RequestHandler

What is RequestHandler?

`RH` (alias for `RequestHandler`) provides secure, centralized access to HTTP request data. It automatically sanitizes input using `vtlib_purify()` and applies type-safe filters from `F::` enum. This replaces direct access to superglobals like `$_REQUEST`, `$_GET`, `$_POST`, etc.
Our goal with this system is to prevent vulnerabilities ex. SQL Injection, XSS, File Inclusion, ecc.

How filtering works

RequestHandler applies a two-level filtering process:

- **Whitelisted keys** (defined in `config/request.config.php`): These keys are automatically filtered using the filter specified in the configuration during initialization. The configured filter is applied first, and if you specify an additional filter when accessing the value, that filter is applied to the already-sanitized value.
- **Non-whitelisted keys**: All other keys pass through `vtlib_purify()` for HTML sanitization by default. If you specify an explicit filter (e.g., `F::int`, `F::email`), it is applied to the purified value.

```
// ex. index.php?module=Accounts&action=CustomAction&custom=123&another=TEST


// In config: 'module' => F::mod
$module = RH::r('module');               // Uses F::mod filter from config
$customParam = RH::r('custom', F::int);  // Not in config: vtlib_purify() + F::int
$anotherParam = RH::r('another');        // Not in config: only vtlib_purify()
```

This approach ensures:

- Unknown parameters are still sanitized to prevent XSS attacks
- Explicit filters always provide additional type safety
- Explicitly mapping more parameters enhances input security in the application.

□ DO

```
// □ Always access superglobals through RH
$module = RH::r('module', F::mod);
$id = RH::r('id', F::int);
```

```
// ✅ Always specify appropriate filters
$email = RH::r('email', F::email);
$age = RH::r('age', F::int);
$status = RH::r('status', F::enum(['active', 'inactive']));

// ✅ Check for null values when data is required
$userId = RH::r('user_id', F::int);
if ($userId === null) {
    throw new InvalidArgumentException('User ID is required');
}

// ✅ Use POST for operations that are sensitive or that modify the CRM state (e.g.: sending an email, saving a record, ...)
RH::ensureMethod('POST');
```

❌ DON'Ts

> **Avoid these practices:** They bypass security, are redundant and can introduce vulnerabilities.

- **Never access superglobals directly:** `$_REQUEST`, `$_GET`, `$_POST`, `$_COOKIE`, `$_FILES`, `$_SERVER`
- **Don't use manual sanitization:** Avoid `vtlib_purify()` calls, RH handles this centrally
- **Don't skip filters:** Calling `RH::r('age')` without a filter may return unvalidated strings
- **Don't use raw methods without reason:** Prefer `RH::r(..., F::...)` over `RH::r_raw(...)`
- **Don't ignore security:** Always validate HTTP methods for sensitive operations or action that modify the CRM state
- **Don't assume data types:** Always use appropriate filters and null checks

```
// ❌ Direct superglobal access
$userId = $_REQUEST['user_id'];
$email = $_POST['email'];
$page = intval($_GET['page']);

// ❌ Manual sanitization
$name = vtlib_purify($_REQUEST['name']);
$description = vtlib_purify($_POST['description']);

// ❌ No filter applied
```

```
$age = RH::r('age');  // Returns string "25" instead of int 25
if ($age > 18) {      // String comparison may not work as expected
    // ...
}

// ❌ Using raw without justification
$data = RH::r_raw('data');  // Bypasses sanitization
echo $data;                 // Potential XSS vulnerability

// ❌ No null check for required data
$recordId = RH::r('record', F::int);
$record = getRecord($recordId);  // May fail if $recordId is null

// ❌ Accepting any HTTP method for sensitive operation or action that modify the CRM state
// No method check here - vulnerable to GET-based attacks
$settings = RH::r('settings', F::json);
saveSettings($settings);

// ❌ Type assumption without validation
$ids = RH::r('ids');     // Assuming it's an array
foreach ($ids as $id) {  // Fatal error if $ids is not an array
    // ...
}
```

Main methods

Method	Description
`RH::r($key, $filter)`	Access $_REQUEST with filter (optional)
`RH::g($key, $filter)`	Access $_GET with filter (optional)
`RH::p($key, $filter)`	Access $_POST with filter (optional)
`RH::c($key, $filter)`	Access $_COOKIE with filter (optional)
`RH::s($key, $filter)`	Access $_SERVER with filter (optional)
`RH::f($key)`	Access uploaded files (PSR-7)
`RH::r_mod()`	Module name from $_REQUEST
`RH::r_action()`	Action name from $_REQUEST
`RH::r_record()`	Record ID from $_REQUEST
`RH::r_has($key)`	Verify key existence (sanitized)

Method	Description
`RH::r_has_raw($key)`	Verify key existence (raw)
`RH::r_all()`	All parameters sanitized
`RH::r_all_raw()`	All parameters raw (unsanitized)
`RH::ensureMethod($method)`	Force HTTP method (405 if different)
`RH::isFromMobile()`	Check if request is from mobile app
`RH::isFromPortal()`	Check if request is from portal

Basic filters

- **Null input always returns null** - All filters preserve null values
- **Filters for primitive types (int, float, bool)** - Convert invalid input to default values (0, 0.0, false)
- **Validation filters (email, url, ip, enum, regex, datetime)** - Return null for invalid input
- **Sanitization filters (str, html, alpha, alphanum, mod, action, etc.)** - Remove invalid characters and return cleaned string
- **Always check for null** - Use null coalescing operator `??` for required values

Filter	Use case	Valid input → Output	Invalid input → Output
`F::int`	IDs, numbers	`"42"` → `42`	`"abc"` → `0` `null` → `null`
`F::float`	Decimal numbers	`"3.14"` → `3.14`	`"abc"` → `0.0` `null` → `null`
`F::str`	General strings (**no HTML; default max length: 1000 characters**)	`"Hello"` → `"Hello"`	`"<b>abc</b>"` → `"abc"` (tags stripped) `"Very long text..."` → Truncated to 1000 chars by default `null` → `null`
`F::str(max: 50)`	Limited strings (**no HTML**)	`"Short text"` → `"Short text"`	`"Very long text..."` → Truncated to 50 chars `null` → `null`
`F::substr(max: 50)`	Substring (**keeps HTML**, may result in invalid HTML)	`"<b>text</b>"` → `"<b>text</b>"`	Long string → Truncated (HTML preserved) `null` → `null`
`F::html`	HTML strings (purified)	`"<b>bold</b>"` → `"<b>bold</b>"`	`"<script>alert(0)</script>"` → `""` (XSS removed) `null` → `null`
`F::bool`	Boolean flags	`"1"`, `"true"`, `"yes"` → `true`	`"0"`, `"false"`, `"no"` → `false` `null` → `null`

Filter	Use case	Valid input → Output	Invalid input → Output

`F::email`	Email addresses	`"user@example.com"` → `"user@example.com"`	`"invalid-email"` → `null`
			`"user@"` → `null`
			`null` → `null`

`F::url`	URLs	`"https://example.com"` → `"https://example.com"`	`"invalid-url"` → `null`
			`"htp://exa"` → `null`
			`null` → `null`

`F::ip`	IP addresses	`"192.168.1.1"` → `"192.168.1.1"`	`"999.999.999.999"` → `null`
			`"invalid-ip"` → `null`
			`null` → `null`

`F::json`	JSON data	`'{"key":"value"}'` → `["key" => "value"]`	`'invalid json'` → `null`
			`'{broken'` → `null`
			`null` → `null`

Specific filters

Filter	Use case	Valid input → Output	Invalid input → Output
`F::mod`	Module names	`"Accounts"` → unchanged	`"Acc<script>"` → `"Accscript"` (special chars removed) `null` → `null`
`F::action`	Action names	`"Save"`, `"Detail.View"` → unchanged	`"Act!on@"` → `"Acton"` (special chars removed) `null` → `null`
`F::field`	Field names	`"firstname"` → unchanged	`"field-name!"` → `"fieldname"` (special chars removed) `null` → `null`
`F::recfields`	Record/field IDs	`"123@-456"`, `"1x2,3|4"` → unchanged	`"123abc!@#"` → `"123@"` (invalid chars removed) `null` → `null`
`F::path`	File paths (**dots allowed; use checkFileAccess to ensure secure file access**)	`"modules/Accounts"` → unchanged	`"path@with#special$chars"` → `"pathwithspecialchars"` (keeps valid chars) `null` → `null`
`F::subpath`	Sub paths (**no dots; use checkFileAccess to ensure secure file access**)	`"modules/Accounts"` → unchanged	`"subpath.with.dots"` → `"subpathwithdots"` (dots removed) `null` → `null`
`F::alpha`	Alphabetic only	`"abcABC_"` → unchanged	`"abc123!@#"` → `"abc"` (numbers/symbols removed) `null` → `null`
`F::alphanum`	Alphanumeric only	`"abcABC123_"` → unchanged	`"abc123!@#"` → `"abc123"` (symbols removed) `null` → `null`

Filter	Use case	Valid input → Output	Invalid input → Output
`F::enum([...])`	Whitelist values	`"active"` (if in list) → unchanged	`"invalid"` (not in list) → `null` `null` → `null`
`F::regex('/pattern/')`	Pattern matching	`"ABC123"` (matches) → unchanged	`"invalid"` (no match) → `null` `null` → `null`

`F::datetime('Y-m-d')`	Date/time parsing	`"2026-01-08"` → unchanged	`"invalid-date"` → `null`
			`"2026-99-99"` → `null`
			`null` → `null`

`F::sep(',', 'intval')`	Split and transform	`"1,2,3"` → `[1, 2, 3]`	Non-string → `null` `null` → `null`
`F::jsfunc`	Safe JS functions	`"closePopup"` (if whitelisted) → unchanged	`"evilFunction"` (not whitelisted) → `null` `null` → `null`

Advanced filters

```
// Enum filter - restrict to specific values
$status = RH::r('status', F::enum(['active', 'inactive', 'pending']));

// Regex filter - custom pattern validation
$code = RH::r('code', F::regex('/^[A-Z]{3}\d{3}$/'));

// DateTime filter - parse date strings
$date = RH::r('date', F::datetime('Y-m-d'));

// Separator filter - split strings into arrays
$ids = RH::r('ids', F::sep(',', 'intval')); // "1,2,3" -> [1,2,3]

// Array filter - validate nested arrays
$user = RH::r('user', F::array([
    'name' => F::str(max: 100),
    'email' => F::email,
    'age' => F::int,
]));

// Custom callable filter
$customCode = RH::r('code', function($value) {
    if (!is_string($value)) return null;
    $value = strtoupper(trim($value));
    return preg_match('/^[A-Z0-9]{6}$/', $value) ? $value : null;
});
```

Examples

Simple form data

```
$name = RH::p('name', F::str(max: 100));
$email = RH::p('email', F::email);
$age = RH::p('age', F::int);

if ($name === null || $email === null) {
    throw new InvalidArgumentException('Name and email are required');
}
```

List page with pagination

```
$page = RH::g('page', F::int) ?? 1;
$limit = RH::g('limit', F::int) ?? 20;
$sort = RH::g('sort', F::enum(['name', 'date', 'id'])) ?? 'name';
$direction = RH::g('dir', F::enum(['asc', 'desc'])) ?? 'asc';
$query = RH::g('q', F::str(max: 255));
```

Complex form with nested arrays

```
$userData = RH::p('user', F::array([
    'personal' => F::array([
        'firstname' => F::str(max: 50),
        'lastname' => F::str(max: 50),
        'email' => F::email,
        'phone' => F::str(max: 20),
    ]),
    'preferences' => F::array([
        'language' => F::enum(['en', 'it', 'es']),
        'timezone' => F::str(max: 50),
    ]),
    'addresses' => F::array([
        'shipping' => F::array([
            'street' => F::str(max: 255),
            'city' => F::str(max: 100),
            'country' => F::str(max: 2),
        ]),
```

```
    ]),
]));
```

File upload

```
$uploadedFile = RH::f('document');

if ($uploadedFile && $uploadedFile->getError() === UPLOAD_ERR_OK) {
    $filename = $uploadedFile->getClientFilename();
    $mimeType = $uploadedFile->getClientMediaType();
    $size = $uploadedFile->getSize();

    // ...
} else {
    // Handle upload error
    $error = $uploadedFile ? $uploadedFile->getError() : 'No file uploaded';
}
```

Advanced Usage

Temporary request context

> **Push/pop operations should be used only for testing or very specific scenarios.**

```
// Save current state and temporarily override
RH::push_r(['module' => 'TestModule', 'action' => 'TestAction']);

// Do something with temporary data
$module = RH::r_mod(); // Returns 'TestModule'

// Restore original state
RH::pop_r();
```

Configuration

The RequestHandler uses a configuration file to control request sanitization behavior and define safe parameters. Configuration is defined in `config/request.config.php` and can be extended via `config/request.config.override.php`.

`preserve_original_request`

```
'preserve_original_request' => false,  // default
```

Controls whether all original request parameters are preserved in the `$_REQUEST` superglobal:

- **`false` (default, recommended):** Only keys listed in `safe_keys` are preserved in `$_REQUEST`. This provides maximum security by removing any unexpected parameters that could be malicious.
- **`true`:** All original parameters remain in `$_REQUEST`. Use only if you need backward compatibility with legacy code that accesses parameters not in the whitelist.

> **Security note:** When `preserve_original_request` is `false`, the `$_REQUEST` array is cleaned to contain only whitelisted keys. This prevents injection of unexpected parameters but does NOT bypass RH's access control - you should still use RH methods for all access.

`safe_keys`

```
'safe_keys' => [
    'module' => F::mod,
    'action' => F::action,
    'record' => F::int,
    'id' => F::int,
    'page' => F::int,
    'limit' => F::int,
    // ... add commonly used keys here
],
```

Defines a whitelist of parameter keys that are:

- **Preserved during request cleanup:** When `preserve_original_request` is `false`, only these keys remain in `$_REQUEST`
- **Pre-sanitized automatically:** Each key is associated with a filter (e.g., `F::mod`, `F::int`) that is applied during initialization

`safe_js_functions`

```
'safe_js_functions' => [
    'closePopup',
    'LPOP.create',
    'parent.ActionTaskScript.addStaticRelatedRecord',
    // ... other trusted JavaScript function names
],
```

Lists JavaScript function names that are allowed to be passed as request parameters (e.g., for callbacks). This prevents XSS attacks by restricting which functions can be executed client-side.

- Used by the `F::jsfunc` filter
- Only listed functions are considered safe
- Reject any function not in the whitelist

Creating custom configuration

To add your own safe keys without modifying the core configuration, create `config/request.config.override.php`:

```
// config/request.config.override.php
return [
    'safe_keys' => [
        // Your custom keys
        'custom_param' => F::str(max: 100),
        'my_record_id' => F::int,
    ],
    'safe_js_functions' => [
        // Your custom callback functions
        'MyApp.customCallback',
    ],
];
```

The override configuration is automatically merged with the default configuration.

> **Migration to version 26.04**
> During the upgrade to version 26.04, the system automatically scans for direct `$_REQUEST` superglobal usage. Any detected parameters are automatically added to `config/request.config.override.php` with the `F::html` filter to maintain backward compatibility and prevent breaking existing customizations.
>
> **All core code has been refactored to use RH methods exclusively** instead of direct superglobal access.

Revision #6
Created 2026-01-23 15:30:21 UTC by m.maporti
Updated 2026-04-20 14:42:31 UTC by manuel.tagliapietra