exo.social.docs.refguide - 1.1.0-CR01

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. Configuration

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

1.1. Spaces Template Configuration

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-type>
  <owner-id></owner-id>
  <name></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-type>
  <owner-id></owner-id>
  <name></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

- TODO User*Profile*Fields

Chapter 2. Technical*Foundations*

2.1. Informations

Table 2.1.

Extension Name	Status	comments
analytics	De-activated	
auth-refresh	Not Tested	
caja	Not Tested	
core	Not Tested	
core.io	Not Tested	
dynamic-height	Tested	
echo	Not Tested	
flash	Not Tested	
locked-domain	Not Tested	
minimessage	Not Tested	
opensocial-0.6	Not Tested	
opensocial-0.7	Tested	
opensocial-0.8	Tested	
opensocial-templates	Not Tested	
pubsub	Not working	
rpc	Not Tested	
setprefs	Tested	
settitle	Tested	
skins	Not Tested	
tabs	Tested	
target	Not Tested	
views	Not Tested	

2.1.1. Developing OpenSocial Gadgets

2.1.2. Introduction

eXo Social with OpenSocial container based on apache shindig supports OpenSocial capability. So we can

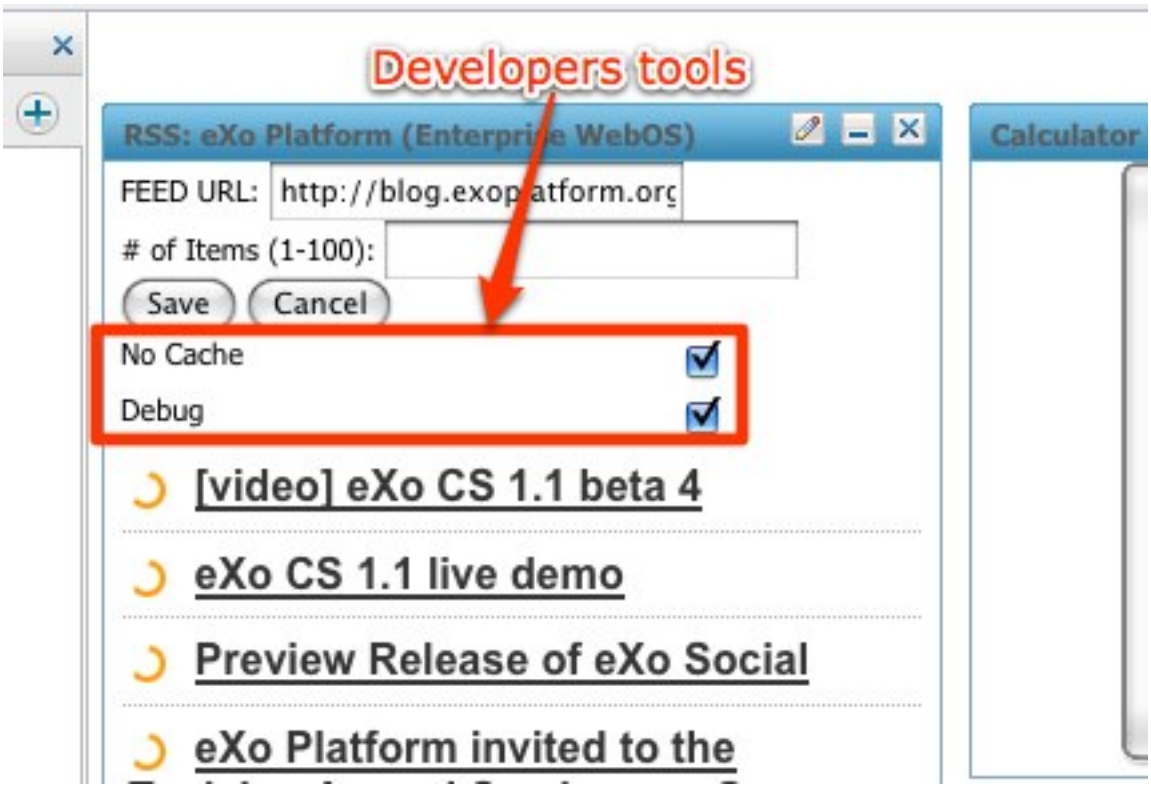
develop gadgets that use OpenSocial API to request OpenSocial data or deploy opensocial gadget using tool supported by portal.

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

2.1.3. 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.

2.1.4. Using Webdav

Note

TODO, but you can look at this [demonstration|http://www.vimeo.com/2069512]

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

2.1.4.1. Shindig configuration

2.1.4.1.1. 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
```

2.1.4.1.2. 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",
```

2.1.4.2. Adding fields to the Person Object

2.1.4.2.1. Modifications in the Java

If you need to add more information to the profile of a Person, you can extend ExoPersonImpl to add your fields. Your new fields will be automatically converted thanks to the POJO -> JSON or XML convertion.

The data of a person is pushed in the class ExoPeopleService. If you want to add some data to the ExoPersonImpl, you need to modify/extend this class.

> **Note**
>
> if you extend [ExoPersonImpl|http://svn.exoplatform.org/svnroot/exoplatform/projects/social/trunk/component/opensocial/sr
> be careful, in [ExoPeopleService|http://svn.exoplatform.org/svnroot/exoplatform/projects/social/trunk/component/opensocial
> the object is created. We should use Guice to create this object (see the task [SOC-71|http://jira.exoplatform.org/browse/SOC-71] about this issue)

After doing so, you should specify to eXo to use your implementation by extending SocialApiGuiceModule as it is done in ExoSocialApiGuiceModule to specify what of your class to use.

2.1.4.2.2. Modifications in the Javascript API

> **Note**
>
> TODO

2.2. Resources

- Read the API Documentation
- Checkout the Developer's guide
- Look at the articles on opensocial
- Watch the video explaining how to deploy the gadgets and the presentation in Bangkok
- Look at a sample of integration between eXo and appengine (Python backend)
- Checkout the existing opensocial app developped for eXo
- Have a look at the project opensocial-resources on google code

2.3. Space Portlet Preferences

2.3.1. Save and get space context from a portlet

> **Note**
>
> Since eXo Social 1.0.1

2.3.2. Why save portlet preference

When working with space related portlet, we have make sure which space (space context) that portlet is working with to handle and display space's information. For example, `SpaceActivityStreamPortlet` has to know space context to display space's activity stream. Before (as of 1.0.0-GA), we rely on actual http address to gets space's url and then gets the current space, and that causes a bug here when edit that page node SOC-715.

2.3.3. How to save space context

If you want to register a portlet to be installed when creating a new space, write a portlet and set configuration in <tt>configuration.xml<tt> file of exo.social.component.space project. Just register your portlet here and it will be dynamically created a page with your portlet.

```
<component>
  <key>org.exoplatform.social.space.SpaceService</key>
  <type>org.exoplatform.social.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>UserListPortlet:true</value>
      <value>YourPortletName:true</value>
    </values-param>
```

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

Notice your portlet line configuration:

```
<value>YourPortletName:true</value>
```

`true` or `false` means that portlet can be uninstalled or mandatory from application settings for space.

If your portlet wants to save space context, register your portlet name in SpaceUtils (it should be configured via xml soon):

```
//SpaceUtils
static public String[] PORTLETS_SPACE_URL_PREFERENCE_NEEDED = {"SpaceActivityStreamPortlet", "SpaceSettingPortle
```

That's it. Whenever creating a page containing your portlet, spaceUrl will be save as portlet preference.

1 How to get space context via portlet preference

Actually, we save a space's url to a portlet preference and by getting that space's url by using <tt>SpaceUtils</tt>:

```
//SpaceUtils
/**
 * Gets spaceName by portletPreference
 * @return space's url
 */
static public String getSpaceUrl() {
  PortletRequestContext pcontext = (PortletRequestContext)WebuiRequestContext.getCurrentInstance();
  PortletPreferences pref = pcontext.getRequest().getPreferences();
  return pref.getValue(SPACE_URL, "");
}
```

and then gets `Space` instance by `SpaceService`.

```
//SpaceService
/**
 * Get a space by its url
 * @param spaceUrl url of space
 * @return Space space with string url specified
 * @throws SpaceException
 */
Space getSpaceByUrl(String spaceUrl) throws SpaceException;
```

Your portlet:

```
//YourPortletName.java
/**
 * constructor
 */
public YourPortletName() throws Exception {
  Space space = getSpaceService().getSpaceByUrl(SpaceUtils.getSpaceUrl());
}


public SpaceService getSpaceService() {
```

```
    return getApplicationComponent(SpaceService.class);
  }
```

Chapter 3. Development

3.1. Activity streams

Note

Some changes for Social 1.0.1{/info}

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|http*repository:exoplatform.org-service-local-repositories-public-archive-org-exoplatform-so*
To get an instance of this class, you will need to use the `PortalContainer`.

h2. 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.

h2. Publish an activity for a user

{code:java} 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.class);

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(); } } {/code}

h2. Publish an activity for a space

{code:java} 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.class);

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(); } } {/code}

h2. Useful functions

- [ActivityManager#getActivity|http*repository:exoplatform.org-service-local-repositories-public-archive-org-*
- [ActivityManager#getActivities|http*repository:exoplatform.org-service-local-repositories-public-archive-or*
- [ActivityManager#saveActivity|http*repository:exoplatform.org-service-local-repositories-public-archive-or*
- [ActivityManager#saveComment|http*repository:exoplatform.org-service-local-repositories-public-archive-o*
- [ActivityManager#saveLike|http*repository:exoplatform.org-service-local-repositories-public-archive-org-ex*
- [ActivityManager#removeLike|http*repository:exoplatform.org-service-local-repositories-public-archive-org*

- [ActivityManager#getComments|http*repository:exoplatform.org-service-local-repositories-public-archive-o*

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`

{code language="java"} package org.exoplatform.social.core.activitystream;

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

String smiley = "";

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

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

{code:language="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> {/code}

"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.

h2. 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 specific portal container: `socialdemo` as in the configuration file:

{code:language="xml"} <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></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></description> <init-params> <properties-param> <name>mash.info</name> <property name="feedURL" value="http://builder.exoplatform.org/hudson/view/social/job/social-trunk-ci/rssAll"/> <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> {/code}

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.

3.2. 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.

3.2.1. 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.

3.2.1.1. 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

3.2.2. 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

3.2.2.1. 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.

3.2.2.2. 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.

3.2.2.3. Tutorial

- Grails + eXo Social tutorial

3.3. 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.

3.3.1. Identity

3.3.2. 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.

3.3.3. 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);
```

3.3.4. Useful methods

- IdentityManager#getIdentities]
- IdentityManager#getIdentitiesByProfileFilter]
- IdentityManager#getIdentity]
- IdentityManager#getOrCreateIdentity]
- IdentityManager#saveIdentity]
- IdentityManager#addIdentityProvider]

3.3.5. 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.

3.3.6. = Useful classes =

- ProfileListenerPlugin]
- ProfileLifeCycleEvent]

= 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.

3.3.7. 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);
}
```

3.3.8. = Useful methods =

- RelationshipManager#confirm]
- RelationshipManager#deny]
- RelationshipManager#ignore]
- RelationshipManager#getPending]
- RelationshipManager#getContacts]
- RelationshipManager#getRelationshipStatus]

3.3.9. 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>
```

3.3.10. = Useful Classes =

- RelationshipListenerPlugin]
- RelationshipEvent]

> **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

> **Note**
>
> Some changes updated for eXo Social 1.0.1

3.4. Additional Tutorials

3.4.1. Developing OpenSocial Gadgets

3.4.2. Introduction

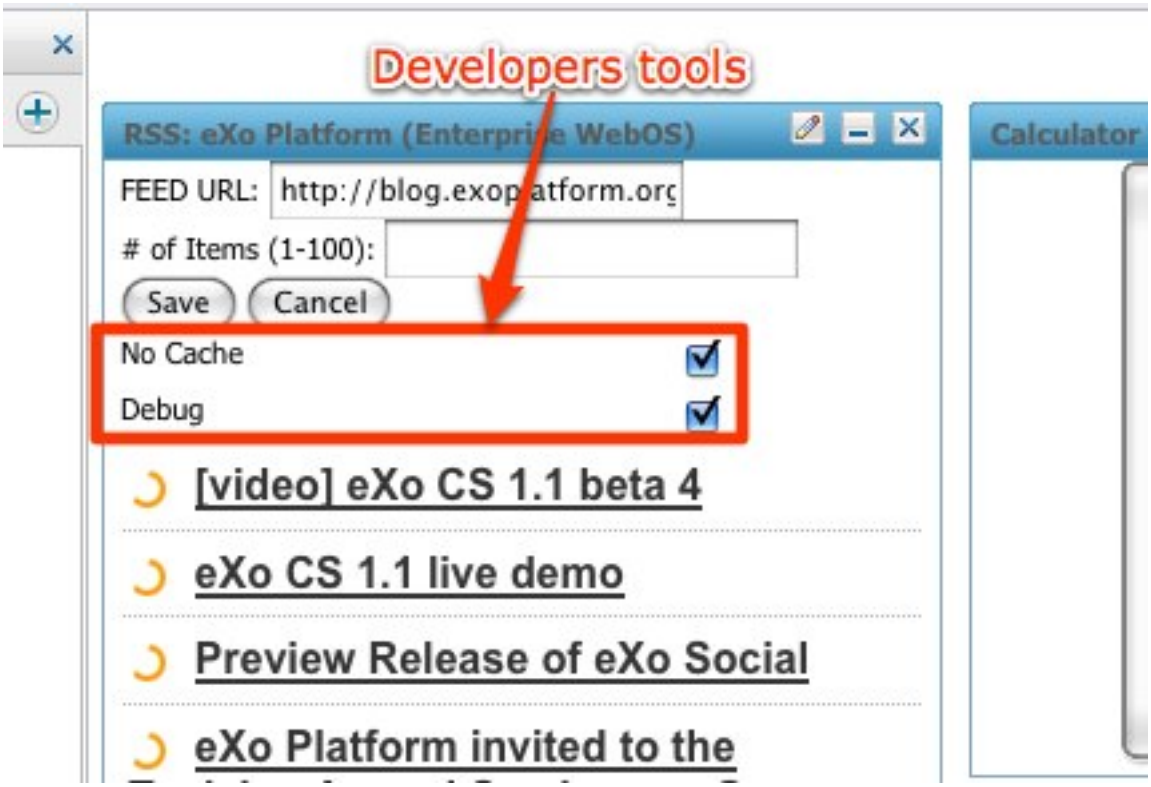
eXo Social with OpenSocial container based on apache shindig supports OpenSocial capability. So we can develop gadgets that use OpenSocial API to request OpenSocial data or deploy opensocial gadget using tool supported by portal.

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

3.4.3. 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.

3.4.4. Using Webdav

Note

TODO, but you can look at this [demonstration|http://www.vimeo.com/2069512]

3.4.5. How to extend the activities rendering

3.4.6. Objective

In this article you will learn how you can implement a preprocessor for activities rendering.

3.4.7. Requirements

You need to be familiar with the eXo Kernel component model and its xml configuration. You also need be able to program in java.

3.4.8. Why would you need to do this ?

An simplistic activity is made of simple text. We have created an extension point at the level of activities renderting for two cases :

- support more html tags
- support @mentions

But you may want to support a special syntax like :

- #hashtags to feel like twitter
- smileys to look like skype
- Markdown to be as cool as Buzz ;-)

You can imagine more sophisticated processing. And not only text-related because a processor actually has full access to the Activity. So you can very well process based on the owner, the app, mediaitem, etc..

3.4.9. Writing an ActivityProcessor

eXo Social let's you pre-process an activity before it is returned by the ActivityManager. To do this you simply need to implement the interface ActivityProcessor :

```
/**
 * An activity processor is responsible to pre-process an activity before it is returned by the {@link ActivityM
 */
public interface ActivityProcessor {

  /**
   * Process an activity
   * @param activity the activity. It can be modified
   */
  void processActivity(Activity activity);

  /**
   * Priority of this processor.
   * All activity processors will be executed in ascending priority order
   * @return
   */
  int getPriority();

}
```

For example let's implement a SmileyProcessor that will replace text smileys by icons :

```
public class SmileyProcessor implements ActivityProcessor {

  String smiley =  "<img src='/images/smiley.gif'/>";

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

  public int getPriority() {
    return 100;
  }

}
```

3.4.10. Configuring the processor

Now that you have a nice procesor you need to hook it to the system. At runtime, processors can be attached to

the ActivityManager via the method ActivityManageraddProcessor(ActivityProcessor processor).

But there is also a component plugin hook for it : public void addProcessorPlugin(BaseActivityProcessorPlugin plugin)

So to make your processor easy to hook, you simply need to let him extend the BaseActivityProcessorPlugin.

```
public class SmileyProcessor extends BaseActivityProcessorPlugin {

  String smiley =  "<img src='/images/smiley.gif'/>";

  public SmileyProcessor(InitParams params) {
    super(params);
  }

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

It will have the additional benefit to make the priority field configurable so you don't need to implement getPriorty().

Then your processor can be configured as a component plugi like this :

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

Restart, place the smiley gif on the server and you should see something like that :

*External**resources*** **OfficialOpenSocialDeveloper'sGuide** **onGoogleCode** How*to*deploy*a*remote*OpenSocial*Google*Gadget*made*with*GWT*in*eXo *by* Laurent*Bois*

- Also check this list of useful RSS feeds to follow.

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

Chapter 4. Support

4.1. eXo Social Frequently Asked Questions

4.2. How to know the version of eXo Social I am using?

Go to your $TOMCAT*HOME/lib and search for the library starting by "exo.social", the version number is at the end of the file name. For example: "exo.social.component.opensocial-1.0-alpha1.jar", the version number is* ***"1.0-alpha1"***.

If the version number ends by ~~SNAPSHOT, it means that this version has been built against a development version. To know what svn version corresponds to this jar:~~

- Unzip the jar
- Open the file **"META-INF/MANIFEST.MF"**
- In this file you will see the svn version, it's the line starting by **SCM-Revision** (ex: *SCM-Revision: 25023*)

4.3. What to do if I'm behind a firewall?

Some OpenSocial gadgets needs to have access to internet. You can copy this gadgets to your local repository, setup your proxy or simply not use it. No OpenSocial gadget is required to have eXo Social working. To setup your proxy you should setup the proxy of your appserver :

```
-Dhttp.proxyHost=yourproxy.your.domain -Dhttp.proxyPort=80
```