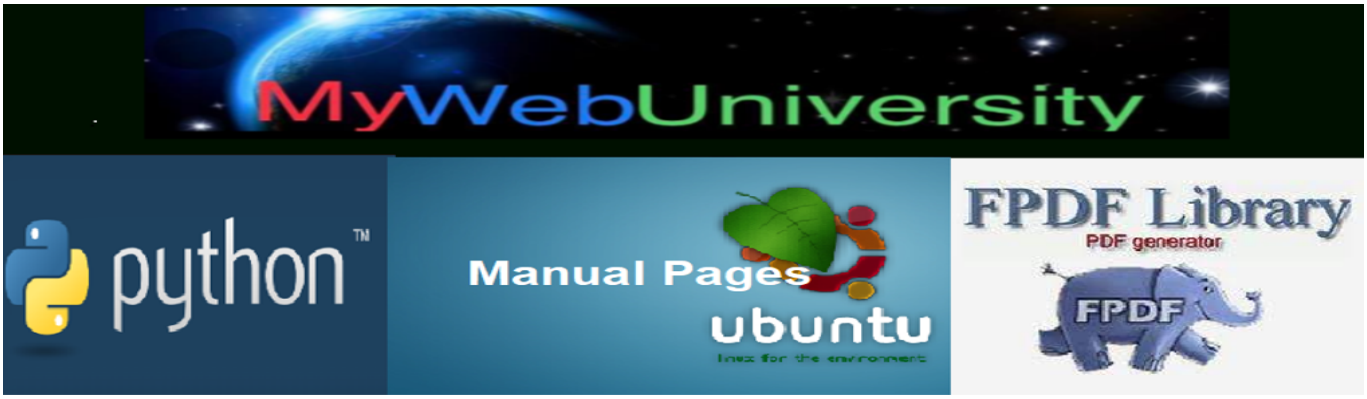

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::SiteControl.3pm'

$ man Apache2::SiteControl.3pm

Apache2::SiteControl(3pm) User Contributed Perl Documentation Apache2::SiteControl(3pm)

NAME

Apache2::SiteControl - Perl web site authentication/authorization system

SYNOPSIS

See samples/site for complete example. Note, this module is intended for mod_perl. See Apache2::SiteControl for mod_perl2.

DESCRIPTION

Apache2::SiteControl is a set of perl object-oriented classes that implement a fine-grained security control system for a web-based application. The intent is to provide a clear, easy-to-integrate system that does not require the policies to be written into your application components. It attempts to separate the concerns of how to show and manipulate data from the concerns of who is allowed to view and manipulate data and why.

For example, say your web application is written in HTML::Mason. Your individual "screens" are composed of Mason modules, and you would like to keep those as clean as possible, but decisions have to be made about what to allow as the component is processed. SiteControl attempts to make that as easy as possible.

DEVELOPER'S VIEWPOINT - EXAMPLE

In this document we use HTML::Mason to create examples of how to use the control mechanisms, but any mod_perl based system should be supportable.

A good mason component tries to do most of the perl processing in a separate block, so that simple substitutions can be made in HTML in the rest of the page. This makes it much easier for web developers and perl developers to co-exist on a project.

The SiteControl system tries to make it possible to continue to follow this model. You

obtain a user object and permission manager from the SiteControl system. These are intended to be opaque data types to the page designer, and are defined elsewhere (see USERS). The actual web page component should carry these objects around without implementing anything in the way of policy.

For example, your mason component might look like this:

```
<HTML>
  <HEAD> ... </HEAD>
% if($manager->can($currentUser, "edit", $table)) {
    <FORM METHOD=POST ACTION="...">
      <P><INPUT TYPE=TEXT NAME="x" VALUE="<% $table->{x} %>">
      ...
    </FORM>
% } else {
    <P>x is <% $table->{x} %>
% }
<%init>
my $currentUser = Apache2::SiteControl->getCurrentUser($r);
my $manager = Apache2::SiteControl->getPermissionManager($r);
... application specific stuff...
i.e.
my $table = ...
</%init>
```

Notice that the component does not bother looking at the user object, and there is no policy code...just a request for permission:

```
if($manager->can($currentUser, "do something to", $resource))
```

Of course the developer needs to know something about the underlying system. For example, the action string "do something to" is rather arbitrary. These can be anything, and must be specified as rule actions. It is recommended that you use some form of Perl constants for these instead of strings, but that is up to you.

The resource is intended to be less opaque. This is likely the object that the page developer wants to muck with, and so probably knows the internals of that object a bit better. This is the crossover point from what SiteControl can figure out on its own to information you have to supply.

The default behavior is for the manager to deny any request. In order for a request to be approved, someone has to write a rule that joins together the user, action, and resource and makes a decision about the permissibility of the action.

If all you want is login and user tracking (but no permission manager), then it is safe to ignore the permission manager altogether.

USERS

Users and Rules are the central components of the SiteControl system. The user object must be Apache2::SiteControl::User (or a subclass). See Apache2::SiteControl::User for a description of what it supports (session storage, logout, etc.). The glue to SiteControl is the UserFactory, which you can define or accept the default of Apache2::SiteControl::UserFactory (recommended).

Whenever a login attempt succeeds, the factory returns an object that represents a valid, logged-in user. See Apache2::SiteControl::UserFactory for more information.

PERMISSION MANAGER

Each site will have a permission manager. There is usually no need for you to subclass Apache2::SiteControl::PermissionManager, but you do need to create one and populate it with your access rules. You do this by creating a factory class, which looks something like this:

```
package samples::site::MyPermissionFactory;
use Apache2::SiteControl::PermissionManager;
use Apache2::SiteControl::GrantAllRule;
use samples::site::EditControlRule;
use base qw(Apache2::SiteControl::ManagerFactory);
our $manager;
sub getPermissionManager
{
  return $manager if defined($manager);
  $manager = new Apache2::SiteControl::PermissionManager;
  $manager->addRule(new Apache2::SiteControl::GrantAllRule);
  $manager->addRule(new samples::site::EditControlRule);
  return $manager;
}
1;
```

The primary goal of your factory is to produce an instance of a permission manager that knows the rules for permitting access to your site. This is an easy process that involves calling the constructor (via new) and then calling addRule one or more times.

RULES

The PermissionManager is the object that the site developers ask about what is allowed and what is not. As you saw in the previous section, you create a manager, and add some rules. Each rule is a custom-written class that implements some aspect of your site's access logic. Rules can choose to grant or deny a request. The following is a pretty complex example that demonstrates the features of a rule.

Most rules with either specifically grant permission, or deny it. Most will not deal with both possibilities. In this example we are assuming that the user is implemented as an object that has attributes which can be retrieved with a getAttribute method (of course, you would have to have implemented that as well). The basic action that this rule handles is called "beat up", so the site makes calls like:

```
if($referee->can($userA, "beat up", $userB)) { ... }
```

In terms of English, we would describe the rule "If A is taller than B, then we say that A can beat up B. If A is less skilled than B, then we say that A cannot beat up B". The rule looks like this:

```
package samples::FightRules;
use strict;
use warnings;
use Carp;
use Apache2::SiteControl::Rule;
use base qw(Apache2::SiteControl::Rule);
sub grants($$$$)
{
  my $this = shift;
  my $user = shift;
  my $action = shift;
  my $resource = shift;
  if($action eq "beat up" && $resource->isa("Apache2::SiteControl::User")) {
    my ($h1, $h2);
    $h1 = $user->getAttribute("height");
```

```
        $h2 = $resource->getAttribute("height");
        return 1 if(defined($h1) && defined($h2) && $h1 > $h2);
      }
      return 0;
    }
    sub denies($$$$)
    {
      my $this = shift;
      my $user = shift;
      my $action = shift;
      my $resource = shift;
      if($action eq "beat up" && $resource->isa("Apache2::SiteControl::User")) {
        my ($s1, $s2);
        $s1 = $user->getAttribute("skill");
        $s2 = $resource->getAttribute("skill");
        return 1 if(defined($s1) && defined($s2) && $s1 < $s2);
      }
      return 0;
    }
    1;
```

The PermissionManager will only give permission if at least one rule grants permission, and no rule denies it.

I think it is clearer to separate rules like the previous one into separate rule classes altogether. A HeightMakesMightRule and a DefenseSkillRule. Splitting into two rules makes things clearer, and there is no limit to the number of rules that the PermissionManager can check.

It is important that your rules never grant or deny a request they do not understand, so it is a good idea to use type checking to prevent strangeness. Assertions should not be used if you expect different rules to accept different resource types or user types, since each rule is used on every access request.

EXPORT

None by default.

SEE ALSO

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

Apache2::SiteControl::PermissionManager, Apache::SiteControl::Rule

AUTHOR

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

COPYRIGHT AND LICENSE

This modules is covered by the GNU public license.