

# **1 Apache2::compat -- 1.0 backward compatibility functions deprecated in 2.0**

## 1.1 Synopsis

```
# either add at the very beginning of startup.pl
use Apache2::compat;
# or httpd.conf
PerlModule Apache2::compat

# override and restore compat functions colliding with mp2 API
Apache2::compat::override_mp2_api('Apache2::Connection::local_addr');
my ($local_port, $local_addr) = sockaddr_in($c->local_addr);
Apache2::compat::restore_mp2_api('Apache2::Connection::local_addr');
```

## 1.2 Description

`Apache2::compat` provides `mod_perl` 1.0 compatibility layer and can be used to smooth the transition process to `mod_perl` 2.0.

It includes functions that have changed their API or were removed in `mod_perl` 2.0. If your code uses any of those functions, you should load this module at the server startup, and everything should work as it did in 1.0. If it doesn't please report the bug, but before you do that please make sure that your code does work properly under `mod_perl` 1.0.

However, remember, that it's implemented in pure Perl and not C, therefore its functionality is not optimized and it's the best to try to port your code not to use deprecated functions and stop using the compatibility layer.

## 1.3 Compatibility Functions Colliding with `mod_perl` 2.0 API

Most of the functions provided by `Apache2::compat` don't interfere with `mod_perl` 2.0 API. However there are several functions which have the same name in the `mod_perl` 1.0 and `mod_perl` 2.0 API, accept the same number of arguments, but either the arguments themselves aren't the same or the return values are different. For example the `mod_perl` 1.0 code:

```
require Socket;
my $sockaddr_in = $c->local_addr;
my ($local_port, $local_addr) = Socket::sockaddr_in($sockaddr_in);
```

should be adjusted to be:

```
require Apache2::Connection;
require APR::SockAddr;
my $sockaddr = $c->local_addr;
my ($local_port, $local_addr) = ($sockaddr->port, $sockaddr->ip_get);
```

to work under `mod_perl` 2.0.

As you can see in `mod_perl 1.0` API `local_addr()` was returning a `SOCKADDR_IN` object (see the `Socket perl` manpage), in `mod_perl 2.0` API it returns an `APR::SockAddr` object, which is a totally different beast. If `Apache2::compat` overrides the function `local_addr()` to be back-compatible with `mod_perl 1.0` API. Any code that relies on this function to work as it should under `mod_perl 2.0` will be broken. Therefore the solution is not to override `local_addr()` by default. Instead a special API is provided which overrides colliding functions only when needed and which can be restored when no longer needed. So for example if you have code from `mod_perl 1.0`:

```
my ($local_port, $local_addr) = Socket::sockaddr_in($c->local_addr);
```

and you aren't ready to port it to to use the `mp2` API:

```
my ($local_port, $local_addr) = ($c->local_addr->port,
                                 $c->local_addr->ip_get);
```

you could do the following:

```
Apache2::compat::override_mp2_api('Apache2::Connection::local_addr');
my ($local_port, $local_addr) = Socket::sockaddr_in($c->local_addr);
Apache2::compat::restore_mp2_api('Apache2::Connection::local_addr');
```

Notice that you need to restore the API as soon as possible.

Both `override_mp2_api()` and `restore_mp2_api()` accept a list of functions to operate on.

### 1.3.1 Available Overridable Functions

At the moment the following colliding functions are available for overriding:

- **Apache2::RequestRec::notes**
- **Apache2::RequestRec::filename**
- **Apache2::RequestRec::finfo**
- **Apache2::Connection::local\_addr**
- **Apache2::Connection::remote\_addr**
- **Apache2::Util::ht\_time**
- **Apache2::Module::top\_module**
- **Apache2::Module::get\_config**
- **APR::URI::unparse**

## 1.4 Use in CPAN Modules

The short answer: **Do not use** `Apache2::compat` in CPAN modules.

The long answer:

`Apache2::compat` is useful during the `mod_perl 1.0` code porting. Though remember that it's implemented in pure Perl. In certain cases it overrides `mod_perl 2.0` methods, because their API is very different and doesn't map 1:1 to `mod_perl 1.0`. So if anything, not under user's control, loads `Apache2::compat`

user's code is forced to use the potentially slower method. Which is quite bad.

Some users may choose to keep using `Apache2::compat` in production and it may perform just fine. Other users will choose not to use that module, by porting their code to use `mod_perl 2.0` API. However it should be users' choice whether to load this module or not and not to be enforced by CPAN modules.

If you port your CPAN modules to work with `mod_perl 2.0`, you should follow the porting Perl and XS module guidelines.

Users that are stuck with CPAN modules preloading `Apache2::compat`, can prevent this from happening by adding

```
$INC{'Apache2/compat.pm'} = __FILE__;
```

at the very beginning of their *startup.pl*. But this will most certainly break the module that needed this module.

## 1.5 API

You should be reading the `mod_perl 1.0` API docs for usage of the methods and functions in this package, since what this module is doing is providing a backwards compatibility and it makes no sense to duplicate documentation.

Another important document to read is: *Migrating from mod\_perl 1.0 to mod\_perl 2.0* which covers all `mod_perl 1.0` constants, functions and methods that have changed in `mod_perl 2.0`.

## 1.6 See Also

`mod_perl 2.0` documentation.

## 1.7 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 2.0.

## 1.8 Authors

The `mod_perl` development team and numerous contributors.

## Table of Contents:

1	Apache2::compat -- 1.0 backward compatibility functions deprecated in 2.0	1
1.1	Synopsis	2
1.2	Description	2
1.3	Compatibility Functions Colliding with mod_perl 2.0 API	2
1.3.1	Available Overridable Functions	3
1.4	Use in CPAN Modules	3
1.5	API	4
1.6	See Also	4
1.7	Copyright	4
1.8	Authors	4