eXo Platform 3.0 - Social Functions

Chapter 1. Applications

1.1. List of Portlets in Social

All the portlets are in *social-portlet.war*

Portlets name	Description
SpaceMenuPortlet	Display the menu on the right of a Space: Home, Dashboard, Members, SpaceSettings
MembersPortlet	Display MemberSpace where you can search for space members or list space members in alphabet
SpaceSettingPortlet	Shows the settings of Space: Settings, Visibility, Members, Application. You can change the setting information of a space if you are the creator of that space or have the manager right on it.
MySpacesPortlet	Displays your Space page where you can add new spaces, search for spaces or list spaces in alphabet
InvitationSpacesPortlet	Displays the **Space invitation** page where you can search for spaces, list spaces in alphabet and view all the Space invitations
PendingSpacesPortlet	Displays the **Space requests to join** page where you can search for spaces, list spaces in alphabet and view all the pending request to join Spaces
PublicSpaces Portlet	Displays the **Public Spaces** page where you can search for spaces, list spaces by alphabet and view all available public spaces to join
SpaceActivityStreamPortlet	Display spaces activities
UserActivityStreamPortlet	Update user's activities/status
PeoplePortlet	The People page where you can find people, display people name by alphabet
ProfileNavigationPortlet	
ConnectionsPortlet	The Conections page where you can access to your network, invitations to connect or connection requests
InvitationsPortlet	
RequestsPortlet	
UserProfilePortlet	Display user profile page where you can view/ edit user's basic information, contacts, experience or change avatar
MyConnectionsNavigationPortlet	

All the portlets are in the file *social-portlet.war*

Portlet name	Description
MembersPortlet	To enable users to search for Space members or list Space members in the alphabet order.
MySpacesPortlet	To display the Space page where you can add new Spaces, search for Spaces or list Spaces in the alphabet order.
SpaceActivityStreamPortlet	To share Spaces activity information.
InvitationsPortlet	To list all people that invite users.
RequestsPortlet	To list all invitations that the user requests.
InvitationSpaces Portlet	To enable users to search for Spaces, and list spaces in the alphabet order and view all the Space invitations.
PendingSpacePortlet	To display the **Space requests to join** page where you can search for Spaces, list spaces in the alphabet order and view all the pending requests to join Spaces.
PublicSpacesPortlet	To display the **Public Spaces** page where you can search for Spaces, list Spaces in the alphabet order and view all available public Spaces to join.
UserActivityStreamPortlet	To update and share user's activities and/or status.
People Portlet	To display the People page where you can find people, display people names in the alphabet order.
ConnectionsPortlet	To display the Connections page where you can access your network, invitations to connect or connection requests.
UserProfilePortlet	To display the user profile page where users can view and/or edit basic information, contacts, experience or change their avatar.

1.2. List of Gadgets in Social

All Social Gadgets are in opensocial.war

1.2.1. Activities

- **Description:** Manages activities of users. Some kinds of activities such as: update status, like/unlike activities, comment activities, delete activities and delete comments.
- **Used REST services:** ActivitiesRestServices

1.2.2. ApplicationList

- **Description:** Gets information of all applications (portlet type) that is existing in Portal. This is gadget is used for demo rest service get application from portal container.
- **Used REST services:** AppsRestService

1.2.3. RSSFetch

- **Description:** Fetches and parses Rss information from specific url.
- **Description of user preferences:** Number of rss per page is 10 by default; Get rss from input url (Default url is http://blog.exoplatform.org/feed/).
- **Used REST services:** N/A

1.2.4. UserSpaces

- **Description:**
- **Used REST services:** SpacesRestService

1.2.5. MySpaces

- **Description:** Gets information of space that is my space and pending space of current login user. This gadget is used for demo using tab in writing gadget.
- **Used REST services:** SpacesRestService

All Social Gadgets are in the *opensocial.war* file

Gadget name	Used REST service	Description
Activities	ActivitiesRestServices	To manage activities of users. For example, update status, like/unlike activities, comment activities, delete activities and delete comments.
ApplicationList	AppsRestService	To get information of all existing applications (portlet type) in the Portal. The gadget is used for the demo rest service to get application from the portal container.
RSSFetch	N/A	The number of rss per page is 10 by default; Get the rss from input url (Default url is http://blog.exoplatform.org/feed/)
UserSpaces	SpacesRestService	

Gadget name	Used REST service	Description
MySpaces	SpacesRestService	To get information of spaces, including my space and pending spaces of current login users. This gadget is used for the demo using tab in writing gadgets.

Chapter 2. Configuration

2.1. Components

2.1.1. ActivityManager

Configuration name	Data type	Default Value	Description
allowedTags	`String list`	b, i, a, span, em, strong, p, ol, ul,li, br, img	html tags

2.1.2. SpaceService

Configuration name	Data type	Default Value	Description
SpaceActivityStreamPortlet	`String`	SpaceActivityStreamPortlet	
space.apps	`String list`	`DashboardPortlet:true`, `SpaceSettingPortlet:false`, `MembersPortlet:true`	

2.1.3. IdentityManager

Configuration name	Data type	Default Value	Description
providers	`String`	`org.exoplatform.social.core.identity.provider.SpaceIdent`	

2.1.4. ProfileConfig

Configuration name	Data type	Default Value	Description
nodetype.emails	`String`	`exo:profileKeyValue`	
nodetype.phones	`String`	`exo:profileKeyValue`	
nodetype.ims	`String`	`exo:profileKeyValue`	
nodetype.urls	`String`	`exo:profileKeyValue`	
nodetype.address	`String`	`exo:profileAddress`	
nodetype.experiences	`String`	`exo:profileExperience`	
nodetype.education	`String`	`exo:profileEducation`	
forceMultiValue	`String`	xxxxxxxxxxxx	

2.1.5. ServiceProviderStore

Configuration name	Data type	Default Value	Description
sample-provider	properties-params		sample service provider

2.2. External component plugins

2.2.1. MentionsProcessor

Configuration name	Data type	Default Value	Description
priority	`String`	2	priority of this processor (lower are executed first)

2.2.2. PortletPreferenceRequiredPlugin

Configuration name	Data type	Default Value	Description
portletsPrefsRequired	`String list`	`SpaceActivityStreamPortlet, SpaceSettingPortlet, MembersPortlet`	

2.2.3. AddNodeTypePlugin

Configuration name	Data type	Default Value	Description
autoCreatedInNewRepository	`String`	`jar:/conf/portal/core-nodetypes.xml`	Node types configuration file

2.3. RelationshipManager

The Service is used to manipulate user relationships.

2.4. SpaceIdentityProvider

The provider is to give identity space instances.

2.5. SpaceApplicationHandler

The service is to handle all events related to creating and managing all the application spaces.

2.6. ExoPeopleService

The service is to manipulate all data related to people in the Portal.

2.7. Space Service

2.7.1. Description

The service for spaces management includes creating spaces, and installing applications.

2.7.2. Components configuration

Component name	Description
SpaceServiceImpl	Implementation class of Space Service.

```
<component>
    <key>org.exoplatform.social.core.space.spi.SpaceService</key>
    <type>org.exoplatform.social.core.space.impl.SpaceServiceImpl</type>
    <init-params>
      <!-- Configure the applications to install in a space -->
      <values-param>
        <name>space.homeNodeApp</name>
        <value>SpaceActivityStreamPortlet</value>
      </values-param>
      <!-- Configure removable application or not <value>Application:removable</value> -->
      <values-param>
        <name>space.apps</name>
        <value>DashboardPortlet:true</value>
        <value>SpaceSettingPortlet:false</value>
        <value>MembersPortlet:true</value>
      </values-param>
    </init-params>
  </component>
```

Configuration name	Data Type	Possible value	Default Value	Description
SpaceActivityStreamPortlet	String	N/A	`SpaceActivityStreamPortlet`	The name of portlet displaying activities of spaces
space.apps	`String list`	`Porlets' name: true/false`	`DashboardPortlet:true, SpaceSettingPortlet:false, MembersPortlet:true`	The list of configurations for portlets used as portlet applications.

2.7.3. External plug-in configuration

```
 <external-component-plugins>
    <target-component>org.exoplatform.social.core.space.spi.SpaceService</target-component>
    <component-plugin>
      <name>portlets.prefs.required</name>
```

```
    <set-method>setPortletsPrefsRequired</set-method>
    <type>org.exoplatform.social.core.application.PortletPreferenceRequiredPlugin</type>
          <init-params>
            <values-param>
              <name>portletsPrefsRequired</name>
              <value>SpaceActivityStreamPortlet</value>
              <value>SpaceSettingPortlet</value>
              <value>MembersPortlet</value>
            </values-param>
          </init-params>
  </component-plugin>
</external-component-plugins>
```

In which:

Name	Set-method	Type	Description
PortletPreferenceRequiredPlugin	setPortletsPrefsRequired	org.exoplatform.social.core.application.PortletPreferenceRequiredPlugin	Configuration for portlets which portlet names which will have portlet preference of space context.

- Init-params:

Name	Possible value	Default Value	Description
portletsPrefsRequired	`Portlet names`	`SpaceActivityStreamPortlet; SpaceSettingPortlet; MembersPortlet`	List of portlets which need to be saved and get the space context name.

2.8. Activity Manager

2.8.1. Description

The service is to manipulate space and user activities.

2.8.2. Component plug-in configuration

```
<component-plugin>
      <name>OSHtmlSanitizer</name>
      <set-method>addProcessorPlugin</set-method>
      <type>org.exoplatform.social.core.processor.OSHtmlSanitizerProcessor</type>
      <init-params>
        <values-param>
          <name>allowedTags</name>
          <value>b</value>
          <value>i</value>
          <value>a</value>
          <value>span</value>
          <value>em</value>
          <value>strong</value>
          <value>p</value>
          <value>ol</value>
          <value>ul</value>
          <value>li</value>
          <value>br</value>
          <value>img</value>
        </values-param>
```

```
        </init-params>
  </component-plugin>
```

In which,

Name	Set-method	Type	Description
OSHtmlSanitizerProcessor	addProcessorPlugin	org.exoplatform.social.core.processor.OSHtmlSanitizerProcessor	The plugin filters invalid html tags appearing in the Activity body (content).

- Init-params:

Name	Possible value	Default Value	Description
allowedTags	`html tags`	b, i, a, span, em, strong, p, ol, ul,li, br, img	To process and render html tags in the activity content (body).

2.8.3. External plug-in configuration

```
<external-component-plugins>
    <target-component>org.exoplatform.social.core.manager.ActivityManager</target-component>
    <component-plugin>
      <name>MentionsProcessor</name>
      <set-method>addProcessorPlugin</set-method>
      <type>org.exoplatform.social.core.processor.MentionsProcessor</type>
      <init-params>
        <value-param>
          <name>priority</name>
          <description>priority of this processor (lower number will be executed first)</description>
          <value>2</value>
        </value-param>
      </init-params>
    </component-plugin>
</external-component-plugins>
```

In which:

Name	Set-method	Type	Description
MentionsProcessor	addProcessorPlugin	org.exoplatform.social.core.processor.MentionsProcessor	A processor that substitutes @username expressions by a link on the user profile.

- Init-params:

Name	Possible value	Default Value	Description
priority	`priority number`	2	Priority of this processor (The lower level will be

Name	Possible value	Default Value	Description
			executed first).

2.9. Identity Manager

2.9.1. Description

The service is to manipulate the identity operations like creating, getting, deleting or finding a profile.

2.9.2. Component plug-in configuration

```
<component>
    <key>org.exoplatform.social.core.manager.IdentityManager</key>
    <type>org.exoplatform.social.core.manager.IdentityManager</type>
    <component-plugins>
      <component-plugin>
        <name>SpaceIdentityProvider plugin</name>
        <set-method>registerIdentityProviders</set-method>
        <type>org.exoplatform.social.core.identity.IdentityProviderPlugin</type>
        <init-params>
          <values-param>
          <name>providers</name>
            <description>Identity Providers</description>
            <value>org.exoplatform.social.core.identity.provider.SpaceIdentityProvider</value>
          </values-param>
        </init-params>
      </component-plugin>
    </component-plugins>
  </component>
```

In which:

Name	Set-method	Type	Description
SpaceIdentityProvider plugin	registerIdentityProviders	org.exoplatform.social.core.identity.IdentityProviderPlugin	The plugin that provides identity for a space

- Init-params:

Name	Possible value	Default Value	Description
providers	`Every other identity providers`	`org.exoplatform.social.identity.provider.SpaceIdent`	Identity Provider instances for managing identities.

2.10. OpenSocial Rest Context Configuration

2.10.1. Description

The service is used to configure the portal container name when there is a OpenSocial REST API request. By configuring this service, we can make sure to to reach the right portal container. The default portal container is `portal`.

2.10.2. Component plug-in configuration

This should be used when there is a portal container different than the default `portal` one.

```
<external-component-plugins>
  <target-component>org.exoplatform.social.opensocial.auth.RestPortalContainerNameConfig</target-component>
  <component-plugin>
    <name>set portal container name used for REST service</name>
    <set-method>setRestContainerName</set-method>
    <type>org.exoplatform.social.opensocial.auth.RestPortalContainerNameConfig</type>
      <init-params>
      <value-param>
        <name>rest-container-name</name>
        <value>socialdemo</value>
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

In which:

Name	Set-method	Type	Description
set portal container name used for REST service	setRestContainerName	org.exoplatform.social.opensocial.auth.RestPortalContainerNameC	set rest portal container name for OpenSocial REST API request

- Init-params:

Name	Possible value	Default Value	Description
rest-container-name	`any valid portal container name`	`N/A`	Portal container name

2.11. Spaces Template configuration

In the eXo Spaces, we may have two space types (classic and webos spaces). This is for the classic mode (it's the only one implemented now).

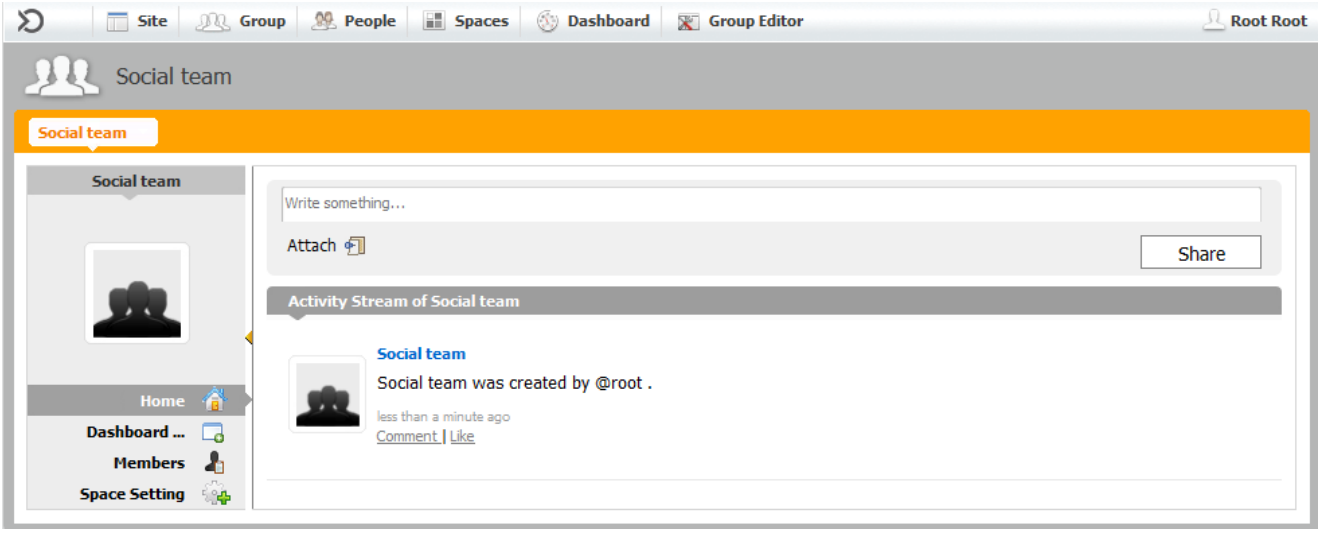
For the classic space, we can pre-configure the template, meaning that you can set up where your **menu** will be displayed or where your **application** will be displayed.

Here is an example of configuration file that displays the menu on the left. The Application will be inserted in the container with the id **Application**:

```
<page>
  <owner-type/>
  <owner-id/>
```

```
  <name/>
  <container id="SpacePage" template="system:/groovy/portal/webui/container/UITableColumnContainer.gtmpl">
    <container id="Menu" template="system:/groovy/portal/webui/container/UIContainer.gtmpl">
      <portlet-application>
        <portlet>
          <application-ref>space</application-ref>
          <portlet-ref>SpaceMenuPortlet</portlet-ref>
        </portlet>
        <access-permissions>*:/platform/users</access-permissions>
        <show-info-bar>false</show-info-bar>
      </portlet-application>
    </container>
    <container id="Application" template="system:/groovy/portal/webui/container/UIContainer.gtmpl">
    </container>
  </container>
</page>
```

In this example, the outer container contains two inner containers: one container has id as **Menu** for your Menu and another has id as **Application** containing your applications.



If you want to put your menu in right and your application in left, you can swap the declared position of two containers:

```
<page>
  <owner-type/>
  <owner-id/>
  <name/>
  <container id="SpacePage" template="system:/groovy/portal/webui/container/UITableColumnContainer.gtmpl">
    <container id="Application" template="system:/groovy/portal/webui/container/UIContainer.gtmpl">
    <container id="Menu" template="system:/groovy/portal/webui/container/UIContainer.gtmpl">
      <portlet-application>
        <portlet>
          <application-ref>space</application-ref>
          <portlet-ref>SpaceMenuPortlet</portlet-ref>
        </portlet>
        <access-permissions>*:/platform/users</access-permissions>
        <show-info-bar>false</show-info-bar>
      </portlet-application>
    </container>
    </container>
  </container>
</page>
```

Here is the configure file in FishEye: http://fisheye.exoplatform.org/browse/social/trunk/extension/war/src/main/webapp/WEB-INF/conf/portal/template/pages/sp

In your tomcat, this configuration file is at *$EXO*TOMCAT/webapps/social-ext/WEB-INF/conf/portal/template/pages/space/page.xml.

2.12. Configure the oauth 2 legged scenario

This is to explain how to configure the oAuth 2 legs scenario in openSocial.

For more information about this, visit the website: great article

2.12.1. Generate the certificates

To generate the key:

```
$ openssl req -newkey rsa:1024 -days 365 -nodes -x509 -keyout testkey.pem
     -out testkey.pem -subj '/CN=mytestkey'
$ openssl pkcs8 -in testkey.pem -out oauthkey.pem -topk8 -nocrypt -outform PEM
```

2.12.2. Configure the property file

Edit container.js and change the following parameter to point to your private key and your key name.

```
"gadgets.signingKeyFile" : "oauth.pem",
"gadgets.signingKeyName" : "oauthKey",
```

Chapter 3. Developers References

3.1. UI Extensions

3.1.1. About Activity Plug-in

Since the Social 1.1.0, the activity plug-in feature was introduced that enables the activity composer extension and custom UI component to be used for displaying one activity by its type.

3.1.2. Create the activity plug-in

At first, you should have an idea about the UI Extension Framework. If you have already worked with the UI Extension Framework, it's really easy to create the activity plug-in. If no, you have chance to work with it visiting UI Extension Framework.

3.1.2.1. Create a custom UI component for displaying the activity based on its type.

Note

("Project Code can be downloadable here")

When an activity is displayed, UIActivityFactory will look for its registered custom activity display by activity's type. If not found, UIDefaultActivity will be called for displaying that activity.

For example, in Social, there is an activity of type "exosocial:spaces" created by SpaceActivityPublisher. If you want to display it with our own UI component instead of default ones.

First, create a sample project:

```
mvn archetype:generate

Choose archetype:
Choose a number: 76
Choose version: 1
Define value for property 'groupId': : org.exoplatform.social.samples
Define value for property 'artifactId': : exo.social.samples.activity-plugin
Define value for property 'version': 1.0-SNAPSHOT: 1.0.0-SNAPSHOT
Define value for property 'package': org.exoplatform.social.samples: org.exoplatform.social.samples.activityPlug
Confirm properties configuration:
groupId: org.exoplatform.social.samples
artifactId: exo.social.samples.activity-plugin
version: 1.0.0-SNAPSHOT
package: org.exoplatform.social.samples.activityPlugin
Y: y
```

Edit the pom.xml file as follows.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:sch
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.exoplatform.social</groupId>
    <artifactId>social-project</artifactId>
    <version>1.1.0-GA</version>
  </parent>
```

```
  <groupId>org.exoplatform.social.samples</groupId>
  <artifactId>exo.social.samples.activity-plugin</artifactId>
  <packaging>jar</packaging>
  <version>1.1.0-GA</version>

  <name>exo.social.samples.activity-plugin</name>

  <build>
    <sourceDirectory>src/main/java</sourceDirectory>
    <outputDirectory>target/classes</outputDirectory>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
        <includes>
          <include>**/*.gtmpl</include>
        </includes>
      </resource>
      <resource>
        <directory>src/main/java</directory>
        <includes>
          <include>**/*.xml</include>
        </includes>
      </resource>
    </resources>
  </build>

  <dependencies>

    <dependency>
      <groupId>org.exoplatform.portal</groupId>
      <artifactId>exo.portal.webui.core</artifactId>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.exoplatform.portal</groupId>
      <artifactId>exo.portal.webui.portal</artifactId>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.exoplatform.social</groupId>
      <artifactId>exo.social.component.core</artifactId>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.exoplatform.social</groupId>
      <artifactId>exo.social.component.webui</artifactId>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.exoplatform.social</groupId>
      <artifactId>exo.social.component.service</artifactId>
      <scope>provided</scope>
    </dependency>

  </dependencies>

</project>
```

To use the custom UI component for displaying its activity, we need a subclass of `BaseUIActivity`. We call this `UISpaceSimpleActivity`:

```
package org.exoplatform.social.samples.activityplugin;

import org.exoplatform.social.webui.activity.BaseUIActivity;
import org.exoplatform.webui.config.annotation.ComponentConfig;
import org.exoplatform.webui.core.lifecycle.UIFormLifecycle;

@ComponentConfig(
  lifecycle = UIFormLifecycle.class,
  template = "classpath:groovy/social/plugin/space/UISpaceSimpleActivity.gtmpl"
```

```
)
public class UISpaceSimpleActivity extends BaseUIActivity {

}
```

The template `UISpaceSimpleActivity.gtmpl` should be created under `main/resources/groovy/social/plugin/space`:

```
<div>This is a space activity ui component displayed for type "exosocial:spaces"</div>
```

An activity builder is also needed which will be explained later.

```
package org.exoplatform.social.samples.activityplugin;

import org.exoplatform.social.core.activity.model.Activity;
import org.exoplatform.social.webui.activity.BaseUIActivity;
import org.exoplatform.social.webui.activity.BaseUIActivityBuilder;

public class SimpleSpaceUIActivityBuilder extends BaseUIActivityBuilder {

  @Override
  protected void extendUIActivity(BaseUIActivity uiActivity, Activity activity) {
    // TODO Auto-generated method stub

  }

}
```

Next, create `configuration.xml` under conf/portal:

```
<configuration xmlns="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSche
  <external-component-plugins>
    <target-component>org.exoplatform.webui.ext.UIExtensionManager</target-component>
    <component-plugin>
      <name>add.action</name>
      <set-method>registerUIExtensionPlugin</set-method>
      <type>org.exoplatform.webui.ext.UIExtensionPlugin</type>
      <init-params>
        <object-param>
          <name>Simple Space Activity</name>
          <object type="org.exoplatform.social.webui.activity.UIActivityExtension">
            <field name="type"><string>org.exoplatform.social.webui.activity.BaseUIActivity</string></field>
            <field name="name"><string>exosocial:spaces</string></field>
            <field name="component"><string>org.exoplatform.social.samples.activityplugin.UISpaceSimpleActivity<
            <field name="activityBuiderClass"><string>org.exoplatform.social.samples.activityplugin.SimpleSpaceU
          </object>
        </object-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
</configuration>
```

Note that `exosocial:spaces` is defined in

```
<field name="name"><string>exosocial:spaces</string></field>
```

, its value must match the activity's type you want to display with your UI component.

Build sample project and copy the jar to tomcat/lib. Run Social, create a space and access it. You can see:

space's activity of type "exosocial:spaces" is displayed by default in Social:

With our custom UI component for displaying activity of type: "exosocial:spaces":

1.1.1 Make the custom UI activity display have the look and feel and function like default one.

When displaying an activity, we should make sure the look and feel of the custom UI component is consistent and match other activities and have the functions of like, comments. So, to create the another UI component to display, we call `UISpaceLookAndFeelActivity`:

```
package org.exoplatform.social.samples.activityplugin;

import org.exoplatform.social.webui.activity.BaseUIActivity;
import org.exoplatform.webui.config.annotation.ComponentConfig;
import org.exoplatform.webui.config.annotation.EventConfig;
import org.exoplatform.webui.core.lifecycle.UIFormLifecycle;

@ComponentConfig(
  lifecycle = UIFormLifecycle.class,
  template = "classpath:groovy/social/plugin/space/UISpaceLookAndFeelActivity.gtmpl",
  events = {
    @EventConfig(listeners = BaseUIActivity.ToggleDisplayLikesActionListener.class),
    @EventConfig(listeners = BaseUIActivity.ToggleDisplayCommentFormActionListener.class),
    @EventConfig(listeners = BaseUIActivity.LikeActivityActionListener.class),
    @EventConfig(listeners = BaseUIActivity.SetCommentListStatusActionListener.class),
    @EventConfig(listeners = BaseUIActivity.PostCommentActionListener.class),
    @EventConfig(listeners = BaseUIActivity.DeleteActivityActionListener.class, confirm = "UIActivity.msg.Are_Yo
    @EventConfig(listeners = BaseUIActivity.DeleteCommentActionListener.class, confirm = "UIActivity.msg.Are_You
  }
)
public class UISpaceLookAndFeelActivity extends BaseUIActivity {

}
```

Now, create the `UISpaceLookAndFeelActivity` template by copying the content of UIDefaultActivity.gtmpl to this template file.

You should make needed modifications for this template. We make a small modification here:

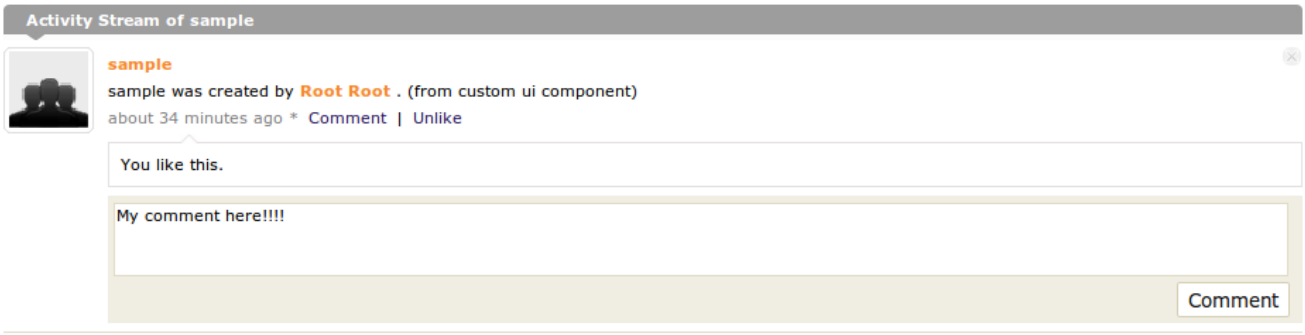
```
<div class="Content">
  $activityContentTitle (from custom ui component)<br>
</div>
```

We need to reconfigure `configuration.xml`:

```
```

```
<configuration xmlns="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSche
  <external-component-plugins>
    <target-component>org.exoplatform.webui.ext.UIExtensionManager</target-component>
    <component-plugin>
      <name>add.action</name>
      <set-method>registerUIExtensionPlugin</set-method>
      <type>org.exoplatform.webui.ext.UIExtensionPlugin</type>
      <init-params>
        <object-param>
          <name>Look And Feel Space Activity</name>
          <object type="org.exoplatform.social.webui.activity.UIActivityExtension">
            <field name="type"><string>org.exoplatform.social.webui.activity.BaseUIActivity</string></field>
            <field name="name"><string>exosocial:spaces</string></field>
            <field name="component"><string>org.exoplatform.social.samples.activityplugin.UISpaceLookAndFeelActi
            <field name="activityBuiderClass"><string>org.exoplatform.social.samples.activityplugin.SimpleSpaceU
          </object>
        </object-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
</configuration>
```

Rebuild the sample project, copy jar to tomcat/lib. Rerun the server and see the result:



Note

Currently, we have to copy and paste in the template file. We will have the full control of the UI, but it is not good when there is any change in UIDefaultActivity.

3.1.2.1.1. What is activity builder?

Note

to do

3.1.2.2. Create a composer extension for composing activity on the UI composer and display it on activity stream.

Note

to do

3.2. Overridable Components

There are 2 components in Social that can be overridden: Space Application Handler & Space Service

- **Space Application Handler**

```
<component>
    <key>org.exoplatform.social.core.space.spi.SpaceApplicationHandler</key>
    <type>org.exoplatform.social.core.space.impl.DefaultSpaceApplicationHandler</type>
  </component>
```

- **Space Service**

```
<component>
    <key>org.exoplatform.social.core.space.spi.SpaceService</key>
    <type>org.exoplatform.social.core.space.impl.SpaceServiceImpl</type>
    <init-params>
      <!-- Configure the applications to install in a space -->
      <values-param>
        <name>space.homeNodeApp</name>
        <value>SpaceActivityStreamPortlet</value>
      </values-param>
      <!-- Configure removable application or not <value>Application:removable</value> -->
      <values-param>
        <name>space.apps</name>
        <value>DashboardPortlet:true</value>
        <value>SpaceSettingPortlet:false</value>
        <value>MembersPortlet:true</value>
      </values-param>
    </init-params>
  </component>
```

3.3. Public REST APIs

3.3.1. Service name and description

3.3.1.1. Activities REST service

Name	Service URL	Description
ActivitiesRestService	$portalName/social/activities	To provide rest services for activity applications: like/unlike; comment; delete activity.

- **API**:

Name	Service URL Endpoint	Description
destroyActivity	restContextName/social/activities/$activityId/destroy/$identity.$format	To destroy activity by identity. The json/xml format.
showLikes	restContextName/social/activities/$activityId/likes.$format	To show the list of likes by activityId and return the json/xml format.
updateLike	restContextName/social/activities/$activityId/updateLike.$format	To update the list of likes by the json/xml format.

Name	Service URL Endpoint	Description
destroyLike	restContextName/social/activities/$activityId/likesdestroy/$identity.$format	To destroy likes by identity and get the json/xml return format.
showComments	restContextName/social/activities/$activityId/comments.$format	To show comments list by the json/xml format.
updateComment	restContextName/social/activities/$activityId/commentsupdate/$commentId.$format	To update the comment by the json/xml format.
destroyComment	restContextName/social/activities/$activityId/commentsdestroy/$commentId.$format	To destroy comments and return the json/xml format.

Parameter	Expected values
`format`	`xml/json`

Example:

- http://localhost:9090/rest/social/activities/s08d397dg6/likesdestroy/abc.xml
- http://localhost:9090/rest/social/activities/s08d397dg6/likesdestroy/abc.json

3.3.1.2. Apps REST service

Name	Service URL	Description
AppsRestService	$restContextName/social/apps/show.$format	To provide rest services for the application registry gadget: shows application list

- **API**:

Name	Service URL Endpoint	Description
showApps	$restContextName/social/apps/show.$format	To show applications by the json/xml format.

Parameter	Expected values
`format`	`xml/json`

Example:

http://localhost:9090/rest/apps/show.json

3.3.1.3. Identity REST service

Name	Service URL	Description
IdentityRestService	$portalName/social/identity/$username/id	To get identityId by the username.

- **API**:

Name	Service URL Endpoint	Description	Example
UserId getId	$portalName/social/identity/$username/show.json	To get the identity by username and return by the json format.	

Example:

http://localhost:9090/Socialdemo/social/identity/John/show.json

3.3.1.4. Linkshare REST service

Name	Service URL	Description
LinkshareRestService	$restContextName/social/linkshare	To get information from a provided link.

- **API**:

Name	Service URL Endpoint	Description
getLink	$restContextName/social/linkshare/show.$format	To get the link content by posting with linkShare request as post data

Parameter	Expected values
`format`	`xml/json`

Example:

http://localhost:9090/rest/social/linkshare/show.json

3.3.1.5. Spaces REST service

Name	Service URL	Description
SpacesRestService	$portalName/social/spaces	To provide rest services for space gadget to display user's spaces and pending spaces

- **API**:

Name	Service URL Endpoint	Description
showMySpaceList	$restContextName/social/spaces/$userId/mySpaces/show.$format	To show mySpaces list by the json/xml format.
showPendingSpaceList	$restContextName/social/spaces/$userId/pendingSpaces/show.$format	To show pendingSpaces list by the json/xml format.

Parameter	Expected values
`format`	`xml/json`

Example:

http://localhost:8080/rest/social/spaces/s08d397dg6/mySpaces/show.xml

3.3.1.6. Widget REST service

Name	Service URL	Description
WidgetRestService	{restContextName}/spaces/{portalName}	To provide rest services for creating spaces or getting space's information.

- **API**:

Name	Service URL Endpoint	Description
goToSpace	{restContextName}/spaces/{portalName}/go_to_space	To go to (if existing) or access a space. Two query parameters needed: `spaceName` and `description`
spaceInfo	{restContextName}/spaces/{portalName}/space_info	To get info HTML page for displaying the information of space. Two query parameters needed: `spaceName` and `description`

Parameter	Expected values
`portalName`	`portal/socialdemo`

Example:

http://localhost:8080/rest-socialdemo/spaces/socialdemo/go_to_space?name=Social&description=Social

http://localhost:8080/rest-socialdemo/spaces/socialdemo/space_info?name=Social&description=Social

3.3.2. Location

- **Maven groupId**: org.exoplatform.social
- **ArtifactId**: exo.social.component.service

3.4. Sample code/tutorial

h3, Activity Stream

eXo Social provides a way to share status updates and activity information for users as well as spaces (aka Activity Streams). With the API, you can customize the activities or publish new ones.

To manipulate activities, you will use the ActivityManager. To get an instance of this class, you will need to use the `PortalContainer`.

3.4.1. Create an activity

There are two types of activities : activities for a user and activities for a space. The following examples will show you how to create an activity for each type.

3.4.2. Publish an activity for a user

Users have activity streams. The code below shows you how to publish a nex activity into the public activity stream of a user.

```
import org.exoplatform.container.PortalContainer;
import org.exoplatform.social.core.activitystream.ActivityManager;
import org.exoplatform.social.core.activitystream.model.Activity;
import org.exoplatform.social.core.identity.IdentityManager;
import org.exoplatform.social.core.identity.impl.organization.OrganizationIdentityProvider;
import org.exoplatform.social.core.identity.model.Identity;


  public void createActivityForUser() {
    String username = "zun";
    // Get current container
    PortalContainer container = PortalContainer.getInstance();

    // Get IdentityManager to handle identity operation
    IdentityManager identityManager = (IdentityManager) container.getComponentInstance(IdentityManager.class);

    // Get ActivityManager to handle activity operation
    ActivityManager activityManager = (ActivityManager) container.getComponentInstanceOfType(ActivityManager.cla

    // Get existing user or create a new one
    try {
      Identity userIdentity = identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME, username);

      // Create new activity for this user
      Activity activity = new Activity();
      activity.setUserId(userIdentity.getId());
      activity.setTitle("Hello World!");
      // Save activity into JCR using ActivityManager
      activityManager.saveActivity(activity);
    } catch (Exception e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
```

3.4.3. Publish an activity for a space

Spaces are also social objects and thus, they own an activity stream. The code below shows you how to publish activity into a space's stream

```
import org.exoplatform.container.PortalContainer;
import org.exoplatform.social.core.activitystream.ActivityManager;
import org.exoplatform.social.core.activitystream.model.Activity;
import org.exoplatform.social.core.identity.IdentityManager;
import org.exoplatform.social.core.identity.impl.organization.OrganizationIdentityProvider;
import org.exoplatform.social.core.identity.model.Identity;
import org.exoplatform.social.space.Space;
import org.exoplatform.social.space.SpaceException;
import org.exoplatform.social.space.SpaceService;
import org.exoplatform.social.space.impl.SpaceIdentityProvider;

//...

  public void createActivityForSpace() {
    //make sure a space with name "mySpace" is created.
    String spaceName = "mySpace";
    String username = "zun";
    // Get current container
    PortalContainer container = PortalContainer.getInstance();

    // Get IdentityManager to handle identity operation
    IdentityManager identityManager = (IdentityManager) container.getComponentInstance(IdentityManager.class);

    // Get ActivityManager to handle activity operation
    ActivityManager activityManager = (ActivityManager) container.getComponentInstanceOfType(ActivityManager.cla

    // Get SpaceService to handle space operation
    SpaceService spaceService = (SpaceService) container.getComponentInstanceOfType(SpaceService.class);
    try {
      Space space = spaceService.getSpaceByName(spaceName);
      if (space != null) {
        // Get space identity via SpaceIdentityProvider
        Identity spaceIdentity = identityManager.getOrCreateIdentity(SpaceIdentityProvider.NAME, spaceName);
        // Get identity instance of the user who wants to create activity
        Identity userIdentity = identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME, username)
        // Create new activity for this space
        Activity activity =  new Activity();
        activity.setUserId(userIdentity.getId());
        activity.setTitle("An activity for space");
        activityManager.saveActivity(spaceIdentity, activity);
      }
    } catch (SpaceException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (Exception e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
```

3.4.4. Useful functions

- ActivityManager#getActivity
- ActivityManager#getActivities
- ActivityManager#saveActivity
- ActivityManager#saveComment
- ActivityManager#saveLike

- ActivityManager#removeLike
- ActivityManager#getComments

3.4.4.1. Creating a custom activity processor

Activity processor is used for modifying the content of activities before they are responsed and rendered at client's browser. For example, we will create an activity processor for replacing all the texts representing the smile face ":-)" in the activity title by the smiley icons.

Firstly, we will create the `SmileyProcessor` class by extending the `BaseActivityProcessorPlugin`

```
package org.exoplatform.social.core.activitystream;

public class SmileyProcessor extends BaseActivityProcessorPlugin {
  public SmileyProcessor(InitParams params) {
    super(params);
  }

  String smiley =  "<img src="http://www.tombraider4u.com/pictures/smiley.gif"/>";

  public void processActivity(Activity activity) {
    String title = activity.getTitle();
    activity.setTitle(title.replaceAll(":-
)", smiley));
  }
}
```

And then, we have to register this processor by adding some XML configuration into the project configuration file (configuration.xml)

```
<component>
  <key>org.exoplatform.social.core.activitystream.ActivityManager</key>
  <type>org.exoplatform.social.core.activitystream.ActivityManager</type>
  <component-plugins>
    <component-plugin>
      <name>SmileyProcessor</name>
      <set-method>addProcessorPlugin</set-method>
      <type>org.exoplatform.social.core.activitystrea.SmileyProcessor</type>
      <init-params>
        <values-param>
          <name>priority</name>
          <value>1</value>
        </values-param>
      </init-params>
    </component-plugin>
  </component-plugins>
</component>
```

"init-params" contains all the key-value data which a processor will use to initialize. At the above config, priority value indicates the order that this processor will be used. So with '1' value, this processor will be used before all remaining processors with lower priority.

3.4.5. Publish an rss feed with feedmash

It's really easy to publish an rss feed to a space's activity stream. eXo Social already provides `FeedmashJobPlugin` for publishing rss feeds. As you can see in project `exo.social.extras.feedmash`, there are `JiraFeedConsumer` and `HudsonFeedConsumer` samples to post eXo Social project's feeds (jira and hudson) to a pre-defined space: `exosocial` in a specifiportal container: `socialdemo` as in the configuration file:

```
<external-component-plugins>
   <target-component>org.exoplatform.services.scheduler.JobSchedulerService</target-component>
    <component-plugin>
      <name>RepubSocialJiraActivityJob</name>
      <set-method>addPeriodJob</set-method>
      <type>org.exoplatform.social.feedmash.FeedmashJobPlugin</type>
      <description/>
      <init-params>
        <properties-param>
          <name>mash.info</name>
          <property name="feedURL" value="http://jira.exoplatform.org/plugins/servlet/streams?key=SOC"/>
          <property name="categoryMatch" value="resolved|created"/>
          <property name="targetActivityStream" value="space:exosocial"/>
          <property name="portalContainer" value="socialdemo"/>
        </properties-param>
        <properties-param>
          <name>job.info</name>
          <description>save the monitor data  periodically</description>
          <property name="jobName" value="JIRAFeedConsumer"/>
          <property name="groupName" value="Feedmash"/>
          <property name="job" value="org.exoplatform.social.feedmash.JiraFeedConsumer"/>
          <property name="repeatCount" value="0"/>
          <property name="period" value="60000"/>
          <property name="startTime" value="+45"/>
          <property name="endTime" value=""/>
        </properties-param>
      </init-params>
    </component-plugin>
    <component-plugin>
      <name>WatchSocialBuildStatus</name>
      <set-method>addPeriodJob</set-method>
      <type>org.exoplatform.social.feedmash.FeedmashJobPlugin</type>
      <description/>
      <init-params>
        <properties-param>
          <name>mash.info</name>
           <property name="feedURL" value="http://builder.exoplatform.org/hudson/view/social/job/social-trunk-ci
           <property name="targetActivityStream" value="space:exosocial"/>
           <property name="portalContainer" value="socialdemo"/>
        </properties-param>
        <properties-param>
          <name>job.info</name>
          <description>save the monitor data  periodically</description>
          <property name="jobName" value="HudsonFeedConsumer"/>
          <property name="groupName" value="Feedmash"/>
          <property name="job" value="org.exoplatform.social.feedmash.HudsonFeedConsumer"/>
          <property name="repeatCount" value="0"/>
          <property name="period" value="60000"/>
          <property name="startTime" value="+100"/>
          <property name="endTime" value=""/>
        </properties-param>
      </init-params>
    </component-plugin>
 </external-component-plugins>
```

When running eXo Social, login with http://localhost:8080/socialdemo and create a space named "exosocial". Done, all the feeds from jira and hudson for Social project will be automatically published to `exosocial` space.

3.4.6. People

eXo Social provides a way to add profile informations, relationships and connections between users: People. With the eXo People API, profile informations and relationship can be easily managed and customized.

3.4.6.1. Identity

The identity allows to identity uniquely a social object. Social objects can be persons, groups, applications, or whatever you think deserves social interactions like connecting, publishing an activity stream or holding a profile.

3.4.6.1.1. IdentityProvider

Creating your Identity Provider allows you to integrate people outside of your portal (for exemple customers) into your social network without having to create a portal account. You can also use this to populate the profile with data coming from other systems. Here is an example :

```
class SampleIdentityProvider extends OrganizationIdentityProvider{
  public SampleIdentityProvider(OrganizationService organizationService) {
    super(organizationService);
  }

  @Override
  public void populateProfile(Profile profile, User user) {
    profile.setProperty(Profile.FIRST_NAME, "this is first name");
    profile.setProperty(Profile.LAST_NAME, "this is last name");
    profile.setProperty(Profile.USERNAME, "this is user name");
    profile.setProperty(Profile.URL, "/path/to/profile/");
  }
}
```

In this example, we created a SampleIdentityProvider class extending OrganizationIdentityProvider which is the IdentityProvider used to connect to the portal user's base. In this class, we overrided the populateProfile method and added some dummy data in the profile fields.

3.4.6.1.2. IdentityManager

The IdentityManager is the service used to manipulate the identity operations like creating, getting, deleting or finding a profile. We can get the IdentityManager via the ExoContainer. The following code will show how to get an IdentityManager instance and create a basic identity instance :

```
import org.exoplatform.container.ExoContainer;
import org.exoplatform.container.ExoContainerContext;
import org.exoplatform.social.core.identity.IdentityManager;
import org.exoplatform.social.core.identity.impl.organization.OrganizationIdentityProvider;
import org.exoplatform.social.core.identity.model.Identity;

//......

String containerName = "portal";
String username = "zun";

//get container to get other registered components
ExoContainer container = ExoContainerContext.getContainerByName(containerName);

//get IdentityManager to handle identity operation
IdentityManager identityManager = (IdentityManager) container.getComponentInstanceOfType(IdentityManager.class);

//get ActivityManager to handle activity operation
ActivityManager activityManager = (ActivityManager) container.getComponentInstanceOfType(ActivityManager.class);

//create new user with name Zun
Identity userIdentity = identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME, username);
```

3.4.6.2. ProfileListener

The People API provides some notification interfaces which programmers can implement in order to create their own handlers for notifications like notifications() for profile modifications(ProfileListenerPlugin]) or for relationship changes(RelationshipListenerPlugin]). The following example will guide you through implementing one of these interfaces and show you how to configure this plugin.

We will create the class ProfileLoggerListener. Its tasks is to log all profile modifications of the systems. The

abstract class ProfileListenerPlugin provides us the interface to implement this method.

```
import org.exoplatform.services.log.ExoLogger;
import org.exoplatform.services.log.Log;
import org.exoplatform.social.core.identity.lifecycle.ProfileListenerPlugin;
import org.exoplatform.social.core.identity.spi.ProfileLifeCycleEvent;


public class ProfileLoggerListener extends ProfileListenerPlugin{
  private static final Log logger = ExoLogger.getExoLogger(ProfileLoggerListener.class);
  @Override
  public void avatarUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " profile has updated his basic profile info.");
  }

  @Override
  public void basicInfoUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " profile has updated his basic profile info.");
  }

  @Override
  public void contactSectionUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " profile has updated his contact info.");
  }

  @Override
  public void experienceSectionUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " profile has an updated experience section.");
  }

  @Override
  public void headerSectionUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " has updated his header info.");
  }
}
```

After creating the ProfileLoggerListener class, we have to add some configurations for this class to the configuration.xml :

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.identity.IdentityManager</target-component>
  <component-plugin>
    <name>ProfileLoggerListener</name>
    <set-method>addProfileListener</set-method>
    <type>path.to.ProfileLoggerListener</type>
  </component-plugin>
</external-component-plugins>
```

3.4.6.3. Relationships

Similarly, you can apply the above steps to implement the RelationshipListenerPlugin for relationship notifications.

Relationship is the bridge between two identities in eXo Social. There are many types of relationships defined in the Relationship class. With these types, a user can invite another user, confirm invitations or remove relationship.

3.4.6.3.1. Connecting users

The following code will show how to invite a user to connect to another :

```
import org.exoplatform.container.ExoContainer;
import org.exoplatform.container.ExoContainerContext;
import org.exoplatform.social.core.identity.IdentityManager;
import org.exoplatform.social.core.identity.impl.organization.OrganizationIdentityProvider;
```

```
import org.exoplatform.social.core.identity.model.Identity;
import org.exoplatform.social.core.relationship.Relationship;
import org.exoplatform.social.core.relationship.RelationshipManager;

public void inviteUser() throws Exception {
  String containerName = "portal";
  ExoContainer container = ExoContainerContext.getContainerByName(containerName);

  IdentityManager identityManager = (IdentityManager) container.getComponentInstanceOfType(IdentityManager.class

  Identity invitedIdentity = identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME, "Hoat");
  String invitedUserId = invitedIdentity.getId();

  String currUserId = "Zun";
  Identity currentIdentity = identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME,  currUserId)

  RelationshipManager relationshipManager = (RelationshipManager) container.getComponentInstanceOfType(Relations
  Relationship relationship = relationshipManager.invite( currentIdentity, invitedIdentity);
}
```

3.4.6.3.2. RelationshipListener

Let's take a look at relationship Listener(RelationshipListenerPlugin). The following example will show you how to implement one of these interfaces and how to configure this plugin. The following step is similar to the Profile notifications implementation.

```
import org.exoplatform.social.core.relationship.Relationship;
import org.exoplatform.social.core.relationship.lifecycle.RelationshipListenerPlugin;
import org.exoplatform.social.relationship.spi.RelationshipEvent;

public class RelationshipLoggerListener extends RelationshipListenerPlugin{
  private static final Log logger = ExoLogger.getExoLogger(RelationshipLoggerListener.class);

  @Override
  public void confirmed(RelationshipEvent event) {
    String[] names = getUserNamesFromEvent(event);
    logger.info(names[0] +" confirmed the invitation of "+ names[1]);
  }

  @Override
  public void ignored(RelationshipEvent event) {
    String[] names = getUserNamesFromEvent(event);
    logger.info(names[0] +" ignored the invitation of "+ names[1]);
  }

  @Override
  public void removed(RelationshipEvent event) {
    String[] names = getUserNamesFromEvent(event);
    logger.info(names[0] +" removed the relationship with "+ names[1]);
  }

  private String[] getUserNamesFromEvent(RelationshipEvent event){
    Relationship relationship = event.getPayload();

    Identity id1 = relationship.getIdentity1();
    Identity id2 = relationship.getIdentity2();

    String user1 = "@" + id1.getRemoteId();
    String user2 = "@" + id2.getRemoteId();

    return new String[]{user1, user2 };
  }
}
```

After creating the RelationshipLoggerListener class, we have to add some configurations for this class to the configuration.xml :

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.relationship.RelationshipManager</target-component>
  <component-plugin>
    <name>RelationshipLoggerListener</name>
```

```
    <set-method>addListenerPlugin</set-method>
    <type>classpath.of.your.RelationshipLoggerListener</type>
  </component-plugin>
</external-component-plugins>
```

3.4.7. Spaces

eXo Social provides a way to create groups and to share data and applications: the space. A space has it's own activity stream in which applications or members can publish information. In each space, members share applications and an openSocial dashboard.

To manipulate the spaces, you will use the SpaceService. To get an instance of this class, you will need to get current PortalContainer instance.

3.4.7.1. Spaces Management

3.4.7.1.1. Creating a space

The following example shows how to create a space:

```
package org.exoplatform.social.sample;

import org.exoplatform.container.PortalContainer;
import org.exoplatform.social.application.impl.DefaultSpaceApplicationHandler;
import org.exoplatform.social.space.Space;
import org.exoplatform.social.space.SpaceException;
import org.exoplatform.social.space.SpaceService;

public class SpaceCreationSample {

  public void createSpace() throws SpaceException {
    String spaceName = "mySpace";
    String creator = "jeremi";
    PortalContainer container = PortalContainer.getInstance();
    SpaceService spaceService = (SpaceService) container.getComponentInstanceOfType(SpaceService.class);
    // We verify if there is no space already created
    Space space = spaceService.getSpaceByName(spaceName);
    if (space == null) {
      space = new Space();
      space.setName(spaceName);
      space.setRegistration(Space.OPEN);
      space.setDescription("space description");
      //DefaultSpaceApplicationHander is the default implementation of SpaceApplicationHandler. You can create y
      space.setType(DefaultSpaceApplicationHandler.NAME);
      //We create the space
      space = spaceService.createSpace(space, creator);
      //We initialize the applications
      spaceService.initApp(space);

    }
  }
}
```

3.4.7.2. Space's applications management

3.4.7.2.1. Adding an application to a space

We can add a portlet or gadget applications into spaces. Once added, all members of the space can use that application. The following code shows how to add an application into a space :

```
public void addApplicationToSpace() throws SpaceException {
  //Your portlet name
  String appId = "AnyGadgetOrAnyPortletName";
```

```
  String spaceId = "zunSpace";

  //get container to get other registered components
  PortalContainer container = PortalContainer.getInstance();

  //get space service for installing operations
  SpaceService spaceService = (SpaceService) container.getComponentInstanceOfType(SpaceService.class);

  //install application for the space
  spaceService.installApplication(spaceId, appId);

  //we must activate installed application to be able to use it
  spaceService.activateApplication(spaceId, appId);
}
```

Note

appId is the portlet or gadget name as defined in portlet.xml or gadget.xml in the web-app. With eXo Social, we can find it in social-portlet.war and social.war.

3.4.7.3. Members Management

The SpaceService allows you to manage the spaces' members.Here is how you would add a now member to a space:

```
String spaceName = "mySpace";

PortalContainer container = PortalContainer.getInstance();
SpaceService spaceService = (SpaceService) container.getComponentInstanceOfType(SpaceService.class);

Space space = service.getSpaceByName(spaceName);
if (space != null) {
    spaceService.addMember(space, "mary");
}
```

3.4.7.4. Listener to a space lifecycle

To receive notifications of what is happening in spaces, you need to extend SpaceListenerPlugin and register it. Every method takes a SpaceLifeCycleEvent object as parameter that contain the information about the event and it's context.

The events available are:

- spaceCreated: this event is called right after space is created successfully with its applications.
- spaceRemoved: this event is called right after a space is removed: application's removed, its group and group navigation is removed
- applicationAdded: this event is called right after an application is added (installed) to a space
- applicationActivated: this event is called right after an application is activated.
- applicationDeactivated: this event is called right after an application is deactivated.
- applicationRemoved: this event is called right after an application is removed from a space.
- joined: this event is called right after a user joins a space.

- left: this event is called right after a space member leaves its space.
- grantedLead: this event is called right after a user is grated space's manager role.
- revokedLead: this event is called right after a space's manager is revoked his role to be a space member.

```
import org.exoplatform.social.space.lifecycle.SpaceListenerPlugin;

public class MySpaceListenerPlugin extends SpaceListenerPlugin {
...
}
```

As an example, see the SpaceActivityPublisher that publishes an activity based on an event that happened in a space.

To register your listener, hook it to the SpaceService like this :

```
<external-component-plugins>
  <target-component>org.exoplatform.social.space.SpaceService</target-component>
  <component-plugin>
    <name>SpaceActivityPublisher</name>
    <set-method>addSpaceListener</set-method>
    <type>org.mycompany.MySpaceListenerPlugin</type>
  </component-plugin>
</external-component-plugins>
```

3.4.8. Open Social

3.4.8.1. Opensocial

eXo Social is implementing the OpenSocial standard. So you can integrate OpenSocial Gadget in your dashboard and use the RPC or REST API to access to the social data.

3.4.8.1.1. Gadget

Gadgets are web-based software components based on HTML, CSS, and JavaScript. They allow developers to easily write useful web applications that work anywhere on the web without modification. To know more, we suggest some links for detail information :

Gadgets Specification

OpenSocial Core Gadget Specification 10

After getting acquainted with Gadget concept, we enter into the detail on how to create an opensocial gadget. However, the tutorials are done greatly in Opensocial official site so we refer you to read these tutorials at that website

We only note that in Opensocial gadgets only work in the dashboard in eXo Social.

3.4.8.1.1.1. Supported APIs

As we have said above, eXo Social is implementing the OpenSocial standard. So every eXo Social implementations apply the Opensocial Specification generally or Apache Shindig specifically. Therefore, eXo Social uses and extends Apache Shindig APIs to compatible with the common Opensocial APIs which is supported by other big social networks like Ning, Hi5, Orkut ...

To get more detail about Supported APIs, we refer you to read Opensocial Specs

3.4.8.1.2. REST/RPC API

If your eXo social server is running on http://localhost:8080/ the address of the API will be:

REST API: http://localhost:8080/social/social/rest

RPC API: http://localhost:8080/social/social/rpc

To learn what you can do with this APIs, have a look at the specification. If you are developing in Java, you can use the opensocial-java-client

3.4.8.1.2.1. Configuring the security

If you are using opensocial, there is good chance you are going to need to configure the oAuth authentication. To do this, you need to edit the configuration to add you oAuth key.

Edit the file: gatein/conf/portal/portal/configuration.xml and add this component:

```
        <component>
                <key>org.exoplatform.social.opensocial.oauth.ServiceProviderStore</key>
                <type>org.exoplatform.social.opensocial.oauth.ServiceProviderStore</type>

                <init-params>
                        <properties-param>
                                <name>grails-book-flow</name>
                                <description>consmer key and secret for sample oauth provider. </description>
                                <property name="consumerKey" value="YOUR_KEY_HERE" />

                                <property name="sharedSecret" value="YOUR_SECRET_KEY_HERE" />
                        </properties-param>

                </init-params>
        </component>
```

The consumerKey and sharedSecret are the key that need to be shared with the application that is doing the request.

3.4.8.1.2.2. Publishing an activity into a space

eXo Social added this functionality that is not available in the standard opensocial API. You can publish activities into a space using the opensocial API.

To do this, instead of publishing your activity to the group @self as usual, publish it to the group "space:spaceID" or "space:spaceName".

Using the opensocial java library and groovy, your code will look like this:

```
      def client = getOpenSocialClient()

      //we create our new activity
      Activity activity = new Activity()
      activity.title = "BookFlow Purchase"
      activity.body = "xx puchased the book xxx"

      //We prepare the request that will create the activity
      Request request = ActivitiesService.createActivity(activity);
      //We specify that the creation of this new activity is for the space bookflow
      request.groupId = "space:bookflow";

      client.send(request);
```

```
```

As you can see in this example, we set the groupId to "space:bookflow", bookflow being the name of our space.

3.4.8.1.2.3. Tutorial

- Grails + eXo Social tutorial

eXo Social is implementing the OpenSocial standard. So you can integrate OpenSocial Gadget in your dashboard and use the RPC or REST API to access to the social data.

3.4.8.2. Gadget

Gadgets are web-based software components based on HTML, CSS, and JavaScript. They allow developers to easily write useful web applications that work anywhere on the web without modification. To know more, we suggest some links for detail information :

Gadgets Specification

OpenSocial Core Gadget Specification 10

After getting acquainted with Gadget concept, we enter into the detail on how to create an opensocial gadget. However, the tutorials are done greatly in Opensocial official site so we refer you to read these tutorials at that website

We only note that in Opensocial gadgets only work in the dashboard in eXo Social.

3.4.8.2.1. Supported APIs

As we have said above, eXo Social is implementing the OpenSocial standard. So every eXo Social implementations apply the Opensocial Specification generally or Apache Shindig specifically. Therefore, eXo Social uses and extends Apache Shindig APIs to compatible with the common Opensocial APIs which is supported by other big social networks like Ning, Hi5, Orkut ...

To get more detail about Supported APIs, we refer you to read Opensocial Specs

3.4.8.3. REST/RPC API

If your eXo social server is running on http://localhost:8080/ the address of the API will be:

REST API: http://localhost:8080/social/social/rest

RPC API: http://localhost:8080/social/social/rpc

To learn what you can do with this APIs, have a look at the specification. If you are developing in Java, you can use the opensocial-java-client

3.4.8.3.1. Configuring the security

If you are using opensocial, there is good chance you are going to need to configure the oAuth authentication. To do this, you need to edit the configuration to add you oAuth key.

Edit the file: gatein/conf/portal/portal/configuration.xml and add this component:

```
        <component>
```

```
                <key>org.exoplatform.social.opensocial.oauth.ServiceProviderStore</key>
                <type>org.exoplatform.social.opensocial.oauth.ServiceProviderStore</type>

                <init-params>
                        <properties-param>
                                <name>grails-book-flow</name>
                                <description>consmer key and secret for sample oauth provider. </description>
                                <property name="consumerKey" value="YOUR_KEY_HERE" />

                                <property name="sharedSecret" value="YOUR_SECRET_KEY_HERE" />
                        </properties-param>

                </init-params>
        </component>
```

The consumerKey and sharedSecret are the key that need to be shared with the application that is doing the request.

3.4.8.3.2. Publishing an activity into a space

eXo Social added this functionality that is not available in the standard opensocial API. You can publish activities into a space using the opensocial API.

To do this, instead of publishing your activity to the group @self as usual, publish it to the group "space:spaceID" or "space:spaceName".

Using the opensocial java library and groovy, your code will look like this:

```
def client = getOpenSocialClient()

//we create our new activity
Activity activity = new Activity()
activity.title = "BookFlow Purchase"
activity.body = "xx puchased the book xxx"

//We prepare the request that will create the activity
Request request = ActivitiesService.createActivity(activity);
//We specify that the creation of this new activity is for the space bookflow
request.groupId = "space:bookflow";

client.send(request);
```

As you can see in this example, we set the groupId to "space:bookflow", bookflow being the name of our space.

3.4.8.3.3. Tutorial

- Grails + eXo Social tutorial

3.4.9. Space widget tutorial

The eXo Social widget enables developers to add capabilities of eXo Social to external applications. Since the widget is hosted on your eXo Social server, it can display personalized information. An activity stream of the most recent user actions will display to the group's members.

There are two options of the eXo Social widget that provide different levels of integration and information.

The basic version of this widget is an iFrame. The more advanced version is a button you can insert in a page; this will display a small pop-up with information about the space.

3.4.9.1. Basic version

To insert the basic version, you need to have an iFrame to insert on your site.

```
<iframe  scrolling="no" height="180" frameborder="no" width="220" src="http://URL_OF_YOUR_EXO_INSTALLATION.COM/r
```

To install this version in your application, replace all the uppercase text below:

- URL*OF*YOUR*EXO*INSTALLATION.COM - this is the URL of your eXo Social installation. If you are testing on your local computer, the URL may be *localhost:8080*_
- NAME*OF*YOUR*SPACE - this is the title of your space in eXo Social. In the URL, it is necessary to avoid special characters. For the space name, you can only use alphanumeric characters and ""*, ".", "-" or ". "
- DESCRIPTION*OF*THE*SPACE - this will be displayed in the list of spaces.*

3.4.9.2. Advanced version

To install an advanced version of the widget, you need to insert a code snippet in your page. This includes some HTMLs plus some JavaScripts. The necessary CSS will be added dynamically.

Next, insert the following code at the position you want the button to be displayed:

```
<div class="exoSpacesContainer"><a href="javascript:void(0);" id="exoSpacesLink" class="exoSpacesLink" target="_
<script src="/socialWidgetResources/javascript/space.js"></script>
<script>spaces.createPopup("exoSpacesLink", "MyAppName - my social object", "my cool description");</script>
```

The important function here is:

```
spaces.createPopup(link, spaceName, description)
```

- link: link is the ID or the HTMLElement where the pop-up will be placed. If you copy and paste the code snippet provided above, you donot need to change this value.
- spaceName: This is the name of space. It is also used to identify the space. We recommend you use the following format: "MyAppName - my social object"
- description: This is the description of your space that will be displayed within eXo Spaces. It is used when a new space is created.

3.4.9.3. Configure

- serverURL (Default: *"http://127.0.0.1:8080"*): The address of your eXo installation. To change it, use *spaces.setServerURL(...);*
- spaceServicePath (Default: *"/rest/private/spaces/"*): The path to the spaces service. It is rare you have to change it; but if needed, use *spaces.setSpaceServicePath(...);*
- portalName (Default: *"socialdemo"*): The name of portal you are using. To change it, use *spaces.setPortalName(...);*

If you want to change any part of this configuration, the best way is to change before creating the pop-up. For example:

```
<div class="exoSpacesContainer"><a href="#" id="exoSpacesLink" class="exoSpacesLink" target="_blank">Space</a></
<script src="/socialWidgetResources/javascript/space.js"></script>
<script>
  spaces.setServerURL("http://192.168.2.100:8080");
  spaces.createPopup("exoSpacesLink", "My cool new space", "my cool description");
</script>
```

You can see an example of integration at: http://localhost:8080/socialWidgetResources/test.html

3.5. Public Javascript APIs

All references for Opensocial Javascript APIs can be viewed here