



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 'Apache2::RequestIO.3pm'

\$ man Apache2::RequestIO.3pm

libapache2-mod-perl2-2.0.12::doUse/libapache2-mod-perl2-2.0.12::docs::api::Apache2::RequestIO(3pm)

NAME

Apache2::RequestIO - Perl API for Apache request record IO

Synopsis

```
use Apache2::RequestIO ();

$rc = $r->discard_request_body();

$r->print("foo", "bar");

$r->puts("foo", "bar"); # same as print, but no flushing

$r->printf("%s $d", "foo", 5);

$r->read($buffer, $len);

$r->rflush();

$r->sendfile($filename);

$r->write("foobartarcar", 3, 5);
```

Description

"Apache2::RequestIO" provides the API to perform IO on the Apache request object.

API

"Apache2::RequestIO" provides the following functions and/or methods:

"discard_request_body"

In HTTP/1.1, any method can have a body. However, most GET handlers wouldn't know what to do with a request body if they received one. This helper routine tests for and reads any message body in the request, simply discarding whatever it receives. We need to do this because failing to read the request body would cause it to be interpreted as the next request on a persistent connection.

```
$rc = $r->discard_request_body();
```

obj: \$r ("Apache2::RequestRec object")

The current request

ret: \$rc (integer)

"APR::Const status constant" if request is malformed, "Apache2::Const::OK" otherwise.

since: 2.0.00

Since we return an error status if the request is malformed, this routine should be called at the beginning of a no-body handler, e.g.,

```
use Apache2::Const -compile => qw(OK);
```

```
$rc = $r->discard_request_body;
```

```
return $rc if $rc != Apache2::Const::OK;
```

"print"

Send data to the client.

```
$cnt = $r->print(@msg);
```

obj: \$r ("Apache2::RequestRec object")

arg1: @msg (ARRAY)

Data to send

ret: \$cnt (number)

How many bytes were sent (or buffered). If zero bytes were sent, "print" will return

0E0, or "zero but true," which will still evaluate to 0 in a numerical context.

excpt: "APR::Error"

since: 2.0.00

The data is flushed only if STDOUT stream's \$| is true. Otherwise it's buffered up to the size of the buffer, flushing only excessive data.

"printf"

Format and send data to the client (same as "print").

```
$cnt = $r->printf($format, @args);
```

obj: \$r ("Apache2::RequestRec object")

arg1: \$format (string)

Format string, as in the Perl core "printf" function.

arg2: @args (ARRAY)

Arguments to be formatted, as in the Perl core "printf" function.

ret: \$cnt (number)

How many bytes were sent (or buffered)

excpt: "APR::Error"

since: 2.0.00

The data is flushed only if STDOUT stream's \$| is true. Otherwise it's buffered up to the size of the buffer, flushing only excessive data.

"puts"

Send data to the client

```
$cnt = $r->puts(@msg);
```

obj: \$r ("Apache2::RequestRec object")

arg1: @msg (ARRAY)

Data to send

ret: \$cnt (number)

How many bytes were sent (or buffered)

excpt: "APR::Error"

since: 2.0.00

"puts()" is similar to "print()", but it won't attempt to flush data, no matter what the value of STDOUT stream's \$| is. Therefore assuming that STDOUT stream's \$| is true, this method should be a tiny bit faster than "print()", especially if small strings are printed.

"read"

Read data from the client.

```
$cnt = $r->read($buffer, $len);
```

```
$cnt = $r->read($buffer, $len, $offset);
```

obj: \$r ("Apache2::RequestRec object")

arg1: \$buffer (SCALAR)

The buffer to populate with the read data

arg2: \$len (number)

How many bytes to attempt to read

opt arg3: \$offset (number)

If a non-zero \$offset is specified, the read data will be placed at that offset in the \$buffer.

META: negative offset and \0 padding are not supported at the moment

ret: \$cnt (number)

How many characters were actually read

excpt: "APR::Error"

since: 2.0.00

This method shares a lot of similarities with the Perl core "read()" function. The main difference in the error handling, which is done via "APR::Error exceptions"

"rflush"

Flush any buffered data to the client.

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

obj: \$r ("Apache2::RequestRec object")

ret: no return value

since: 2.0.00

Unless STDOUT stream's \$| is false, data sent via "\$r->print()" is buffered. This method flushes that data to the client.

"sendfile"

Send a file or a part of it

```
$rc = $r->sendfile($filename);
```

```
$rc = $r->sendfile($filename, $offset);
```

```
$rc = $r->sendfile($filename, $offset, $len);
```

obj: \$r ("Apache2::RequestRec object")

arg1: \$filename (string)

The full path to the file (using "/" on all systems)

opt arg2: \$offset (integer)

Offset into the file to start sending.

No offset is used if \$offset is not specified.

opt arg3: \$len (integer)

How many bytes to send.

If not specified the whole file is sent (or a part of it, if \$offset if specified)

ret: \$rc ("APR::Const status constant")

On success, "APR::Const::SUCCESS" is returned.

In case of a failure -- a failure code is returned, in which case normally it should be returned to the caller.

excpt: "APR::Error"

Exceptions are thrown only when this function is called in the VOID context. So if you

don't want to handle the errors, just don't ask for a return value and the function will handle all the errors on its own.

since: 2.0.00

"write"

Send partial string to the client

```
$cnt = $r->write($buffer);
```

```
$cnt = $r->write($buffer, $len);
```

```
$cnt = $r->write($buffer, $len, $offset);
```

obj: \$r ("Apache2::RequestRec object")

arg1: \$buffer (SCALAR)

The string with data

opt arg2: \$len (SCALAR)

How many bytes to send. If not specified, or -1 is specified, all the data in \$buffer

(or starting from \$offset) will be sent.

opt arg3: \$offset (number)

Offset into the \$buffer string.

ret: \$cnt (number)

How many bytes were sent (or buffered)

excpt: "APR::Error"

since: 2.0.00

Examples:

Assuming that we have a string:

```
$string = "123456789";
```

Then:

```
$r->write($string);
```

sends:

```
123456789
```

Whereas:

```
$r->write($string, 3);
```

sends:

```
123
```

And:

```
$r->write($string, 3, 5);
```

sends:

678

Finally:

```
$r->write($string, -1, 5);
```

sends:

6789

TIE Interface

The TIE interface implementation. This interface is used for HTTP request handlers, when running under "SetHandler perl-script" and Perl doesn't have perlio enabled.

See the [perltie manpage](#) for more information.

"BINMODE"

since: 2.0.00

NoOP

See the [binmode Perl entry in the perlfunc manpage](#)

"CLOSE"

since: 2.0.00

NoOP

See the [close Perl entry in the perlfunc manpage](#)

"FILENO"

since: 2.0.00

See the [fileno Perl entry in the perlfunc manpage](#)

"GETC"

since: 2.0.00

See the [getc Perl entry in the perlfunc manpage](#)

"OPEN"

since: 2.0.00

See the [open Perl entry in the perlfunc manpage](#)

"PRINT"

since: 2.0.00

See the [print Perl entry in the perlfunc manpage](#)

"PRINTF"

since: 2.0.00

See the [printf Perl entry in the perlfunc manpage](#)

"READ"

since: 2.0.00

See the read Perl entry in the perfunc manpage

"TIEHANDLE"

since: 2.0.00

See the tie Perl entry in the perfunc manpage

"UNTIE"

since: 2.0.00

NoOP

See the untie Perl entry in the perfunc manpage

"WRITE"

since: 2.0.00

See the write Perl entry in the perfunc manpage

Deprecated API

The following methods are deprecated, Apache plans to remove those in the future, therefore avoid using them.

"get_client_block"

This method is deprecated since the C implementation is buggy and we don't want you to use it at all. Instead use the plain "\$r->read()".

"setup_client_block"

This method is deprecated since "\$r->get_client_block" is deprecated.

"should_client_block"

This method is deprecated since "\$r->get_client_block" is deprecated.

See Also

mod_perl 2.0 documentation.

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.

perl v5.34.0

libapache2-mod-perl2-2.0.12::docs::api::Apache2::RequestIO(3pm)