Input/output streams ¶

The CLI SAPI defines a few constants for I/O streams to make programming for the command line a bit easier.

CLI specific Constants

Constant	Description
`STDIN`	An already opened stream to *stdin*. This saves opening it with `<?php $stdin = fopen('php://stdin', 'r'); ?>` If you want to read single line from *stdin*, you can use `<?php $line = trim(fgets(STDIN)); // reads one line from STDIN` `fscanf(STDIN, "%d\n", $number); // reads number from STDIN ?>`
`STDOUT`	An already opened stream to *stdout*. This saves opening it with `<?php $stdout = fopen('php://stdout', 'w'); ?>`
`STDERR`	An already opened stream to *stderr*. This saves opening it with `<?php $stderr = fopen('php://stderr', 'w'); ?>`

Given the above, you don't need to open e.g. a stream for *stderr* yourself but simply use the constant instead of the stream resource:

```
php -r 'fwrite(STDERR, "stderr\n");'
```

You do not need to explicitly close these streams, as they are closed automatically by PHP when your script ends.

Nota:

These constants are not available if reading the PHP script from *stdin*.

add a note

User Contributed Notes 3 notes

up
down
2
Aurelien Marchand ¶
3 years ago

```
Please remember in multi-process applications (which are best suited under CLI),
that I/O operations often will BLOCK signals from being processed.

For instance, if you have a parent waiting on fread(STDIN), it won't handle SIGCHLD,
even if you defined a signal handler for it, until after the call to fread has
returned.

Your solution in this case is to wait on stream_select() to find out whether reading
will block. Waiting on stream_select(), critically, does NOT BLOCK signals from
being processed.

Aurelien
```

up
down
1
ecrist at secure-computing dot net ¶
3 years ago

```
The following code shows how to test for input on STDIN.  In this case, we were
looking for CSV data, so we use fgetcsv to read STDIN, if it creates an array, we
assume CVS input on STDIN, if no array was created, we assume there's no input from
STDIN, and look, later, to an argument with a CSV file name.

Note, without the stream_set_blocking() call, fgetcsv() hangs on STDIN, awaiting
input from the user, which isn't useful as we're looking for a piped file. If it
isn't here already, it isn't going to be.

<?php
stream_set_blocking(STDIN, 0);
$csv_ar = fgetcsv(STDIN);
if (is_array($csv_ar)){
  print "CVS on STDIN\n";
} else {
  print "Look to ARGV for CSV file name.\n";
}
?>
```

up
down
-2
James Zhu ¶
3 years ago

```
Example:

<?php
function ReadStdin($prompt, $valid_inputs, $default = '') {
    while(!isset($input) || (is_array($valid_inputs) && !in_array($input,
$valid_inputs)) || ($valid_inputs == 'is_file' && !is_file($input))) {
        echo $prompt;
        $input = strtolower(trim(fgets(STDIN)));
        if(empty($input) && !empty($default)) {
            $input = $default;
        }
    }
    return $input;
}

// you can input <Enter> or 1, 2, 3
$choice = ReadStdin('Please choose your answer or press Enter to continue: ',
array('', '1', '2', '3'));

// check input is valid file name, use /var/path for input nothing
$file = ReadStdin('Please input the file name(/var/path):', 'is_file', '/var/path');
?>

you can add more functions if you want.
```

add a note