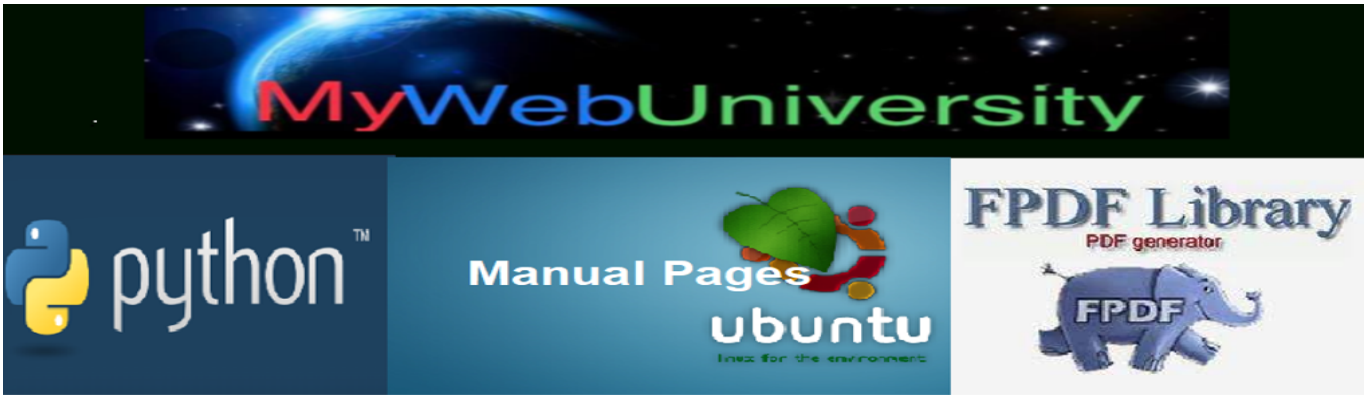

Rocky Enterprise Linux 9.2 Manual Pages on command 'ModPerl::RegistryCooker.3pm'

$ man ModPerl::RegistryCooker.3pm

libapache2-mod-perl2-2.0.12::libapache2-mod-perl2-2.0.12::docs::api::ModPerl::RegistryCooker(3pm)

NAME

ModPerl::RegistryCooker - Cook mod_perl 2.0 Registry Modules

Synopsis

```
# shouldn't be used as-is but sub-classed first
# see ModPerl::Registry for an example
```

Description

"ModPerl::RegistryCooker" is used to create flexible and overridable registry modules which emulate mod_cgi for Perl scripts. The concepts are discussed in the manpage of the following modules: "ModPerl::Registry", "ModPerl::Registry" and "ModPerl::RegistryBB".

"ModPerl::RegistryCooker" has two purposes:

? Provide ingredients that can be used by registry sub-classes

? Provide a default behavior, which can be overridden in sub-classed

META: in the future this functionality may move into a separate class.

Here are the current overridable methods:

META: these are all documented in RegistryCooker.pm, though not using pod. please help to port these to pod and move the descriptions here.

? new()

create the class's object, bless it and return it

```
my $obj = $class->new($r);
```

$class -- the registry class, usually "__PACKAGE__" can be used.

$r -- "Apache2::Request" object.

default: new()

? init()

initializes the data object's fields: "REQ", "FILENAME", "URI". Called from the new().

default: init()

? default_handler()

default: default_handler()

? run()

default: run()

? can_compile()

default: can_compile()

? make_namespace()

default: make_namespace()

? namespace_root()

default: namespace_root()

? namespace_from()

If "namespace_from_uri" is used and the script is called from the virtual host, by default the virtual host name is prepended to the uri when package name for the compiled script is created. Sometimes this behavior is undesirable, e.g., when the same (physical) script is accessed using the same path_info but different virtual hosts. In that case you can make the script compiled only once for all vhosts, by specifying:

```
$ModPerl::RegistryCooker::NameWithVirtualHost = 0;
```

The drawback is that it affects the global environment and all other scripts will be compiled ignoring virtual hosts.

default: namespace_from()

? is_cached()

default: is_cached()

? should_compile()

default: should_compile()

? flush_namespace()

default: flush_namespace()

? cache_table()

default: cache_table()

? cache_it()

default: cache_it()

? read_script()

default: read_script()

? shebang_to_perl()

default: shebang_to_perl()

? get_script_name()

default: get_script_name()

? chdir_file()

default: chdir_file()

? get_mark_line()

default: get_mark_line()

? compile()

default: compile()

? error_check()

default: error_check()

? strip_end_data_segment()

default: strip_end_data_segment()

? convert_script_to_compiled_handler()

default: convert_script_to_compiled_handler()

Special Predefined Functions

The following functions are implemented as constants.

? NOP()

Use when the function shouldn't do anything.

? TRUE()

Use when a function should always return a true value.

? FALSE()

Use when a function should always return a false value.

Sub-classing Techniques

To override the default "ModPerl::RegistryCooker" methods, first, sub-class "ModPerl::RegistryCooker" or one of its existing sub-classes, using "use base". Second, override the methods.

Those methods that weren't overridden will be resolved at run time when used for the first time and cached for the future requests. One way to to shortcut this first run resolution

is to use the symbol aliasing feature. For example to alias "ModPerl::MyRegistry::flush_namespace" as "ModPerl::RegistryCooker::flush_namespace", you can do:

```
package ModPerl::MyRegistry;
use base qw(ModPerl::RegistryCooker);
*ModPerl::MyRegistry::flush_namespace =
    \&ModPerl::RegistryCooker::flush_namespace;
1;
```

In fact, it's a good idea to explicitly alias all the methods so you know exactly what functions are used, rather then relying on the defaults. For that purpose "ModPerl::RegistryCooker" class method install_aliases() can be used. Simply prepare a hash with method names in the current package as keys and corresponding fully qualified methods to be aliased for as values and pass it to install_aliases(). Continuing our example we could do:

```
package ModPerl::MyRegistry;
use base qw(ModPerl::RegistryCooker);
my %aliases = (
    flush_namespace => 'ModPerl::RegistryCooker::flush_namespace',
);
__PACKAGE__->install_aliases(\%aliases);
1;
```

The values use fully qualified packages so you can mix methods from different classes.

Examples

The best examples are existing core registry modules: "ModPerl::Registry", "ModPerl::Registry" and "ModPerl::RegistryBB". Look at the source code and their manpages to see how they subclass "ModPerl::RegistryCooker".

For example by default "ModPerl::Registry" uses the script's path when creating a package's namespace. If for example you want to use a uri instead you can override it with:

```
*ModPerl::MyRegistry::namespace_from =
    \&ModPerl::RegistryCooker::namespace_from_uri;
1;
```

Since the "namespace_from_uri" component already exists in "ModPerl::RegistryCooker". If

you want to write your own method, e.g., that creates a namespace based on the inode, you can do:

```
sub namespace_from_inode {
    my $self = shift;
    return (stat $self->[FILENAME])[1];
}
```

META: when $r->finfo will be ported it'll be more effecient. (stat $r->finfo)[1]

Authors

Doug MacEachern

Stas Bekman

See Also

"ModPerl::Registry", "ModPerl::RegistryBB" and "ModPerl::PerlRun".