PHP – Filters

It is important that the input data received in the form of client request is validated before processing in a PHP application. To perform input validation, the filter extension in PHP provides a number of filter functions, backed up by predefined filter constants and flags. The filter extension of PHP library also helps in sanitizing the input received by either GET or POST methods.

The filter extension is a powerful feature that helps prevention of security vulnerabilities, such as SQL injection and cross-site scripting. The extension has two types of filters –

Validation Filters

Validation filters check if the data meets certain criteria. For example, you want to ensure that the user has correctly input an email field in the HTML form. The FILTER_VALIDATE_EMAIL will determine if the data is a valid email address. The validation filters, however, will not change the data itself.

Sanitization Filters

Sanitization refers to the process of removing undesired characters from the input. Hence, it may alter the data by removing undesired characters. For example, passing in FILTER_SANITIZE_EMAIL will remove characters that are inappropriate for an email address to contain, without performing validation.

Filter Flags

The filter extension in PHP defines a number of **filter flags** as follows –

Sr.No	ID & Description
1	**FILTER_FLAG_STRIP_LOW** Strips characters that have a numerical value <32.
2	**FILTER_FLAG_STRIP_HIGH** Strips characters that have a numerical value >127.
3	**FILTER_FLAG_STRIP_BACKTICK** Strips backtick characters.
4	**FILTER_FLAG_ALLOW_FRACTION** Allows a period (.) as a fractional separator in numbers.

5	**FILTER_FLAG_ALLOW_THOUSAND** Allows a comma (,) as a thousands separator in numbers.
6	**FILTER_FLAG_ALLOW_SCIENTIFIC** Allows an e or E for scientific notation in numbers.
7	**FILTER_FLAG_NO_ENCODE_QUOTES** If this flag is present, single (') and double (") quotes will not be encoded.
8	**FILTER_FLAG_ENCODE_LOW** Encodes all characters with a numerical value <32.
9	**FILTER_FLAG_ENCODE_HIGH** Encodes all characters with a numerical value >127.
10	**FILTER_FLAG_ENCODE_AMP** Encodes ampersands (&).
11	**FILTER_NULL_ON_FAILURE** Returns **null** for unrecognized values.
12	**FILTER_FLAG_ALLOW_OCTAL** Regards inputs starting with a zero (0) as octal numbers.
13	**FILTER_FLAG_ALLOW_HEX** Regards inputs starting with 0x or 0X as hexadecimal numbers.
14	**FILTER_FLAG_EMAIL_UNICODE** Allows the local part of the email address to contain Unicode characters.
15	**FILTER_FLAG_IPV4** Allows the IP address to be in IPv4 format.
16	**FILTER_FLAG_IPV6** Allows the IP address to be in IPv6 format.
17	**FILTER_FLAG_NO_PRIV_RANGE** Fails validation for the following private IPv4 ranges: 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16.
18	**FILTER_FLAG_NO_RES_RANGE** Fails validation for the following reserved IPv4 ranges: 0.0.0.0/8, 169.254.0.0/16, 127.0.0.0/8 and 240.0.0.0/4. Fails validation for the following reserved IPv6 ranges: ::1/128, ::/128, ::ffff:0:0/96 and fe80::/10.
19	**FILTER_FLAG_GLOBAL_RANGE** Fails validation for non global IPv4/IPv6 ranges

20	**FILTER_FLAG_SCHEME_REQUIRED** Requires the URL to contain a scheme part.
21	**FILTER_FLAG_HOST_REQUIRED** Requires the URL to contain a host part.
22	**FILTER_FLAG_PATH_REQUIRED** Requires the URL to contain a path part.
23	**FILTER_FLAG_QUERY_REQUIRED** Requires the URL to contain a query string.
24	**FILTER_REQUIRE_SCALAR** Requires the value to be scalar.
25	**FILTER_REQUIRE_ARRAY** Requires the value to be an array.
26	**FILTER_FORCE_ARRAY** If the value is a scalar, it is treated as array with the scalar value as only element.

Filter Functions

The filter extension includes the following **filter functions** –

Sr.No	ID & Description
1	**filter_has_var()** Checks if variable of specified type exists
2	**filter_id()** Returns the filter ID belonging to a named filter
3	**filter_input_array()** Gets external variables and optionally filters them
4	**filter_input ()** Gets a specific external variable by name and filters it
5	**filter_list()** Returns a list of all supported filters
6	**filter_var_array()** Gets multiple variables and optionally filters them
7	**filter_var()**

	Filters a variable with a specified filter

Predefined Constants

The above functions use one parameter called input_type which is one of the predefined enumerated constants representing how the input has been provided to the PHP script for filtering purpose.

Constant	Types
INPUT_POST (int)	POST Variables
INPUT_GET (int)	GET Variables
INPUT_COOKIE (int)	COOKIE Variables
INPUT_ENV (int)	ENV Variables
INPUT_SERVER (int)	SERVER Variables
INPUT_SESSION (int)	SESSION Variables
INPUT_REQUEST (int)	REQUEST Variables

filter_has_var() function

The filter_has_var() function checks if variable of specified type exists.

```
filter_has_var(int $input_type, string $var_name): bool
```

The input_type is one of predefined constants INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, or INPUT_ENV; where as the var_name parameter is the name of a variable to check. The function returns true on success or false on failure.

Example

Visit the following PHP script on the XAMPP server.

```
<?php
   if (!filter_has_var(INPUT_GET, "email")) {
      echo("Email not found");
   } else {
      echo("Email found");
   }
?>
```

It will produce the following **output** –

Visit **http://localhost/hello.php?email=abc@example.com**

```
Email found
```

filter_input() function

The filter_input() function gets a specific external variable by name and filters it accorfing to the applied filter constant

```
filter_input(
   int $type,
   string $var_name,
   int $filter = FILTER_DEFAULT,
   array|int $options = 0
): mixed
```

The type parameter is one of the constants INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, or INPUT_ENV. Second parameter is var_name, the name of a variable to get. You can use the filter to be applied. Use any of the predefined filter flags. If omitted, FILTER_DEFAULT will be used

The function returns the value of the requested variable on success, false if the filter fails, or null if the var_name variable is not set.

Example

Take a look at the following example –

```
<?php
   if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL)) {
      echo("Email is not valid");
   } else {
      echo("Email is valid");
```

```
    }
?>
```

It will produce the following **output** −

If you use the URL **http://localhost/hello.php?email=abc@example.com**,

```
Email is valid
```

If the URL is **http://localhost/hello.php?email=a b c@example.com**,

```
Email is not valid
```

You can also use INPUT_POST type for validating the input received through the POST method −

```
<?php
   if (!filter_input(INPUT_POST, "email", FILTER_VALIDATE_EMAIL)) {
      echo("Email is not valid");
   } else {
      echo("Email is valid");
   }
?>
```

To pass data with POST request, open the command prompt, and use the following CURL command

```
curl -X POST -d "{\"email\": \"a@b.com\"}" http://localhost/hello.php
```

filter_list() function

The filter_list() function returns a list of all supported filters

```
filter_list(): array
```

Example

The function returns an array of names of all supported filters, empty array if there are no such filters.

Open Compiler

```php
<?php
   print_r(filter_list());
?>
```

It will produce the following **output** –

```
Array
(
   [0] => int
   [1] => boolean
   [2] => float
   [3] => validate_regexp
   [4] => validate_domain
   [5] => validate_url
   [6] => validate_email
   [7] => validate_ip
   [8] => validate_mac
   [9] => string
   [10] => stripped
   [11] => encoded
   [12] => special_chars
   [13] => full_special_chars
   [14] => unsafe_raw
   [15] => email
   [16] => url
   [17] => number_int
   [18] => number_float
   [19] => add_slashes
   [20] => callback
)
```

filter_input_array() function

The filter_input_array() gets external variables and optionally filters them.

```php
filter_input_array(int $type, array|int $options = FILTER_DEFAULT,
   bool $add_empty = true): array|false|null
```

This function is useful for retrieving many values without repetitively calling filter_input().

The type parameter is one of INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, or INPUT_ENV.

The options parameter is an array defining the arguments. A valid key is a string containing a variable name and a valid value is either a filter type, or an array optionally specifying the filter, flags and options. This parameter can be also an integer holding a filter constant. Then all values in the input array are filtered by this filter.

The function returns an array containing the values of the requested variables on success. If the input array designated by type is not populated, the function returns null if the FILTER_NULL_ON_FAILURE flag is not given, or false otherwise. For other failures, false is returned.

Example

To include an array in the HTTP request, we use the following HTML form in "hello.html", and send it by POST method.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<body>
   <h1>Filter Input Array</h1>
   <form action="hello.php" method="POST">
      <p><label for="email">Enter your email:</label>
      <input type="text" id="email" name="email"></p>
      <p><label for="age">Enter your age<label>
      <input type = "text" id="age" name="age"></p>
      <input type="submit">
   </form>
</body>
</html>
```

The PHP script to validate the input array is as follows −

```
<?php
   $filters = array (
      "age" => array ("filter"=>FILTER_VALIDATE_INT,
         "options"=>array("min_range"=>20,"max_range"=>40) ),
      "email" => FILTER_VALIDATE_EMAIL
   );
   print_r(filter_input_array(INPUT_POST, $filters));
?>
```

Open the HTML form and enter 30 as age, **abc@example.com** as email, the result will be an array, validating both the inputs −

```
Array ( [age] => 30 [email] => abc@example.com )
```

Try giving invalid inputs such as "age=15". The output array will show a null value for **age** key

```
Array ( [age] => [email] => abc@example.com )
```