	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

Gestion d'identité et politique de confidentialité, Gestion des projets et notion de groupe

Livrable du au titre du projet	COCLICO
Lot	1
Tache	0.3
Livrable	1.3.1


Rédacteur(s)	Vérificateur(s)	Approbateur(s)
R Mas , C Bayle		



Titre du document : Compte rendu d'étude
Référence : L 1.3.1
Version du 24/09/2011

Table of Contents

I.Overview.....	3
II.Interfaces.....	4
A.PFO_RBACEngine.....	4
B.PFO_Role.....	4
C.PFO_RoleExplicit (extends PFO_Role).....	5
D.PFO_RoleUnion (extends PFO_Role).....	5
E.PFO_RoleAnonymous (extends PFO_Role).....	5
F.PFO_RoleLoggedIn (extends PFO_Role).....	5
G.Other possible classes.....	5
III.Permissions.....	7
A.Global permissions.....	7
B.Project-related permissions.....	7
C.Tool-related permissions.....	8
IV.PHP definition.....	10
A.PFO-RBAC.interface.php.....	10
B.Possible usage.....	11

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

I. Overview

Following the discussion started at <http://lists.planetforge.org/pipermail/discussions/2010-April/000055.html>, here's a proposed specification document for a common API for RBAC across forges.

We want to define a clean model for permission handling, with a common API. This comes from the quirks in FusionForge's RBAC (such as the RoleObserver and the groups whose function is only to track permission bits) and those in Codendi's "Ugroups" system (such as the permission matrix that still exists in projects). These two models aren't too different, so the convergence isn't too hard.


The permissions are managed by a role-based access control (RBAC) model. An individual user (or session) doesn't have explicit permissions; instead, the permissions are assigned to roles, and the user gets the permissions from the role (or roles) she has. There are several kinds of roles, with varying relationships to users, projects and permissions, but they behave similarly for permission checking, and their core purpose is to answer the question "is this role allowed to do action A on tool T?". Other methods are used for management, such as listing their members, updating their permissions set, and so on.

The roles can be classified according to whether their membership is explicit (a list of usernames) or dynamic (members are defined by a function and calculated at runtime). Explicit membership roles need extra methods to manage their member list. Implicit membership roles include the "logged-in" role, used when the client has opened a session, and the "anonymous" role, used otherwise. Another possibility is to define union roles, whose members are simply the set of their sub-roles (think about a {Developers} role defined as {Junior Developers} \cup {Senior Developers} for instance).

Another distinction for roles is that some are attached to a project in particular (the old-style roles in FusionForge, such as "Senior Developer"), while some others are global and spread the whole of the forge (such as the global "Registered users" Ugroup in Codendi). The roles attached to a "home project" can only be edited by the admins of that project; the roles with no home project can only be edited by the forge admins. Note that this only applies to editing the properties of the role (such as its name, and its list of members in the case of explicit membership roles). The permission set defined for a role in a target project are defined by the admins of that target project. To allow for a role from a project to be referenced in another project (home \neq target), roles have a public flag. A non-public role can only be referenced by its home project; a public role can be referenced by other projects, and used to define permissions for tools that are not related to any project in particular (such as the ability to approve new projects, for instance). The list of extra projects referenced by a public role is kept separate from the home project.

Once a role is referenced by an individual project (either because it's the home project for that role, or following an explicit action by the admin of that project), the project admin can grant it special permissions on some of the tools of that project.

It is also possible to define site-wide permissions on a whole *category* of tools for forge-wide roles; for instance, an account used for purposes of reporting might need to be granted read access to all trackers, without every project admin needing to grant these permissions by hand. Or, on a public forge, a "forum moderator" would need sufficient permissions on all the forums, without necessarily being a member of all projects.

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

II. Interfaces

A.PFO_RBACEngine

(The PFO_ prefix is for "planetforge.org")

This interface is meant to be implemented with a singleton pattern. Its methods use the session management to decide what roles are available within the current session (if any), and to provide the answer to the question “Does the current client have the permission for this action?”. Other interesting questions that this interface is meant to answer include “does *another* account have the permission for that action?” and, more generically, “who is allowed that action?”.

Methods:


- getInstance()
- getAvailableRoles()
- isActionAllowed(\$section,\$reference,\$action)
- isGlobalActionAllowed(\$section,\$action)
- isActionAllowedForUser(\$user,\$section,\$reference,\$action)
- isGlobalActionAllowedForUser(\$user,\$section,\$action)
- getRolesByAllowedAction(\$section,\$reference,\$action)
- getUsersByAllowedAction(\$section,\$reference,\$action)

B.PFO_Role

Abstract interface, not meant to be implemented directly.

Methods:

- get/setName()
- getID()
- getUsers() → array of users
- hasUser(\$user) → boolean
- hasPermission(\$section, \$reference, \$action): possibly the most important method. \$section is the kind of tool referenced ("tracker", "forum" and so on); \$reference is an identifier for the specific tool; \$action can vary depending on the tool (tech/manager for trackers, for instance)
- hasGlobalPermission(\$section, \$action): for forge-wide permissions not linked to a specific tool, such as project approval
- normalizeData() → ensure consistency, add missing values in the database if new plugins have appeared, remove values referring to obsolete tools, etc.
- getSettings() → get full list of permission settings
- getSettingsForProject(\$project) → get list of permission settings related to a project
- setSettings(\$data) → update list of permission settings; only the settings mentioned in \$data are changed
- isPublic() and setPublic(\$flag) → accessor for the “public” boolean flag

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

- `getHomeProject()` → NULL if role is “floating”
- `getLinkedProjects()` → possibly empty array including the home project if any
- `link/unlinkProject($project)` → reference/unreference a public role in an extra project

C.PFO_RoleExplicit (extends PFO_Role)

Standard, explicit membership role.

- if “public” is not set but “home project” is set: GForge-like project-local role;
- if both “public” and “home project” are set: Codendi-like shared role, can be referenced by other projects;
- if “public” is set but there's no “home project”: global role with an explicit membership;
- not public, and no home project: undefined for now; potential semantics (to be evaluated) could be a role that can only be used for forge-wide permissions, but not referenced in projects themselves. Is there a need for that?

Methods:

- `addUsers($users)`
- `removeUsers($users)`

D.PFO_RoleUnion (extends PFO_Role)

Union of roles. Beware of loops when defining union roles! Also beware of consistency for public/non-public roles.

Methods:

- `addRole($role)`
- `removeRole($role)`

E.PFO_RoleAnonymous (extends PFO_Role)


Global scope (public, no home project), *always* available (even when logged in). `hasUser()` *always* returns true.

F.PFO_RoleLoggedIn (extends PFO_Role)


Global scope (public, no home project), available whenever a valid session is opened. `hasUser()` *always* returns true.

G.Other possible classes

This document doesn't specify the following, but they are given as an example of other classes that

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

might be implemented. Extending the PFO_Role interface in a way similar to the RoleAnonymous and RoleLoggedIn, it is possible to define extra “implicit” roles for client sessions coming from the local network, or from a VPN, or from none of the above. The code determining whether a role is available or not would probably run checks on the client's IP address.

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

III. Permissions

The following permission bits are defined.

A. Global permissions


Section	Action	Comment
forge_admin	-	“God mode”: all actions allowed
approve_projects	-	Ability to approve pending projects
approve_news	-	Ability to approve news bits to the forge front page
forge_stats	read	Ability to read forge-wide statistics
	admin	Alter forge-wide statistics (rebuild them, for instance)

I would suggest (Christian) :

Resource	Capacity	Comment
forge	admin	“God mode”: all actions allowed
projects	approve	Ability to approve pending projects
news	approve	Ability to approve news bits to the forge front page
stats	read	Ability to read forge-wide statistics
	admin	Alter forge-wide statistics (rebuild them, for instance)

B. Project-related permissions

Section	Reference	Action	Comment
project_read	project_id	-	Ability to view project (replaces public/private projects)
project_admin		-	All permissions on the project
tracker_admin		-	All permissions on trackers of that project
pm_admin		-	All permissions on task managers of that project
forum_admin		-	All permissions on forums of that project
scm	project_id	read	Read access to the SCM repository
		write	Commit access
docman	project_id	read	Read access to the docs
		submit	Allowed to submit a new doc
		approve	Allowed to approve new submitted docs
		admin	Administrative access (manage folders)
frs	project_id	read_public	Read access to the public packages

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011


		read_private	Read access to all packages
		write	Allowed to publish files through the FRS

I would suggest (Christian) :


Resource	Reference	Capacity	Comment
project	project_id	read	Ability to view project (replaces public/private projects)
		admin	All permissions on the project
tracker	project_id	admin	All permissions on trackers of that project
pm	project_id	admin	All permissions on task managers of that project
forum	project_id	admin	All permissions on forums of that project
scm	project_id	read	Read access to the SCM repository
		write	Commit access
docman	project_id	read	Read access to the docs
		submit	Allowed to submit a new doc
		approve	Allowed to approve new submitted docs
		admin	Administrative access (manage folders)
frs	project_id	read_public	Read access to the public packages
		read_private	Read access to all packages
		write	Allowed to publish files through the FRS

C.Tool-related permissions

Section	Reference	Action	Comment
forum	forum_id	read	Ability to view forum and messages
		post	Allowed to post (with moderation)
		post_unmoderated	Allowed to post without moderation
		moderate	Allowed to moderate pending messages
new_forum	project_id	read/post/post_unmoderated/moderate	Default permissions for newly created forums
tracker	tracker_id	read	View tracker and artifacts in it
		tech	Can be assigned an artifact
		manager	Can assign artifacts to techs
new_tracker	project_id	read/tech/manager	Default settings for newly created trackers
pm	tracker_id	read	View task manager and tasks in it
		tech	Can be assigned a task
		manager	Can assign tasks to techs

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

new_pm	project_id	read/tech/manager	Default settings for newly created task managers
--------	------------	-------------------	--

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

IV.PHP definition

A.PFO-RBAC.interface.php

```

<?php
/**
 * API for role-based access control
 * Defined at Planetforge.org
 *
 * Copyright 2010, Roland Mas
 *
 * This file is part of FusionForge.
 *
 * FusionForge is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published
 * by the Free Software Foundation; either version 2 of the License,
 * or (at your option) any later version.
 *
 * FusionForge is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FusionForge; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
 * USA
 */

// Constants to identify role classes
define ("PFO_ROLE_EXPLICIT", 1) ;
define ("PFO_ROLE_ANONYMOUS", 2) ;
define ("PFO_ROLE_LOGGEDIN", 3) ;
define ("PFO_ROLE_UNION", 4) ;

// Interface for the RBAC engine
interface PFO_RBACEngine {
    public static function getInstance() ;
    public function getAvailableRoles() ; // From session
    public function isActionAllowed($section, $reference, $action = NULL) ;
    public function isGlobalActionAllowed($section, $action = NULL) ;
    public function isActionAllowedForUser($user, $section, $reference,
$action = NULL) ;
    public function isGlobalActionAllowedForUser($user, $section, $action =
NULL) ;
    public function getRolesByAllowedAction($section, $reference, $action =
NULL) ;
    public function getUsersByAllowedAction($section, $reference, $action =
NULL) ;
}

// Interfaces for the capabilities
interface PFO_Role {

```



```
public function getName() ;
public function setName($name) ;
public function getID() ;

public function isPublic() ;
public function setPublic($flag) ;
public function getHomeProject() ;
public function getLinkedProjects() ;
public function linkProject($project) ;
public function unlinkProject($project) ;

public function getUsers() ;
public function hasUser($user) ;
public function hasPermission($section, $reference, $action = NULL) ;
public function hasGlobalPermission($section, $action = NULL) ;
public function normalizeData() ;
public function getSettings() ;
public function getSettingsForProject($project) ;
public function setSettings($data) ;
}

interface PFO_RoleExplicit extends PFO_Role {
    const roleclass = PFO_ROLE_EXPLICIT ;
    public function addUsers($users) ;
    public function removeUsers($users) ;
}

interface PFO_RoleUnion extends PFO_Role {
    const roleclass = PFO_ROLE_UNION ;
    public function addRole($role) ;
    public function removeRole($role) ;
}

interface PFO_RoleAnonymous extends PFO_Role {
    const roleclass = PFO_ROLE_ANONYMOUS ;
}

interface PFO_RoleLoggedin extends PFO_Role {
    const roleclass = PFO_ROLE_LOGGEDIN ;
}

// Local Variables:
// mode: php
// c-file-style: "bsd"
// End:


?>
```

B.Possible usage

```
require "PFO-RBAC.interface.php" ;

// Code shared between classes

abstract class Error {}
```

	Titre du document : Compte rendu d'étude
	Référence : L 1.3.1
	Version du 24/09/2011

```
abstract class BaseRole extends Error implements PFO_Role {
    public function getName() {
        return $this->name ;
    }
    [...]
}

// Actual classes

class RoleExplicit extends BaseRole implements PFO_RoleExplicit {
    public function addUsers($users) {
        [...]
    }
    public function removeUsers($users) {
        [...]
    }
    public function getUsers() {
        [...]
    }
}

[...]
```