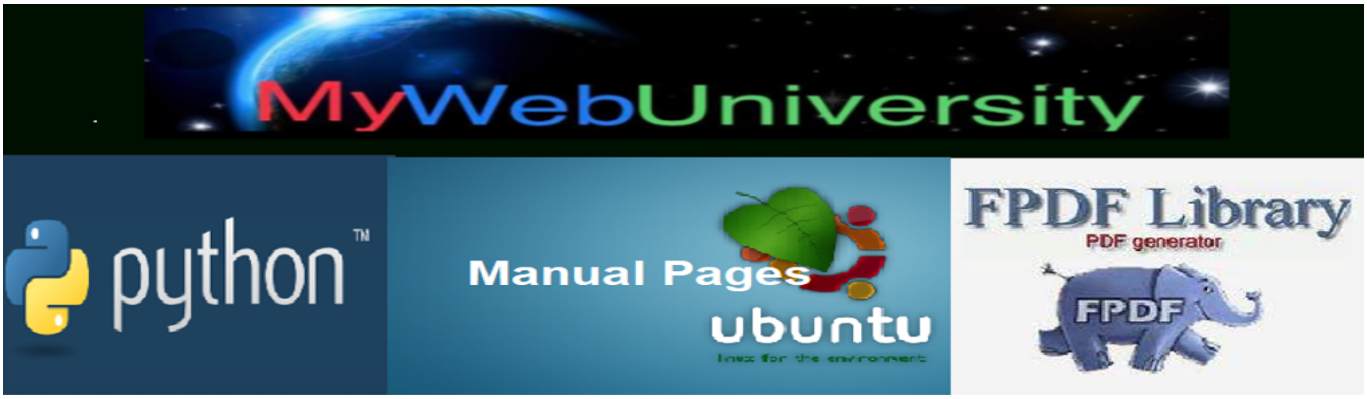

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::SiteControl::PermissionManager.3pm'

$ man Apache2::SiteControl::PermissionManager.3pm

Apache2::SiteControl::PermissioUseraContributed Perl Apache2::SiteControl::PermissionManager(3pm)

NAME

Apache2::SiteControl::PermissionManager - Rule-based permission management

SYNOPSIS

```
use Apache2::SiteControl::PermissionManager;
$manager = new Apache2::SiteControl::PermissionManager();
$rule1 = new SomeSubclassOfSiteControl();
$manager->addRule($rule1);
...
$user = new SomeUserTypeYouDefineThatMakesSenseToRules;
if($manager->can($user, $action, $resource)) {
  # OK to do action
}
# For example
if($manager->can($user, "read", "/etc/shadow")) {
  open DATA, "</etc/shadow";
  ...
}
```

DESCRIPTION

This module implements a user capabilities API. The basic idea is that you have a set of users and a set of things that can be done in a system. In the code of the system itself, you want to surround sensitive operations with code that determines if the current user is allowed to do that operation.

This module attempts to make such a system possible, and easily extensible. The module requires that you write implementations of rules for you system that are subclasses of Apache2::SiteControl::Rule. The rules can be written to use any data types, which are abstractly known as "users", "actions", and "resources."

A user is some object that your applications uses to identify the person operating the program. The expectation is that at some point the application authenticated the user and obtained their identity, and the rest of the application is merely applying a ruleset to determine what that user is allowed to do. In the context of the SiteControl system, this user is a Apache2::SiteControl::User or subclass thereof.

An action can be any data type (i.e. simply a string). Again, it is really up to the code of the rules (which are primarily written by you) to determine what is valid.

The overall usage of this package is as follows:

1. Decide how you want to represent a user. (i.e. Apache2::SiteControl::User)

2. Decide the critical sections of your code that need to be protected, and decide what to do if the user doesn't pass muster. For example if a screen should just hide fields, then the application code needs to reflect that.

3. Create a permission manager instance for your application. Typically use a singleton pattern (there need be only one manager). In the SiteControl system, this is done by a ManagerFactory that you write.

4. Surround sensitive sections of code with something like:

```
if($manager->can($user, "view salary", $payrollRecord))
{
  # show salary fields
} else
  # hide salary fields
}
```

5. Create rules that spell out the behavior you want and add them to your application's permission manager. The basic idea is that a rule can grant permission, or deny it. If it neither grants or denies, then the manager will take the safe route and say that the action cannot be taken. Part of the code for the rule for protecting salaries might look like:

```
package SalaryViewRule;
use Apache2::SiteControl::Rule;
```

```
use Apache2::SiteControl::User;
use base qw(Apache2::SiteControl::Rule);
sub grants
{
  $this = shift;
  $user = shift;
  $action = shift;
  $resource = shift;
  # Do not grant on requests we don't understand.
  return 0 if(!$user->isa("Apache2::SiteControl::User") ||
              !$this->isa("Apache2::SiteControl::Rule"));
  if($action eq "view salary" && $resource->isa("Payroll::Record")) {
    if($user->getUsername() eq $resource->getEmployeeName()) {
      return "user can view their own salary";
    }
  }
  return 0;
}
```

Then in your subclass of ManagerFactory:

```
use SalaryViewRule;
...
$viewRule = new SalaryViewRule;
$manager->addRule($viewRule);
```

METHODS

can(user, action verb, resource)

This is the primary method of the PermissionManager. It asks if the specified user can do the specified action on the specified resource. For example,

```
$manager->can($user, "eat", "cake");
```

would return true if the user is allowed to eat cake. Note that this gives you quite a bit of flexibility, but at the expense of strong type safety. It is suggested that all of your rules do type checking to insure that a rule is properly applied.

SEE ALSO

Apache2::SiteControl::Rule, Apache::SiteControl::ManagerFactory,

Apache2::SiteControl::UserFactory, Apache::SiteControl

AUTHOR

This module was written by Tony Kay, <tkay@uoregon.edu>.

COPYRIGHT AND LICENSE