input.php

```php
<?php

namespace dokuwiki\Input;

/**
 * Encapsulates access to the $_REQUEST array, making sure used
parameters are initialized and
 * have the correct type.
 *
 * All function access the $_REQUEST array by default, if you want to
access $_POST or $_GET
 * explicitly use the $post and $get members.
 *
 * @author Andreas Gohr <andi@splitbrain.org>
 */
class Input
{

    /** @var Post Access $_POST parameters */
    public $post;
    /** @var Get Access $_GET parameters */
    public $get;
    /** @var Server Access $_SERVER parameters */
    public $server;

    protected $access;

    /**
     * @var Callable
     */
    protected $filter;

    /**
     * Intilizes the dokuwiki\Input\Input class and it subcomponents
     */
    public function __construct()
    {
        $this->access = &$_REQUEST;
        $this->post = new Post();
        $this->get = new Get();
        $this->server = new Server();
    }

    /**
     * Apply the set filter to the given value
     *
     * @param string $data
     * @return string
     */
```

```
    protected function applyfilter($data)
    {
        if (!$this->filter) return $data;
        return call_user_func($this->filter, $data);
    }

    /**
     * Return a filtered copy of the input object
     *
     * Expects a callable that accepts one string parameter and returns
a filtered string
     *
     * @param Callable|string $filter
     * @return Input
     */
    public function filter($filter = 'stripctl')
    {
        $this->filter = $filter;
        $clone = clone $this;
        $this->filter = '';
        return $clone;
    }

    /**
     * Check if a parameter was set
     *
     * Basically a wrapper around isset. When called on the $post and
$get subclasses,
     * the parameter is set to $_POST or $_GET and to $_REQUEST
     *
     * @see isset
     * @param string $name Parameter name
     * @return bool
     */
    public function has($name)
    {
        return isset($this->access[$name]);
    }

    /**
     * Remove a parameter from the superglobals
     *
     * Basically a wrapper around unset. When NOT called on the $post
and $get subclasses,
     * the parameter will also be removed from $_POST or $_GET
     *
     * @see isset
     * @param string $name Parameter name
     */
    public function remove($name)
    {
```

```
        if (isset($this->access[$name])) {
            unset($this->access[$name]);
        }
        // also remove from sub classes
        if (isset($this->post) && isset($_POST[$name])) {
            unset($_POST[$name]);
        }
        if (isset($this->get) && isset($_GET[$name])) {
            unset($_GET[$name]);
        }
    }

    /**
     * Access a request parameter without any type conversion
     *
     * @param string $name Parameter name
     * @param mixed $default Default to return if parameter isn't set
     * @param bool $nonempty Return $default if parameter is set but
empty()
     * @return mixed
     */
    public function param($name, $default = null, $nonempty = false)
    {
        if (!isset($this->access[$name])) return $default;
        $value = $this->applyfilter($this->access[$name]);
        if ($nonempty && empty($value)) return $default;
        return $value;
    }

    /**
     * Sets a parameter
     *
     * @param string $name Parameter name
     * @param mixed $value Value to set
     */
    public function set($name, $value)
    {
        $this->access[$name] = $value;
    }

    /**
     * Get a reference to a request parameter
     *
     * This avoids copying data in memory, when the parameter is not
set it will be created
     * and intialized with the given $default value before a reference
is returned
     *
     * @param string $name Parameter name
     * @param mixed $default If parameter is not set, initialize with
this value
```

```
     * @param bool $nonempty Init with $default if parameter is set but
empty()
     * @return mixed (reference)
     */
    public function &ref($name, $default = '', $nonempty = false)
    {
        if (!isset($this->access[$name]) || ($nonempty &&
empty($this->access[$name]))) {
            $this->set($name, $default);
        }

        return $this->access[$name];
    }

    /**
     * Access a request parameter as int
     *
     * @param string $name Parameter name
     * @param int $default Default to return if parameter isn't set or
is an array
     * @param bool $nonempty Return $default if parameter is set but
empty()
     * @return int
     */
    public function int($name, $default = 0, $nonempty = false)
    {
        if (!isset($this->access[$name])) return $default;
        if (is_array($this->access[$name])) return $default;
        $value = $this->applyfilter($this->access[$name]);
        if ($value === '') return $default;
        if ($nonempty && empty($value)) return $default;

        return (int)$value;
    }

    /**
     * Access a request parameter as string
     *
     * @param string $name Parameter name
     * @param string $default Default to return if parameter isn't set
or is an array
     * @param bool $nonempty Return $default if parameter is set but
empty()
     * @return string
     */
    public function str($name, $default = '', $nonempty = false)
    {
        if (!isset($this->access[$name])) return $default;
        if (is_array($this->access[$name])) return $default;
        $value = $this->applyfilter($this->access[$name]);
        if ($nonempty && empty($value)) return $default;
```

```
        return (string)$value;
    }

    /**
     * Access a request parameter and make sure it is has a valid value
     *
     * Please note that comparisons to the valid values are not done
typesafe (request vars
     * are always strings) however the function will return the correct
type from the $valids
     * array when an match was found.
     *
     * @param string $name Parameter name
     * @param array $valids Array of valid values
     * @param mixed $default Default to return if parameter isn't set
or not valid
     * @return null|mixed
     */
    public function valid($name, $valids, $default = null)
    {
        if (!isset($this->access[$name])) return $default;
        if (is_array($this->access[$name])) return $default; // we
don't allow arrays
        $value = $this->applyfilter($this->access[$name]);
        $found = array_search($value, $valids);
        if ($found !== false) return $valids[$found]; // return the
valid value for type safety
        return $default;
    }

    /**
     * Access a request parameter as bool
     *
     * Note: $nonempty is here for interface consistency and makes not
much sense for booleans
     *
     * @param string $name Parameter name
     * @param mixed $default Default to return if parameter isn't set
     * @param bool $nonempty Return $default if parameter is set but
empty()
     * @return bool
     */
    public function bool($name, $default = false, $nonempty = false)
    {
        if (!isset($this->access[$name])) return $default;
        if (is_array($this->access[$name])) return $default;
        $value = $this->applyfilter($this->access[$name]);
        if ($value === '') return $default;
        if ($nonempty && empty($value)) return $default;
```

```
        return (bool)$value;
    }

    /**
     * Access a request parameter as array
     *
     * @param string $name Parameter name
     * @param mixed $default Default to return if parameter isn't set
     * @param bool $nonempty Return $default if parameter is set but
empty()
     * @return array
     */
    public function arr($name, $default = array(), $nonempty = false)
    {
        if (!isset($this->access[$name])) return $default;
        if (!is_array($this->access[$name])) return $default;
        if ($nonempty && empty($this->access[$name])) return $default;

        return (array)$this->access[$name];
    }

    /**
     * Create a simple key from an array key
     *
     * This is useful to access keys where the information is given as
an array key or as a single array value.
     * For example when the information was submitted as the name of a
submit button.
     *
     * This function directly changes the access array.
     *
     * Eg. $_REQUEST['do']['save']='Speichern' becomes $_REQUEST['do']
= 'save'
     *
     * This function returns the $INPUT object itself for easy chaining
     *
     * @param string $name
     * @return Input
     */
    public function extract($name)
    {
        if (!isset($this->access[$name])) return $this;
        if (!is_array($this->access[$name])) return $this;
        $keys = array_keys($this->access[$name]);
        if (!$keys) {
            // this was an empty array
            $this->remove($name);
            return $this;
        }
        // get the first key
        $value = array_shift($keys);
```

```
        if ($value === 0) {
            // we had a numeric array, assume the value is not in the
key
            $value = array_shift($this->access[$name]);
        }

        $this->set($name, $value);
        return $this;
    }
}
```

«Подробности»

From:
https://wwoss.ru/ - **worldwide open-source software**

Permanent link:
https://wwoss.ru/doku.php?id=wiki:xref:dokuwiki:inc:input:input.php

Last update: **2025/01/03 18:14**