



**Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!**

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

***\$ man ModPerl::MethodLookup.3pm***

libapache2-mod-perl2-2.0.12::dolibapache2-mod-perl2-2.0.12::docs::api::ModPerl::MethodLookup(3pm)

NAME

ModPerl::MethodLookup -- Lookup mod\_perl modules, objects and methods

Synopsis

```
use ModPerl::MethodLookup;

# return all module names containing XS method 'print'

my ($hint, @modules) =

    ModPerl::MethodLookup::lookup_method('print');

# return only module names containing method 'print' which
# expects the first argument to be of type 'Apache2::Filter'
# (here $filter is an Apache2::Filter object)

my ($hint, @modules) =

    ModPerl::MethodLookup::lookup_method('print', $filter);

# or

my ($hint, @modules) =

    ModPerl::MethodLookup::lookup_method('print', 'Apache2::Filter');

# what XS methods defined by module 'Apache2::Filter'

my ($hint, @methods) =

    ModPerl::MethodLookup::lookup_module('Apache2::Filter');

# what XS methods can be invoked on the object $r (or a ref)

my ($hint, @methods) =

    ModPerl::MethodLookup::lookup_object($r);

# or
```

```

my ($hint, @methods) =
    ModPerl::MethodLookup::lookup_object('Apache2::RequestRec');
# preload all mp2 modules in startup.pl
ModPerl::MethodLookup::preload_all_modules();
# command line shortcuts
% perl -MModPerl::MethodLookup -e print_module \
    Apache2::RequestRec Apache2::Filter
% perl -MModPerl::MethodLookup -e print_object Apache2
% perl -MModPerl::MethodLookup -e print_method \
    get_server_built request
% perl -MModPerl::MethodLookup -e print_method read
% perl -MModPerl::MethodLookup -e print_method read APR::Bucket

```

## Description

mod\_perl 2.0 provides many methods, which reside in various modules. One has to load each of the modules before using the desired methods. "ModPerl::MethodLookup" provides the Perl API for finding module names which contain methods in question and other helper functions, to find out what methods defined by some module, what methods can be called on a given object, etc.

## API

"lookup\_method()"

Find modules (packages) containing a certain method

```
($hint, @modules) = lookup_method($method_name);
```

```
($hint, @modules) = lookup_method($method_name, $object);
```

```
($hint, @modules) = lookup_method($method_name, $class);
```

arg1: \$method\_name ( string )

the method name to look up

opt arg2: \$object or \$class

a blessed object or the name of the class it's blessed into. If there is more than one match, this extra information is used to return only modules containing methods operating on the objects of the same kind.

If a sub-classed object is passed it'll be handled correctly, by checking its super-class(es). This usage is useful when the "AUTOLOAD" is used to find a not yet loaded module which include the called method.

ret1: \$hint

a string containing a human readable lookup result, suggesting which modules should be loaded, ready for copy-n-paste or explaining the failure if the lookup didn't succeed.

ret2: @modules

an array of modules which have matched the query, i.e. the names of the modules which contain the requested method.

since: 2.0.00

Examples:

Return all module names containing XS method print:

```
my ($hint, @modules) =  
    ModPerl::MethodLookup::lookup_method('print');
```

Return only module names containing method print which expects the first argument to be of type "Apache2::Filter":

```
my $filter = bless {}, 'Apache2::Filter';  
my ($hint, @modules) =  
    ModPerl::MethodLookup::lookup_method('print', $filter);
```

or:

```
my ($hint, @modules) =  
    ModPerl::MethodLookup::lookup_method('print', 'Apache2::Filter');
```

"lookup\_module()"

Find methods contained in a certain module (package)

```
($hint, @methods) = lookup_module($module_name);
```

arg1: \$module\_name ( string )

the module name

ret1: \$hint

a string containing a human readable lookup result, suggesting, which methods the module \$module\_name implements, or explaining the failure if the lookup failed.

ret2: @methods

an array of methods which have matched the query, i.e. the names of the methods defined in the requested module.

since: 2.0.00

Example:

What XS methods defined by module "Apache2::Filter":

```
my ($hint, @methods) =  
    ModPerl::MethodLookup::lookup_module('Apache2::Filter');
```

"lookup\_object()"

```
($hint, @methods) = lookup_object($object);
```

```
($hint, @methods) = lookup_object($class);
```

arg1: \$object or \$class

an object or a name of a class an object is blessed into

If a sub-classed object is passed it'll be handled correctly, by including methods provided by its super-class(es).

ret1: \$hint

a string containing a human readable lookup result, suggesting, which methods the given object can invoke (including module names that need to be loaded to use those methods), or explaining the failure if the lookup failed.

ret2: @methods

an array of methods which have matched the query, i.e. the names of the methods that can be invoked on the given object (or its class name).

since: 2.0.00

META: As of this writing this function may miss some of the functions/methods that can be invoked on the given object. Currently we can't programmatically deduct the objects they are invoked on, because these methods are written in pure XS and manipulate the arguments stack themselves. Currently these are mainly XS functions, not methods, which of course aren't invoked on objects. There are also logging function wrappers ("Apache2::Log").

Examples:

What XS methods can be invoked on the object \$r:

```
my ($hint, @methods) =  
    ModPerl::MethodLookup::lookup_object($r);
```

or \$r's class -- "Apache2::RequestRec":

```
my ($hint, @methods) =  
    ModPerl::MethodLookup::lookup_object('Apache2::RequestRec');
```

"preload\_all\_modules()"

The function "preload\_all\_modules()" preloads all mod\_perl 2.0 modules, which implement their API in XS. This is similar to the mod\_perl 1.0 behavior which has most of its methods loaded at the startup.

CPAN modules developers should make sure their distribution loads each of the used mod\_perl 2.0 modules explicitly, and not use this function, as it takes the fine control away from the users. One should avoid doing this the production server (unless all modules are used indeed) in order to save memory.

since: 2.0.00

"print\_method()"

"print\_method()" is a convenience wrapper for "lookup\_method()", mainly designed to be used from the command line. For example to print all the modules which define method read execute:

```
% perl -MModPerl::MethodLookup -e print_method read
```

Since this will return more than one module, we can narrow the query to only those methods which expect the first argument to be blessed into class "APR::Bucket":

```
% perl -MModPerl::MethodLookup -e print_method read APR::Bucket
```

You can pass more than one method and it'll perform a lookup on each of the methods. For example to lookup methods "get\_server\_built" and "request" you can do:

```
% perl -MModPerl::MethodLookup -e print_method \  
get_server_built request
```

The function "print\_method()" is exported by default.

since: 2.0.00

"print\_module()"

"print\_module()" is a convenience wrapper for "lookup\_module()", mainly designed to be used from the command line. For example to print all the methods defined in the module "Apache2::RequestRec", followed by methods defined in the module "Apache2::Filter" you can run:

```
% perl -MModPerl::MethodLookup -e print_module \  
Apache2::RequestRec Apache2::Filter
```

The function "print\_module()" is exported by default.

since: 2.0.00

"print\_object()"

"print\_object()" is a convenience wrapper for "lookup\_object()", mainly designed to be used from the command line. For example to print all the methods that can be invoked on object blessed into a class "Apache2::RequestRec" run:

```
% perl -MModPerl::MethodLookup -e print_object \  

```

Apache2::RequestRec

Similar to "print\_object()", more than one class can be passed to this function.

The function "print\_object()" is exported by default.

since: 2.0.00

## Applications

### "AUTOLOAD"

When Perl fails to locate a method it checks whether the package the object belongs to has an "AUTOLOAD" function defined and if so, calls it with the same arguments as the missing method while setting a global variable \$AUTOLOAD (in that package) to the name of the originally called method. We can use this facility to lookup the modules to be loaded when such a failure occurs. Though since we have many packages to take care of we will use a special "UNIVERSAL::AUTOLOAD" function which Perl calls if can't find the "AUTOLOAD" function in the given package.

In that function you can query "ModPerl::MethodLookup", require() the module that includes the called method and call that method again using the goto() trick:

```
use ModPerl::MethodLookup;

sub UNIVERSAL::AUTOLOAD {

    my ($hint, @modules) =

        ModPerl::MethodLookup::lookup_method($UNIVERSAL::AUTOLOAD, @_);

    if (@modules) {

        eval "require $_" for @modules;

        goto &$UNIVERSAL::AUTOLOAD;

    }

    else {

        die $hint;

    }

}
```

However we don't endorse this approach. It's a better approach to always abort the execution which printing the \$hint and use fix the code to load the missing module.

Moreover installing "UNIVERSAL::AUTOLOAD" may cause a lot of problems, since once it's installed Perl will call it every time some method is missing (e.g. undefined "DESTROY"

methods). The following approach seems to somewhat work for me. It installs

"UNIVERSAL::AUTOLOAD" only when the the child process starts.

httpd.conf:

-----

PerlChildInitHandler ModPerl::MethodLookupAuto

startup.pl:

-----

```
{
    package ModPerl::MethodLookupAuto;
    use ModPerl::MethodLookup;
    use Carp;
    sub handler {
        *UNIVERSAL::AUTOLOAD = sub {
            my $method = $AUTOLOAD;
            return if $method =~ /DESTROY/; # exclude DESTROY resolving
            my ($hint, @modules) =
                ModPerl::MethodLookup::lookup_method($method, @_);
            $hint ||= "Can't find method $AUTOLOAD";
            croak $hint;
        };
        return 0;
    }
}
```

This example doesn't load the modules for you. It'll print to STDERR what module should be loaded, when a method from the not-yet-loaded module is called.

A similar technique is used by "Apache2::porting".

META: there is a better version of AUTOLOAD discussed on the dev list. Replace the current one with it. (search the archive for EazyLife)

### Command Line Lookups

When a method is used and mod\_perl has reported a failure to find it, it's often useful to use the command line query to figure out which module needs to be loaded. For example if when executing:

```
$r->construct_url();
```

mod\_perl complains:

```
Can't locate object method "construct_url" via package
```

"Apache2::RequestRec" at ...

you can ask "ModPerl::MethodLookup" for help:

```
% perl -MModPerl::MethodLookup -e print_method construct_url
```

To use method 'construct\_url' add:

```
use Apache2::URI ();
```

and after copy-n-pasting the use statement in our code, the problem goes away.

One can create a handy alias for this technique. For example, C-style shell users can do:

```
% alias lookup "perl -MModPerl::MethodLookup -e print_method"
```

For Bash-style shell users:

```
% alias lookup="perl -MModPerl::MethodLookup -e print_method"
```

Now the lookup is even easier:

```
% lookup construct_url
```

to use method 'construct\_url' add:

```
use Apache2::URI;
```

Similar aliases can be provided for "print\_object()" and "print\_module()".

## Todo

These methods aren't yet picked by this module (the extract from the map file):

```
modperl_filter_attributes | MODIFY_CODE_ATTRIBUTES
```

```
modperl_spawn_proc_prog | spawn_proc_prog
```

```
apr_ipsubnet_create | new
```

Please report to the mod\_perl development mailing list if you find any other missing methods. But remember that as of this moment the module reports only XS functions. In the future we may add support for pure perl functions/methods as well.

## See Also

- ? the mod\_perl 1.0 backward compatibility document
- ? porting Perl modules
- ? porting XS modules
- ? "Apache2::porting"

## Copyright

mod\_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 2.0.

## Authors

The mod\_perl development team and numerous contributors.

