exo.social.docs.refguide - 1.1.0-CR03

eXo Social Reference Guide

eXo Platform (eXo Platform)

Acknowledgements

This book is produced by the Wikbook tool. Wikbook is an open source project for converting wiki files into a set of docbook files.

Chapter 1. Prerequisites

Chapter 2. Applications

2.1. List of Portlets in Social

All the portlets are in social-portlet.war

Portlets name	Description	
Space Menu Portlet	Display the menu on the right of a Space: Home, Dashboard, Members, SpaceSettings	
Members Portlet	Display MemberSpace where you can search for space members or list space members in alphabet	
Space Setting Portlet	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.	
My Spaces Portlet	Displays your Space page where you can add new spaces, search for spaces or list spaces in alphabet	
Invitation Spaces Portlet	Displays the **Space invitation** page where you can search for spaces, list spaces in alphabet and view all the Space invitations	
Pending Spaces Portlet	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	
Public Spaces Portlet	Displays the **Public Spaces** page where you can search for spaces, list spaces by alphabet and view all available public spaces to join	
Space Activity Stream Portlet	Display spaces activities	
User Activity Stream Portlet	Update user's activities/status	
People Portlet	The People page where you can find people, display people name by alphabet	
Profile Navigation Portlet		
Connections Portlet	The Conections page where you can access to your network,	

Portlets name	Description	
	invitations to connect or connection requests	
Invitations Portlet		
Requests Portlet		
User Profile Portlet	Display user profile page where you can view/ edit user's basic information, contacts, experience or change avatar	
My Connections Navigation Portlet		

2.2. List of Gadgets in Social

All Social Gadgets are in opensocial.war

2.2.1. 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

2.2.2. 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

2.2.3. 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

2.2.4. UserSpaces

- **Description:**

- **Used REST services:** SpacesRestService

2.2.5. 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

Chapter 3. Configuration

3.1. Components

3.1.1. 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

3.1.2. SpaceService

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

3.1.3. IdentityManager

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

3.1.4. 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	

3.1.5. ServiceProviderStore

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

3.2. External component plugins

3.2.1. MentionsProcessor

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

3.2.2. PortletPreferenceRequiredPlugin

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

3.2.3. AddNodeTypePlugin

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

3.3. Spaces Template configuration

> **Note**
>
> Needs to be updated

In eXo Spaces we will be able to have two type of space (classic and webos spaces). This is for the classic mode (it's the only one implemented now).

For classic space we can pre-configure template, it mean that you can set up where your **menu** will be display or where your **application** will be display.

Here is an example of configuration file that display 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 2 inner containers: one with id is **Menu** for your Menu and another with id is **Application** contains your applications

If you want your menu will be in right and your application in left you can swap two containers declared position:

```
<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

3.4. Configure the oauth 2 legged scenario

Note

not completed yet

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

For more information about this, you can have a look at this great article

3.4.1. Generating 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
```

3.4.2. Configuring 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 4. Developers reference

eXo Social provides many extension point to make it fit to your needs. Every part can be customized and/or automated.

4.1. 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. With the API, you can create spaces, invite users, add applications be notified of events.

> **Note**
>
> The UWC detected velocity templates not surrounded by pre blocks in this page. If you would like it to attempt to convert it, re-run the conversion with the following converter from conf/converter.xwiki.properties commented out: Xwiki.0060.clean-velocity.class=com.atlassian.uwc.converters.xwiki.VelocityCleaner

4.2. 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.

4.2.1. Identity

4.2.1.1. 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.

4.2.1.2. 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);
```

4.2.2. Notifications

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>
```

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

4.2.3. Relationship

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.

4.2.3.1. Invite a user

The following code will show how to invite a user from a user :

```
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);
}
```

4.2.3.2. Notifications

Let's take a look at relationship notifications(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>
```

4.3. 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`.

4.3.1. 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.

4.3.2. Publish an activity for 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;


/**
 * Created by The eXo Platform SAS
 * Author : eXoPlatform
 *          exo@exoplatform.com
 * Jun 25, 2010
 */
//.....

  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();
    }
  }
```

4.3.3. Publish an activity for a space

```

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();
    }
  }
```

4.3.4. Useful functions

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

4.3.4.1. 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.

4.3.5. 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.

4.4. Activity Plugin

4.4.1. About Activity Plugin

Since Social 1.1.0, the activity plugin feature was introduced to allow using activity composer extension and use custom ui component for displaying activity by its type.

4.4.2. How to create activity plugin

You should have an idea about UI Extension Framework. If you have already worked with UI Extension Framework, it's really easy to create activity plugin. If no, you have chance to work with it now . You should have a look at UI Extension Framework.

4.4.2.1. 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 actvity display by activity's type. If not found, UIDefaultActivity will be called for displaying that activity.

For example, in Social there's an activity of type "exosocial:spaces" created by SpaceActivityPublisher. And now we want to display it with out own ui component instead of default one.

First of all, create a sample project:

```
mvn archetype:generate
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] artifact org.apache.maven.plugins:maven-archetype-plugin: checking for updates from central
[INFO] snapshot org.apache.maven.plugins:maven-archetype-plugin:2.0-alpha-6-SNAPSHOT: checking for updates from
[INFO] snapshot org.apache.maven.archetype:maven-archetype:2.0-alpha-6-SNAPSHOT: checking for updates from centr
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [archetype:generate] (aggregator-style)
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Preparing archetype:generate
[INFO] No goals needed for project - skipping
[INFO] snapshot org.apache.maven.archetype:archetype-common:2.0-alpha-6-SNAPSHOT: checking for updates from cent
[INFO] snapshot org.apache.maven.archetype:archetype-common:2.0-alpha-6-SNAPSHOT: checking for updates from apac
[INFO] [archetype:generate {execution: default-cli}]
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.archetypes:maven-archetype-quick
Choose archetype:
1: remote -> docbkx-quickstart-archetype (null)
2: remote -> gquery-archetype (null)
3: remote -> gquery-plugin-archetype (null)
//....

285: remote -> wicket-scala-archetype (Basic setup for a project that combines Scala and Wicket,
                depending on the Wicket-Scala project. Includes an example Specs
                test.)
286: remote -> circumflex-archetype (null)
Choose a number: 76: 76
Choose version:
1: 1.0
2: 1.0-alpha-1
3: 1.0-alpha-2
4: 1.0-alpha-3
5: 1.0-alpha-4
6: 1.1
Choose a number: : 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 following. We'll work with Social latest release: 1.1.0-CR01.

```
<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-CR01</version>
  </parent>

  <groupId>org.exoplatform.social.samples</groupId>
  <artifactId>exo.social.samples.activity-plugin</artifactId>
  <packaging>jar</packaging>
  <version>1.1.0-CR01</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 custom ui component for displaying its activity, we need a class that must be 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;

/**
 * Created by The eXo Platform SAS
 * Author : eXoPlatform
 *          exo@exoplatform.com
 * Aug 23, 2010
 */
@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;

/**
 * Created by The eXo Platform SAS
 * Author : eXoPlatform
 *          exo@exoplatform.com
 * Aug 19, 2010
 */
public class SimpleSpaceUIActivityBuilder extends BaseUIActivityBuilder {

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

  }

}
```

We're done. Now 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`, its value have to match the activity's type you want to display with your ui component.

Assume that we have already built Social project with version 1.1.0-CR01 or above. If you don't know how to, have a look at Building from Sources with Social 1.1.0-CR01. Build sample project and copy jar file 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, feel and function like default one.

When displaying an activity, we should make sure the look and feel of custom ui component is consistent and matched other activities and have the functions of like, comments, too. So we're going to create another ui component to display, we call it: `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;

/**
 * Created by The eXo Platform SAS
 * Author : eXoPlatform
 *          exo@exoplatform.com
 * Aug 23, 2010
 */
@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 `UISpaceLookAndFeelActivity` template. The easiest way is to copy the content of `UIDefaultActivity.gtmpl` to this template file:

```
<%
  import org.exoplatform.portal.webui.util.Util;
  import org.exoplatform.webui.form.UIFormTextAreaInput;

  def pcontext = Util.getPortalRequestContext();
  def jsManager = pcontext.getJavascriptManager();
  def labelActivityHasBeenDeleted = _ctx.appRes("UIActivity.label.Activity_Has_Been_Deleted");
  def activity = uicomponent.getActivity();
  def activityDeletable = uicomponent.isActivityDeletable();
%>

<% if (activity) { //process if not null

  jsManager.importJavascript("eXo.social.Util", "/social-resources/javascript");
  jsManager.importJavascript("eXo.social.PortalHttpRequest", "/social-resources/javascript");
  jsManager.importJavascript("eXo.social.webui.UIForm", "/social-resources/javascript");
  jsManager.i!mportJavascript("eXo.social.webui.UIActivity", "/social-resources/javascript");

  def labelComment = _ctx.appRes("UIActivity.label.Comment");
  def labelLike = _ctx.appRes("UIActivity.label.Like");
  def labelUnlike = _ctx.appRes("UIActivity.label.Unlike");
  def labelSource = _ctx.appRes("UIActivity.label.Source");
  def inputWriteAComment = _ctx.appRes("UIActivity.input.Write_A_Comment");
  def labelShowAllComments = _ctx.appRes("UIActivity.label.Show_All_Comments");
  def labelHideAllComments = _ctx.appRes("UIActivity.label.Hide_All_Comments");
  def labelOnePersonLikeThis = _ctx.appRes("UIActivity.label.One_Person_Like_This");
  def labelPeopleLikeThis = _ctx.appRes("UIActivity.label.People_Like_This");
  def labelYouLikeThis = _ctx.appRes("UIActivity.label.You_Like_This");
  def labelYouAndOnePersonLikeThis = _ctx.appRes("UIActivity.label.You_And_One_Person_Like_This");
  def labelYouAndPeopleLikeThis = _ctx.appRes("UIActivity.label.You_And_People_Like_This");

  def likeActivityAction = uicomponent.event("LikeActivity", "true");
  def unlikeActivityAction = uicomponent.event("LikeActivity", "false");

  def commentList = uicomponent.getComments();
  def allComments = uicomponent.getAllComments();
  if (allComments) {
    labelShowAllComments = labelShowAllComments.replace("{0}", allComments.size() + "");
    labelHideAllComments = labelHideAllComments.replace("{0}", allComments.size() + "");

  }
  def displayedIdentityLikes = uicomponent.getDisplayedIdentityLikes();
  def identityLikesNum = 0;
  def labelLikes = null;
  def toggleDisplayLikesAction = uicomponent.event("ToggleDisplayLikes");
  def startTag = "<a onclick="$toggleDisplayLikesAction" id="ToggleDisplayListPeopleLikes${activity.id}" href="#
```

```
  def endTag = "</a>";
  if (displayedIdentityLikes != null) {
    identityLikesNum = displayedIdentityLikes.length;
  }
  def commentListStatus = uicomponent.getCommentListStatus();
  def commentFormDisplayed = uicomponent.isCommentFormDisplayed();
  def likesDisplayed = uicomponent.isLikesDisplayed();
  //params for init UIActivity javascript object
  def params = """
    {activityId: '${activity.id}',
     inputWriteAComment: '$inputWriteAComment',
     commentMinCharactersAllowed: ${uicomponent.getCommentMinCharactersAllowed()},
     commentMaxCharactersAllowed: ${uicomponent.getCommentMaxCharactersAllowed()},
     commentFormDisplayed: $commentFormDisplayed,
     commentFormFocused: ${uicomponent.isCommentFormFocused()}
    }
  """
  jsManager.addOnLoadJavascript("initUIActivity${activity.id}");
  //make sures commentFormFocused is set to false to prevent any refresh to focus, only focus after post a comme
  uicomponent.setCommentFormFocused(false);
  def activityUserName, activityUserProfileUri, activityImageSource, activityContentBody, activityPostedTime;
  def commentFormBlockClass = "", listPeopleLikeBlockClass = "", listPeopleBGClass = "";
  if (!commentFormDisplayed) {
    commentFormBlockClass = "DisplayNone";
  }

  if (!likesDisplayed) {
    listPeopleLikeBlockClass = "DisplayNone";
  }

  if (uicomponent.isLiked()) {
    if (identityLikesNum > 1) {
      labelLikes = labelYouAndPeopleLikeThis.replace("{start}", startTag).replace("{end}", endTag).replace("{0}"
    } else if (identityLikesNum == 1) {
      labelLikes = labelYouAndOnePersonLikeThis.replace("{start}", startTag).replace("{end}", endTag);
    } else {
      labelLikes = labelYouLikeThis;
    }
  } else {
    if (identityLikesNum > 1) {
        labelLikes = labelPeopleLikeThis.replace("{start}", startTag).replace("{end}", endTag).replace("{0}", id
    } else if (identityLikesNum == 1) {
        labelLikes = labelOnePersonLikeThis.replace("{start}", startTag).replace("{end}", endTag);
    }
  }

  if (!labelLikes) {
   //hides diplayPeopleBG
   listPeopleBGClass = "DisplayNone";
  }

  activityContentTitle = activity.title;
  activityPostedTime = uicomponent.getPostedTimeString(activity.postedTime);
  activityUserName = uicomponent.getUserFullName(activity.userId);
  activityUserProfileUri = uicomponent.getUserProfileUri(activity.userId);

  activityImageSource = uicomponent.getUserAvatarImageSource(activity.userId);
  if (!activityImageSource)  {
    activityImageSource = "/social-resources/skin/ShareImages/SpaceImages/SpaceLogoDefault_61x61.gif";
  }

%>

<div class="UIActivity">
  <script type="text/javascript">
    function initUIActivity${activity.id}() {
      new eXo.social.webui.UIActivity($params);
    }
  </script>

  <% uiform.begin() %>
  <div class="NormalBox clearfix">
    <a class="Avatar" title="$activityUserName" href="$activityUserProfileUri">
      <img title="$activityUserName" src="$activityImageSource" alt="$activityUserName" height="54px" width="56p
    </a>
    <div class="ContentBox" id="ContextBox${activity.id}">
      <div class="TitleContent clearfix">
        <div class="Text">
```

```
        <a title="$activityUserName" href="$activityUserProfileUri">$activityUserName</a>
      </div>
    <% if (activityDeletable) {%>
      <div onclick="<%= uicomponent.event("DeleteActivity", uicomponent.getId(), ""); %>" class="CloseContentB
    <%}%>
    </div>
    <div class="Content">
      $activityContentTitle (from custom ui component)<br>
    </div>
    <div class="LinkCM">
      <span class="DateTime">$activityPostedTime *</span>
    <% def toggleDisplayCommentAction = uicomponent.event('ToggleDisplayCommentForm', null, false);
       def commentLink = "";
    %>
      <a class="LinkCM $commentLink" onclick="$toggleDisplayCommentAction" id="CommentLink${activity.id}" href
        $labelComment
      </a> |
    <% if (uicomponent.isLiked()) { %>
      <a onclick="$unlikeActivityAction" class="LinkCM" id="UnLikeLink${activity.id}" href="#unlike">
        $labelUnlike
      </a>
    <% } else { %>
      <a onclick="$likeActivityAction" class="LinkCM" id="LikeLink${activity.id}" href="#like">
        $labelLike
      </a>
    <% }%>
    </div>
  </div>

  <div class="ListPeopleLikeBG $listPeopleBGClass">
    <div class="ListPeopleLike">
      <div class="ListPeopleContent">
      <% if (!labelLikes) labelLikes = ""; %>
        <div class="Title">$labelLikes</div>
        <div class="$listPeopleLikeBlockClass">
        <%
        //def personLikeFullName, personLikeProfileUri, personLikeAvatarImageSource;

        displayedIdentityLikes.each({
          personLikeFullName = uicomponent.getUserFullName(it);
          personLikeProfileUri = uicomponent.getUserProfileUri(it);
          personLikeAvatarImageSource = uicomponent.getUserAvatarImageSource(it);
          if (!personLikeAvatarImageSource) {
            personLikeAvatarImageSource = "/social-resources/skin/ShareImages/activity/AvatarPeople.gif";
          }
          %>
            <a class="AvatarPeopleBG" title="$personLikeFullName" href="$personLikeProfileUri">
              <img width="47px" height="47px" src="$personLikeAvatarImageSource" alt="$personLikeFullName" tit
            </a>
         <% })%>
        </div>
        <div class="ClearLeft">
          <span></span>
        </div>
      </div>
    </div>
  </div>

  <div class="CommentListInfo">
  <% if (uicomponent.commentListToggleable()) {
  def showAllCommentsAction = uicomponent.event("SetCommentListStatus", "all");
  def hideAllCommentsAction = uicomponent.event("SetCommentListStatus", "none");
  %>
    <div class="CommentBlock">
      <div class="CommentContent">
        <div class="CommentBorder">
        <% if (commentListStatus.getStatus().equals("latest") || commentListStatus.getStatus().equals("none"))
          <a onclick="$showAllCommentsAction" href="#show-all-comments">
            $labelShowAllComments
          </a>
        <% } else if (commentListStatus.getStatus().equals("all")) { %>
          <a onclick="$hideAllCommentsAction" href="#hide-all-comments">
            $labelHideAllComments
          </a>
        <% } %>
        </div>
      </div>
    </div>
```

```
    <% } %>
    </div>
  <% if (allComments.size() > 0) { %>
    <div class="DownIconCM"><span></span></div>
  <% }%>

  <%
  def commenterFullName, commenterProfileUri, commenterImageSource, commentMessage, commentPostedTime;
  def first = true, commentContentClass;
  commentList.each({
    if (first & !uicomponent.commentListToggleable()) {
      commentContentClass = "CommentContent";
      first = false;
    } else {
      commentContentClass = "";
    }
    commenterFullName = uicomponent.getUserFullName(it.userId);
    commenterProfileUri = uicomponent.getUserProfileUri(it.userId);
    commenterImageSource = uicomponent.getUserAvatarImageSource(it.userId);
    if (!commenterImageSource) {
      commenterImageSource = "/social-resources/skin/ShareImages/activity/AvatarPeople.gif";
    }
    commentMessage = it.title;
    commentPostedTime = uicomponent.getPostedTimeString(it.postedTime);
  %>
    <div id="CommentBlock${activity.id}" class="CommentBox clearfix">
      <a class="AvatarCM" title="$commenterFullName" href="$commenterProfileUri">
        <img src="$commenterImageSource" alt="$commenterFullName" height="36px" width="38px">
      </a>
      <div class="ContentBox">
        <div class="Content">
          <a href="$commenterProfileUri"><span class="Commenter">$commenterFullName<span></a><br />
            $commentMessage
          <br/>
        </div>
        <div class="LinkCM">
          <span class="DateTime">$commentPostedTime</span>
        </div>
      </div>
    <%
      if (uicomponent.isCommentDeletable(it.userId)) {
    %>
      <div onclick="<%= uicomponent.event("DeleteComment", uicomponent.id, it.id); %>" class="CloseCMContentHili
    <% } %>
    </div>
  <% }) %>

    <div class="CommentBox $commentFormBlockClass clearfix" id="CommentFormBlock${activity.id}">
      <% uicomponent.renderChild(UIFormTextAreaInput.class); %>
      <input type="button" onclick="<%= uicomponent.event("PostComment") %>" value="$labelComment" class="Commer
    </div>

  </div>
  <% uiform.end() %>
</div>
<% } else { %> <!-- activity deleted -->
<div class="UIActivity Deleted">$labelActivityHasBeenDeleted</div>
<% }%>
```

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

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

That's all. 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. Re-run server and see the result:

> **Note**
>
> Currently, we have to copy and paste in template file. This way we have full control of the ui but it's not good when there's changes in UIDefaultActivity. We'll improve this soon so that no copy and pasted needed.

4.4.2.1.1. What is activity builder?

4.4.2.2. Create a composer extension for composing activity on the ui composer and display it on activity stream.

4.4.3. Summary

You can see more code for Social built-in plugin (Link Composer Plugin and Document Composer Plugin) in our project code

4.5. Developing Open Social Gadgets

4.5.1. Developing OpenSocial Gadgets

4.5.2. Introduction

eXo Social with OpenSocial container based on apache shindig supports OpenSocial capability. So we can develop gadgets that use OpenSocial API to request OpenSocial data or deploy opensocial gadget using tool supported by portal.

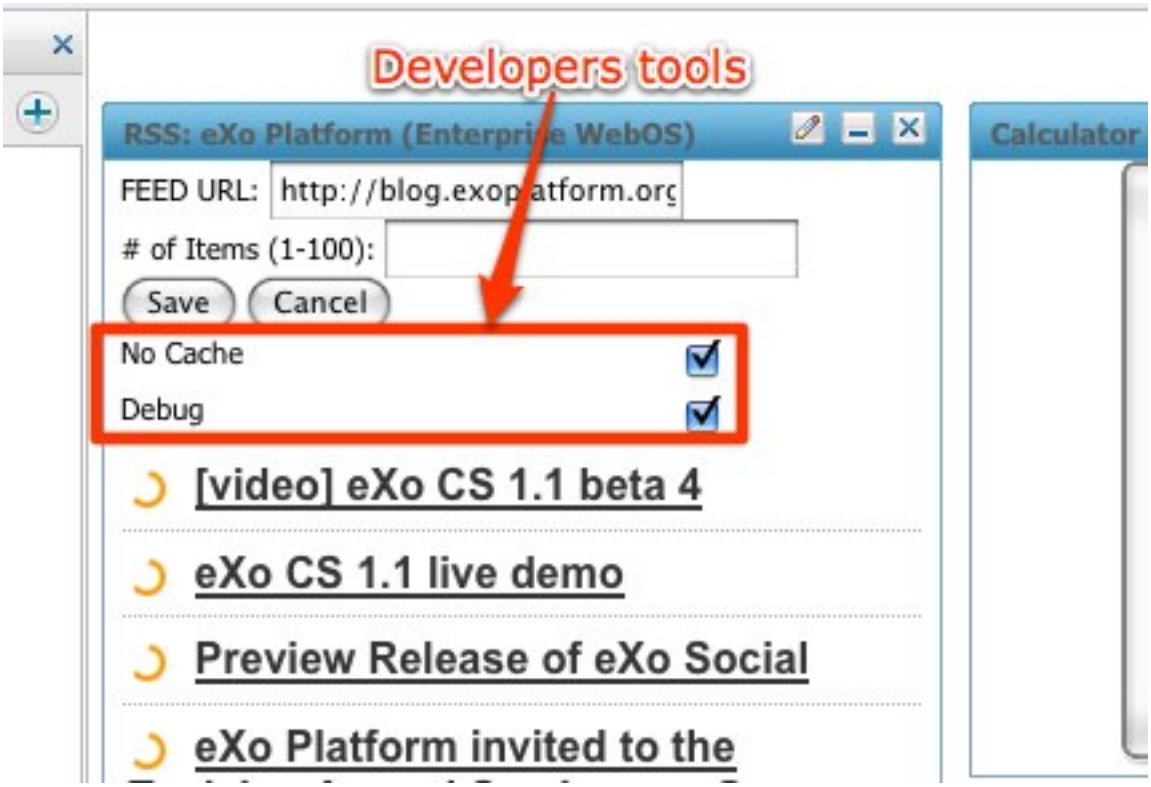
To develop your gadgets, you can use the tool that supported by portal.

4.5.3. Development Parameters

To be able to set this parameters, you have to be in the group *"/platform/administrators"*. So by default, only the root user has access to this functionality. If you want to change this group, you need to change the

configuration of the service *GadgetRegistryService*.

- Set *debug* to **true** to not compress the javascript inside the gadget.
- Set *nocache* to **true** to not cache the rendering of your gadget.



Note

The settings are saved **just after clicking** on the checkbox. But you need to **refresh the page** to see the parameters activated.

4.5.4. Using Webdav

Note

TODO, but you can look at this demonstration

4.6. Open Social

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.

4.6.1. 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.

4.6.1.1. 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

4.6.2. 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

4.6.2.1. 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.

4.6.2.2. 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.

4.6.2.3. Tutorial

- Grails + eXo Social tutorial