atelier supports ▶ UserCreation ▶ UserCreationImpl ▶ src ▶ main ▶ java ▶ com ▶ bonitasoft ▶			
Nouveau dossier			
Nom	Modifié le	Type	Taille
AbstractUserCreation.java	27/08/2015 15:35	Fichier JAVA	2 Ko
UserCreationImpl.java	27/08/2015 15:35	Fichier JAVA	2 Ko

Figure 96. Java files should be generated

8.4.5. Launch Eclipse and import the maven project

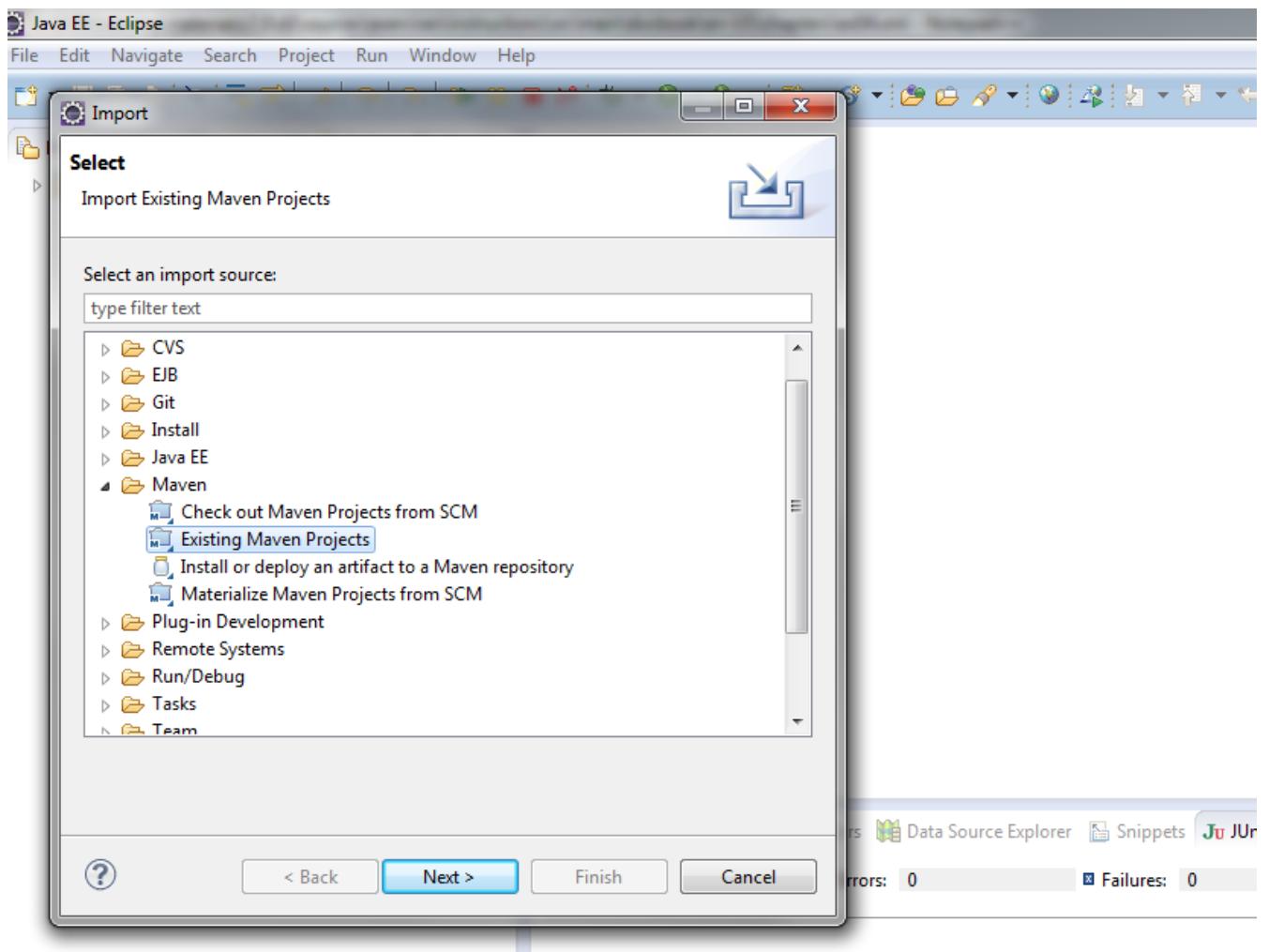


Figure 97. Import the Maven project

Select "pom.xml" file located under UserCreationImpl directory

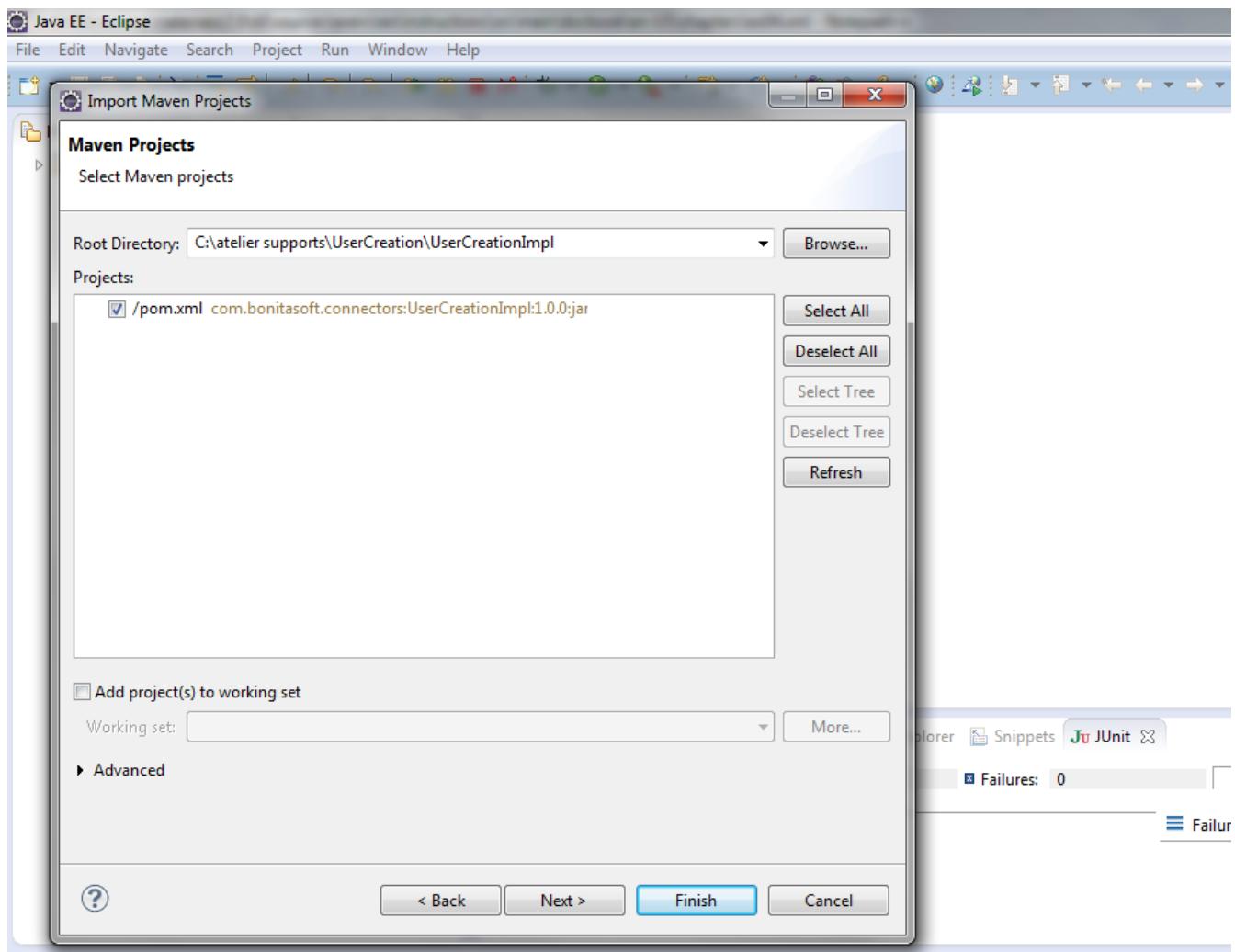


Figure 98. Access the pom.xml

Click on Finish. This operation adds one project in Eclipse.

8.4.6. Complete your connector implementation.

Open "UserCreationImpl" class and modify the method "executeBusinessLogic" with the following code:

```
IdentityAPI identityAPI = getAPIAccessor().getIdentityAPI();
String result;
try {
    // Execute the delegate object business logic. Provide identityAPI
    reference.
    User newUser = identityAPI.createUser(getUsername(), getPassword());
    result = String.valueOf(newUser.getId() );
} catch (Exception e) {
    throw new ConnectorException(e);
}
setResult(result);
```

8.4.7. Generate the connector

In the command prompt, go to "UserCreationImpl" directory and execute the following command:

```
cd "c:\atelier supports\userCreation\userCreationImpl"  
mvn package
```

The connector is now generated and packaged in zip file under "UserCreation/UserCreationImpl/target" directory

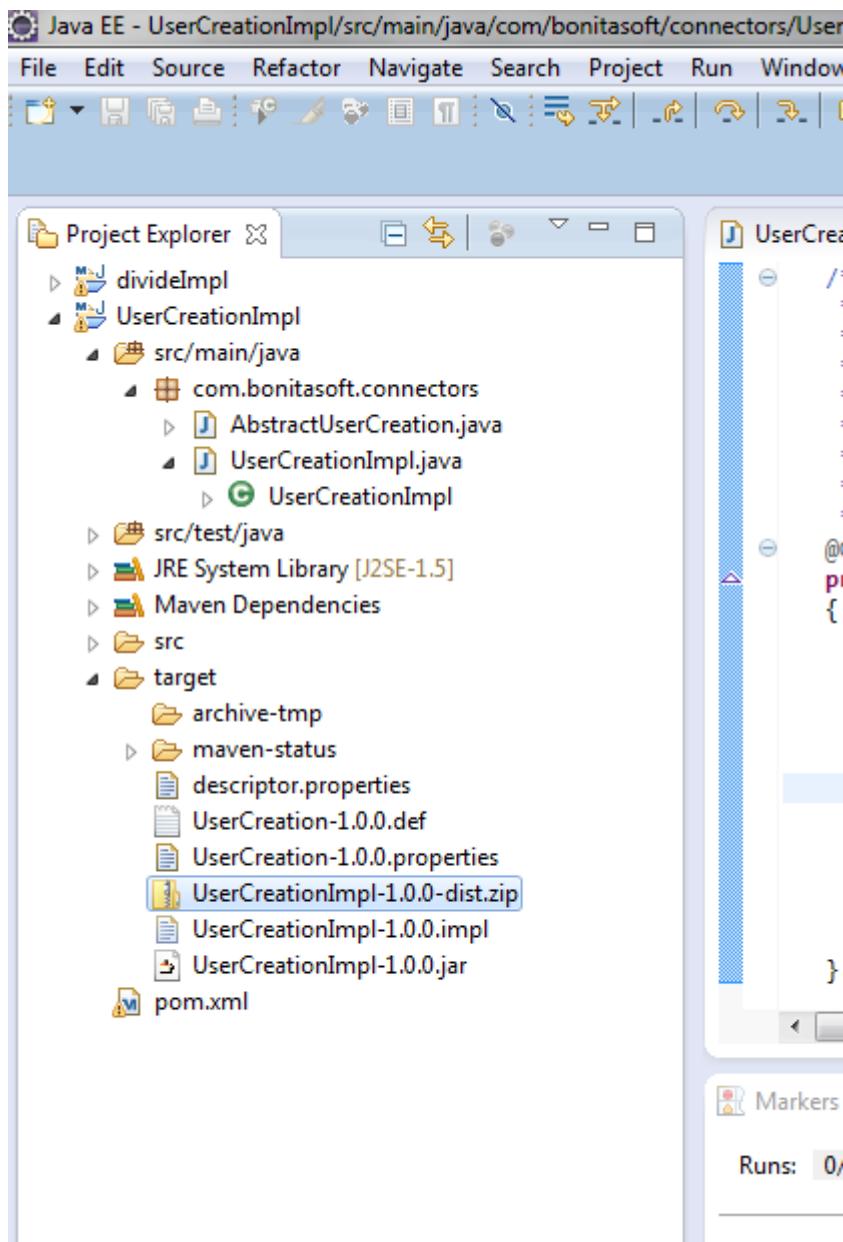


Figure 99. Check the Zip file

8.4.8. Import the zip file generated in the previous step in Bonita studio

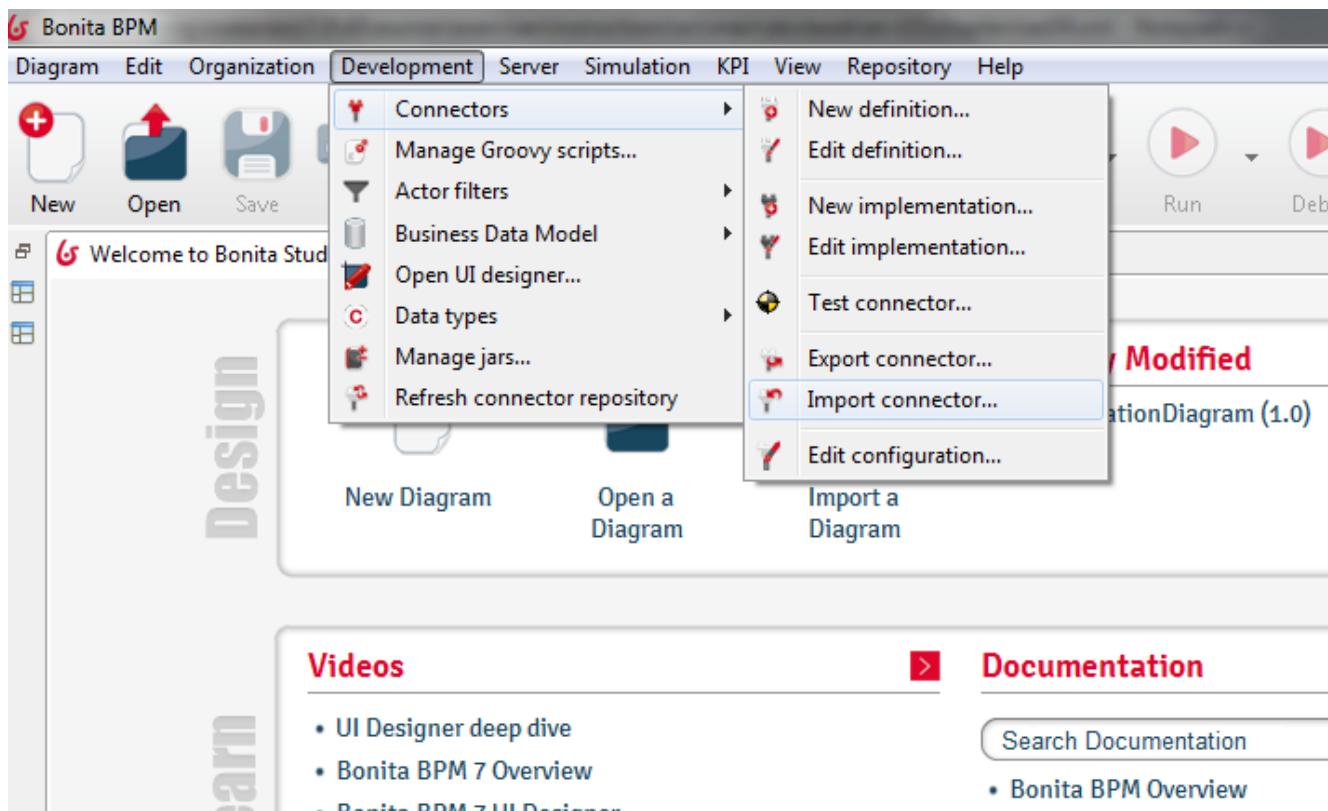


Figure 100. Import the connector

8.4.9. Import "UserCreationDiagram.bos" from provided_files folder

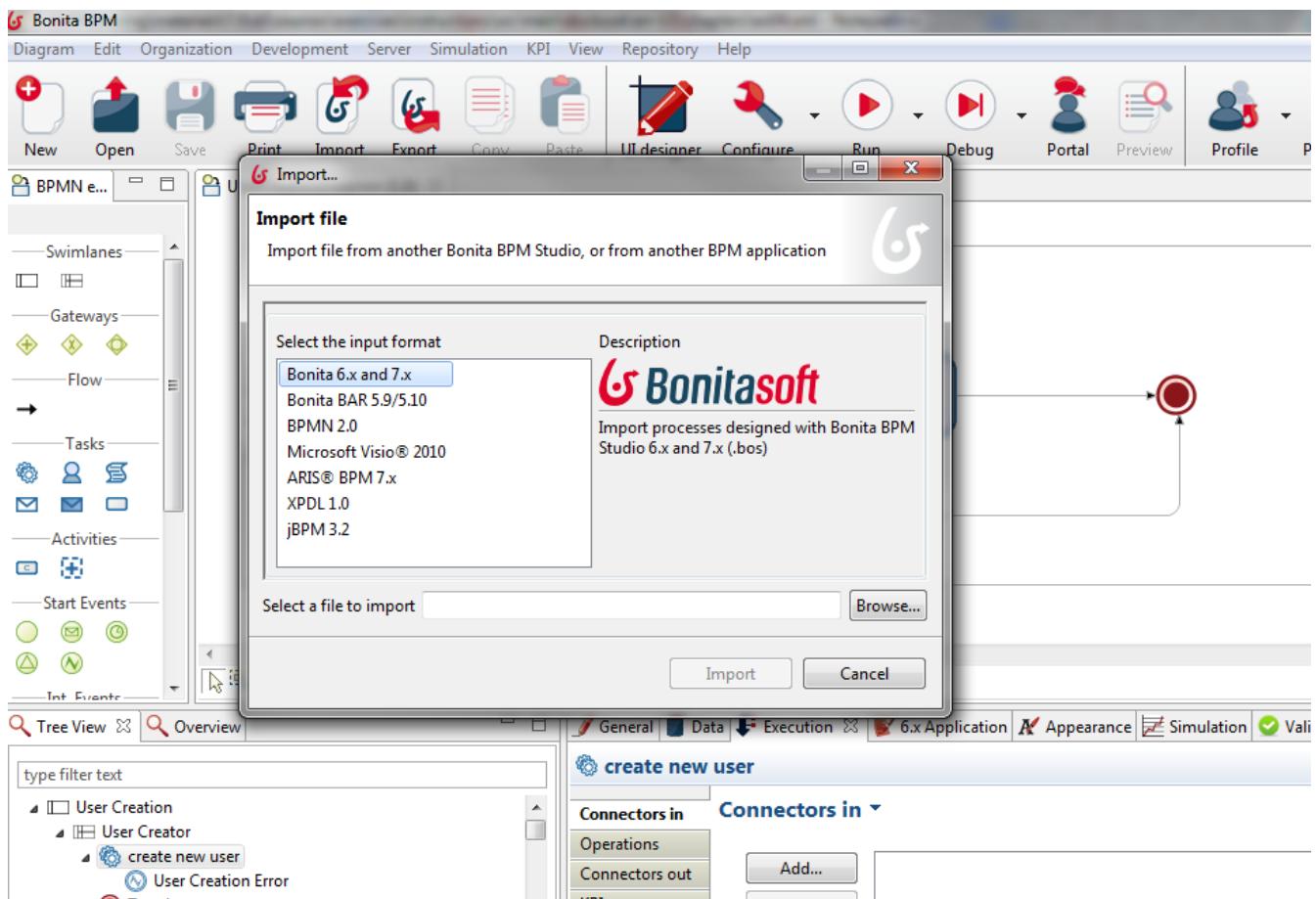


Figure 101. Import the process

- Create a connector OUT in the "Create new user" activity.

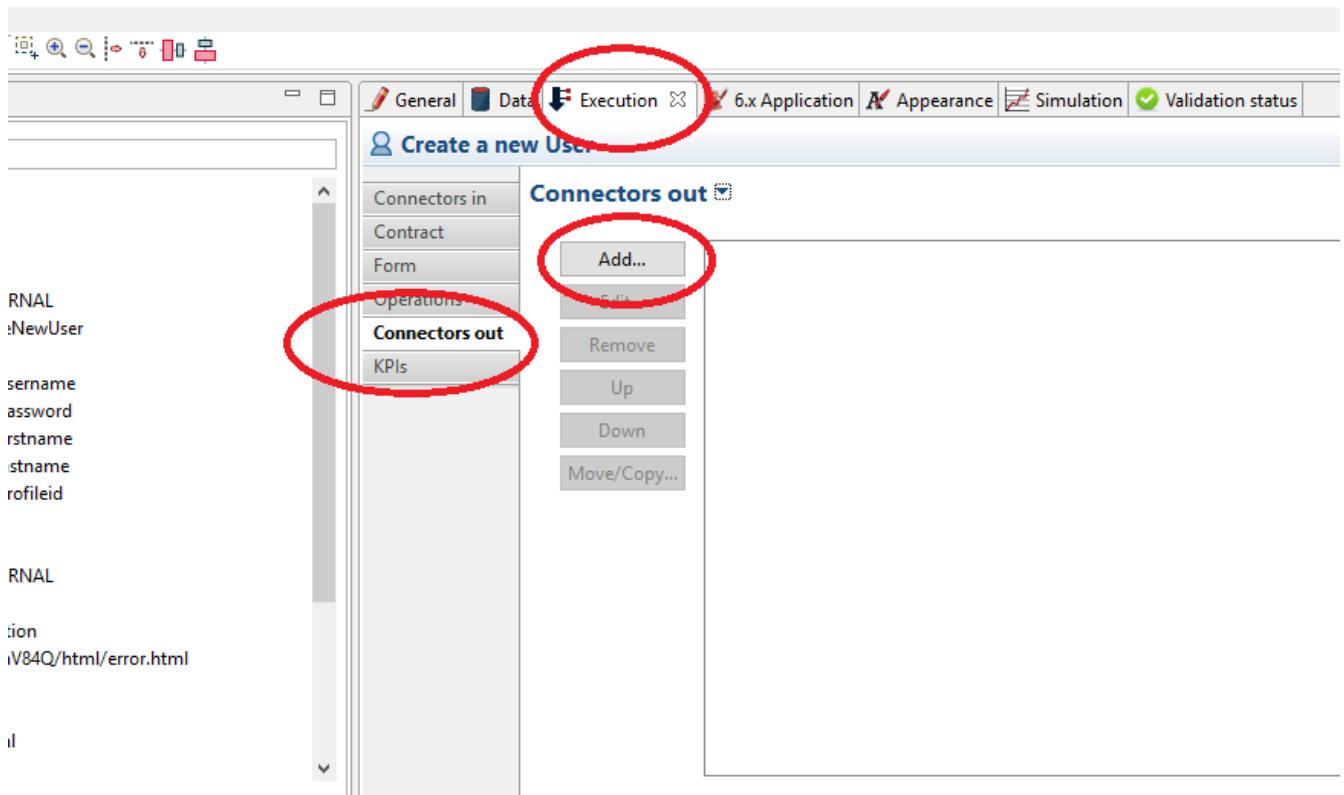
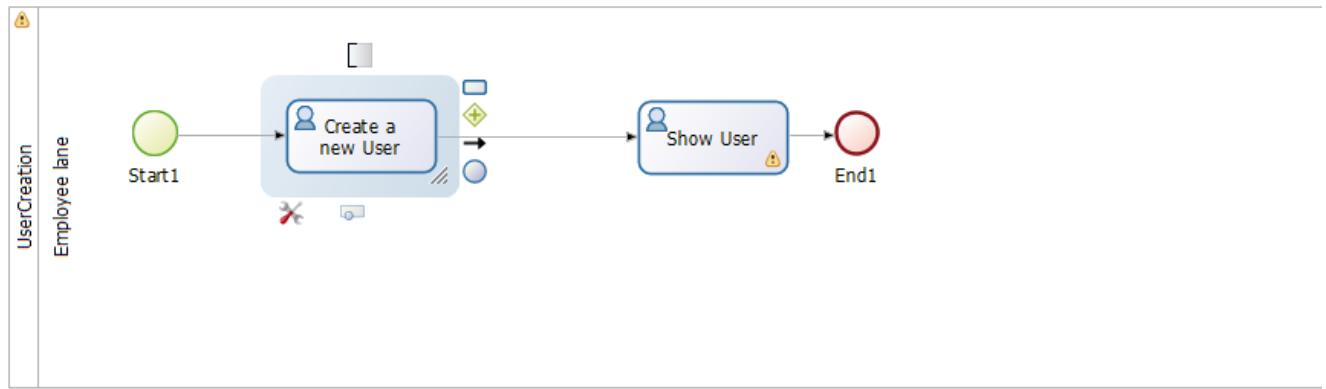


Figure 102. Create the connector Out

- The username, password, firstName, lastName, profileId connector inputs are mapped to the relative **contract** data.
- The resultMessage process data will hold a message coming from the connector showing the result.

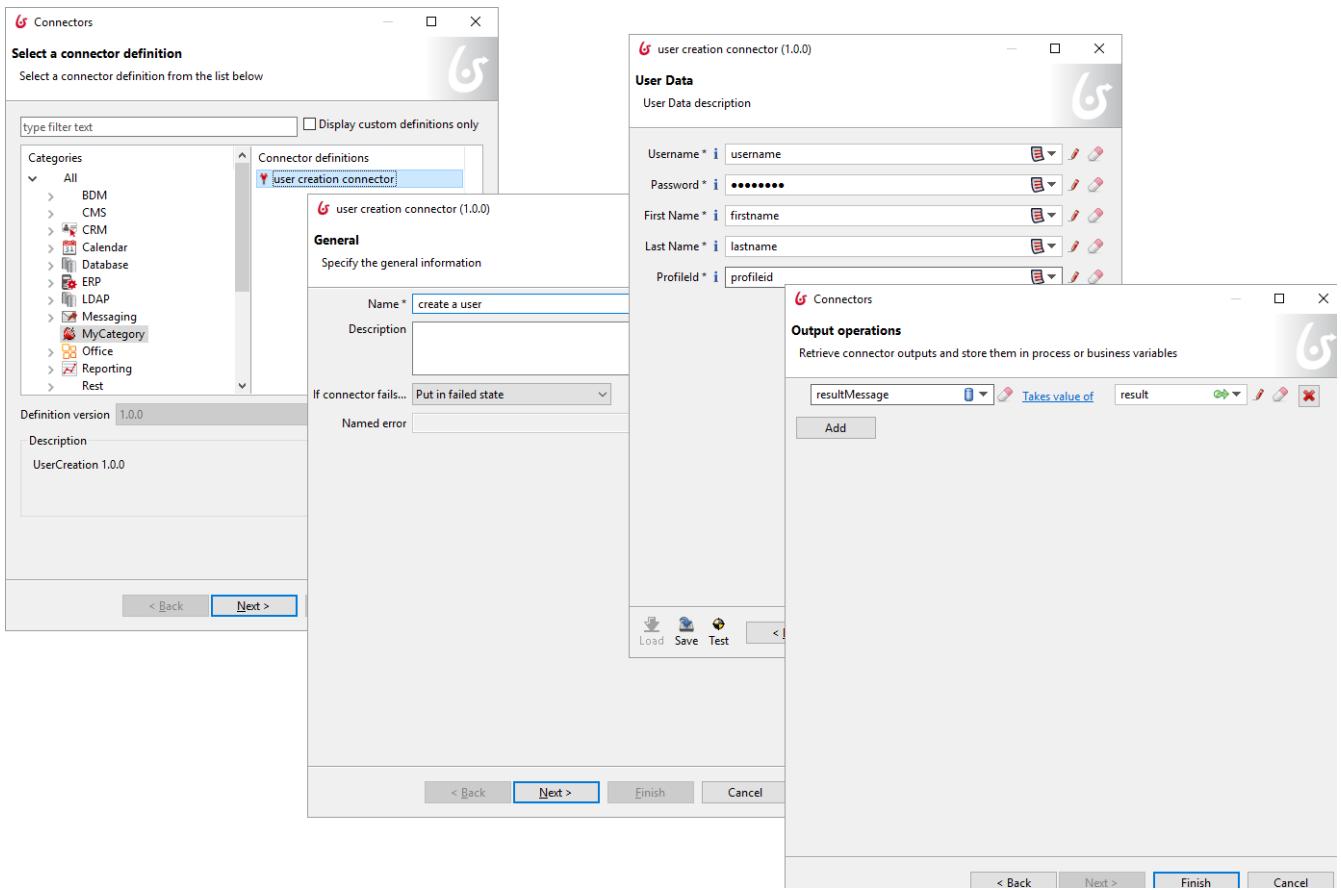


Figure 103. Reference the connector

- This resultMessage is mapped to the display name of "Show User" task

8.4.10. Test the process

- Run the process from the studio
- In the empty instantiation form click on the Start button to start the process
- Open the task "Create a new User" from the task list
- Fill the text widgets, choose the "User" profile, and click on "Submit"
- In the task list you should get a task that has for display name the connector output message
- From the portal change the profile to Administrator, switch to Organization, and double check that the user has been added in the organization with the User profile

Chapter 9. Create a new Custom Widget

9.1. Prerequisites

You must have Bonita BPM Studio SP installed and started.

9.2. Objectives

The goal of this exercise is to create a new Custom Widget. This widget is a Select list where the user can select multiple values, not only one.

9.3. Instructions

In this exercise we will create a custom widget.

- Create a new Custom widget
- Define the properties of the widget : list of available values (and key, label), list of items selected (you should call it value to be consistent with other widgets).
- Define the HTML and the Angular JS part. The HTML is the look and feel, and it will contain the reference to the AngularJS scope. Then, the HTML present a select box with values retrieved from the scope. When the user clicks on an item, the controller is called, and maintains the list of items selected.
- To test the custom widget, reference the widget in a page, and call the Preview function. When you do a change, just save again the page, and in the UI Designer refresh the Preview windows.

9.3.1. Correction

9.3.2. Create the custom widget

Launch the UI Designer from the Studio, and then go to Create. Select Custom Widget and give MultipleDropDown as name

[exCustomWidget 01] | *exCustomWidget_01.png*

Figure 104. Create a new custom widget

9.3.3. Access the page

You have different area :

[exCustomWidget 02] | *exCustomWidget_02.png*

Figure 105. Custom Widget

- Template is the HTML Part
- Controller is the AngularJS part
- Asset is used if you want to add some JAVASCRIPT library or any resource file

- Angular Module is used to reference some external angular module
- Properties represents the properties of your widget

9.3.4. Properties

Please indicate the following properties:

Properties name	Label	Treat value as	Type	Default
label	Label	Interpolation		Select
availableValues	Available Values	Dynamic value	Collection	
displayedKey	Displayed Key	Dynamic value	Text	
returnedKey	Returned Key	Dynamic value	Text	
value	Value	Dynamic value	Collection	

Properties are:

[exCustomWidget 03] | *exCustomWidget_03.png*

Figure 106. Properties

9.3.5. HTML

The HTML part will display the result of the select box. It will use the AngularJS syntax to present the list of values.

If we display the list, we loop on it and display the glyphicon image if the element is selected.

On each element we can click to update the list.

```
<label class="control-label col-xs-12">{{ properties.label | uiTranslate }}</label>
<button type="button" ng-
click="ctrl.toggleDropdown()">{{ctrl.getButtonLabel()}}</button>
<div class="panel panel-warning" ng-show="ctrl.isOpen">
  <div class="panel-heading">
    <table>
      <tr ng-repeat="oneline in ctrl.getValues() track by $index">
        <td>
          <span class="glyphicon glyphicon-ok" ng-
show="ctrl.isAlreadySelected(oneline)"></span>
        </td>
        <td>
          <a ng-click="ctrl.clickOnLine(oneline)">{{ctrl.getLabel(oneline)}}</a>
        </td>
      </tr>
    </table>
  </div>
</div>
```

9.3.6. Controller

The control is the following code.

The controller put in the scope 2 properties: open, list of value.

The next methods are for the behavior.

```
function ($scope) {
    var ctrl = this;

    var isOpen = true;

    var listValues = $scope.properties.value;

    this.toggleDropdown = function(){
        this.isOpen = !this.isOpen;
    }

    this.getValues = function () {
        return $scope.properties.availableValues;
    }

    this.getButtonLabel = function(){
        if (listValues.length === 0){
            return "No selection";
        }else if (listValues.length === 1){
            return "1 item selected";
        }else{
            return listValues.length+ " items selected";
        }
    }

    this.getLabel = function (oneLine){
        return oneLine[$scope.properties.displayedKey];
    }

    this.clickOnLine = function (oneLine){
        var id = oneLine[ $scope.properties.returnedKey ];
        // is this id is in the list of value ? If yes, remove it, else add id
        listValues = $scope.properties.value;
        iId = listValues.indexOf( id );
        if (iId !== -1 ) {
            listValues.splice( iId, 1 );
        }else {
            listValues.push( id );
        }
    }

    this.isAlreadySelected = function( oneLine ){
        id = oneLine[$scope.properties.returnedKey];
```

```

listValues = $scope.properties.value;
if (!angular.isDefined( listValues)){
    alert("Undefined");
    return false;
}
if (listValues.indexOf( id ) !== -1) {
    return true;
}
return false;
}
}

```

9.3.7. How to test

Use two different tabs in your browser. in one tab, display the custom widget.

In a second tab, access the UI Designer (click again on the UI Designer in the studio) and create a page (TestMultiDropDown).

[exCustomWidgetCreatePage 04] | *exCustomWidgetCreatePage_04.png*

Figure 107. Create a page to test the widget

In this page, access the custom widget panel, and select the new custom widget

[exCustomWidgetAddWidget 05] | *exCustomWidgetAddWidget_05.png*

Figure 108. Select the custom widget in the panel

Drag and drop the widget in the page. Create two JSON variables :

Variable	Value	Type
availableValue	[{"car":"Dodge","carid":"DO"}, {"car":"Nissan Guest","carid":"NQU"}, {"car":"Renault Espace","carid":"RESP"}, {"car":"Suburban","carid":"SUB"}]	JSON
selectedCars	["NQU"]	JSON

And then match the properties in the widget:

property	Value
Label	Choose a car
Available value	availableValue (change the selector to a variable)
Displayed key	car
Returned key	carid

Value	selectedCars
-------	--------------

Add a text widget, and set as the text

```
<span class="label label-success">Value:{{selectedCars}}</span>
```

The result should be

[exCustomWidgetResultpage 05] | *exCustomWidgetResultpage_05.png*

Figure 109. Definition of the test page

So, now, it's time to test it : click on Preview. The result is

[exCustomWidgetPreview 06] | *exCustomWidgetPreview_06.png*

Figure 110. Preview page

Tips: To change and to test the new custom Widget, in the tab related to the custom widget, save the modification. Then, in the application page, just click on save (even if you don't change anything) : then preview then reload the page, and take into account change in the custom widget.