Form Validator - Complete Documentation

Afficher l'image Afficher l'image Afficher l'image

A powerful, lightweight jQuery-based form validation library with comprehensive error handling and user-friendly feedback.

Installation • Quick Start • API Reference • Examples

Table of Contents

- Overview
- Features
- Installation
- Quick Start
- API Reference
  - Error Message Functions
    - `smallError()`
    - `validatorErrorField()`
    - `createSmallErrorMessage()`
  - DOM Management Functions
    - `addErrorMessageFieldDom()`
    - `handleErrorsManyForm()`
    - `clearErrorInput()`
  - Utility Functions
    - `getInputPatternRegex()`
    - `getAttr()`
    - `stringToRegex()`
- Type Definitions
- Examples
- Best Practices
- Support & Contact

Overview

Form Validator is a comprehensive validation library designed to simplify form error handling in web applications. Built with jQuery compatibility, it seamlessly integrates with modern frameworks like React and provides Bootstrap-compatible styling out of the box.

Whether you're building traditional multi-page forms or single-page applications with client-side validation, Form Validator provides the tools you need to deliver professional, user-friendly error feedback.

Why Form Validator?

- ✓ **Simple API**: Intuitive functions that handle complex validation scenarios
- ✓ **Framework Agnostic**: Works with vanilla JavaScript, jQuery, React, and Vue
- ✓ **Bootstrap Ready**: Default Bootstrap 5 styling with full customization support
- ✓ **Type Safe**: Full TypeScript support with comprehensive type definitions
- ✓ **Lightweight**: Minimal dependencies (jQuery only)
- ✓ **Production Ready**: Battle-tested error handling and edge case management

---

Features

- 🎯 Single and multiple error messages per field
- 🎨 Customizable error container styling
- 🔄 Real-time error clearing and updating
- 📝 Regex pattern extraction and validation
- 🛡️ Type-safe with full TypeScript support
- 🔗 Nested field error handling (dot notation)
- 📦 Automatic DOM element generation
- ⛵ Bootstrap validation feedback integration

Installation

Via NPM

bash

```
npm install form-validator
```

Via Yarn

bash

```
yarn add form-validator
```

Manual Installation

html

```
<script src="path/to/form-validator.js"></script>
```

Requirements

- **jQuery** 3.0 or higher
- **Bootstrap** 5.x (optional, for default styling)

Quick Start

Basic Setup

javascript

```javascript
import {
  addErrorMessageFieldDom,
  handleErrorsManyForm,
  clearErrorInput
} from 'form-validator';

// Display error for a single field
const emailInput = $('#user_email');
addErrorMessageFieldDom(emailInput, [
  'This field is required.',
  'Must be a valid email address.'
]);

// Clear errors
clearErrorInput(emailInput);

// Handle multiple form errors at once
handleErrorsManyForm('user', 'user_form', {
  email: ['Invalid email format'],
  password: ['Password too short'],
  'address.city': ['City is required']
});
```

React Integration

jsx

```
import React, { useRef } from 'react';
import { addErrorMessageFieldDom, clearErrorInput } from 'form-validator';

export function LoginForm() {
  const emailRef = useRef(null);

  const handleSubmit = (e) => {
    e.preventDefault();

    if (!emailRef.current?.value) {
      addErrorMessageFieldDom(jQuery(emailRef.current), [
        'Email is required'
      ]);
    } else {
      clearErrorInput(jQuery(emailRef.current));
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input ref={emailRef} id="user_email" type="email" />
      <button type="submit">Login</button>
    </form>
  );
}
```

API Reference

Error Message Functions

`smallError()`

Generates a semantic HTML `<small>` tag for displaying individual error messages.

Signature:

typescript

```
function smallError(
  message_error: string,
  className: string,
  id: string,
  key?: number
): string
```

Parameters:

Parameter	Type	Description
`message_error`	`string`	The error message text to display
`className`	`string`	Space-separated CSS classes to apply
`id`	`string`	Unique identifier for the element
`key`	`number?`	Optional numeric key for data attribute

Returns: HTML string representing the `<small>` element

Example:


javascript

```
const errorHtml = smallError(
  'Email format is invalid',
  'text-danger fw-bold',
  'error-email-1',
  1
);
// Output: <small id="error-email-1" class="text-danger fw-bold" data-key="1">
//           Email format is invalid
//         </small>
```

Use Cases:

- Creating individual error message elements
- Building custom error containers
- Generating semantic HTML for accessibility

`validatorErrorField()`

Generates a complete block of formatted error messages with optional separators.

Signature:


typescript

```
function validatorErrorField(
  validate_error_field: ValidatorErrorFieldProps = {
    messageerror: ' ',
    classnameerror: ["fw-bold", "text-danger", "mt-2"],
    id: `error-field-${Date.now()}`,
    separator_join: "<br/><hr/>"
  }
): string
```

Parameters (ValidatorErrorFieldProps):

Property	Type	Description
messageerror	string | string[]	Single or multiple error messages
classnameerror	string[]?	Array of CSS classes
id	string	Base ID for elements
separator_join	string	HTML to join multiple messages

Returns: HTML string with all error messages

Example:

javascript

```
const errors = validatorErrorField({
  messageerror: [
    'Field is required',
    'Must be at least 8 characters',
    'Must contain uppercase letter'
  ],
  classnameerror: ['text-danger', 'small', 'fw-bold'],
  id: 'password-errors',
  separator_join: '<br/>'
});
```

Use Cases:

- Grouping multiple errors for a field
- Custom error formatting and styling
- Building error collections

`createSmallErrorMessage()`

Creates or retrieves an error message element with automatic DOM detection and element reuse.

Signature:


typescript

```
function createSmallErrorMessage(
  fieldInputID: string,
  errorMessage: string,
  keyError: number | string
): JQuery<HTMLElement>
```

Parameters:

Parameter	Type	Description
`fieldInputID`	`string`	ID of the form field
`errorMessage`	`string`	The error message text
`keyError`	`number | string`	Unique key for this specific error

Returns: jQuery object of the error message element

Example:


javascript

```
const errorElement = createSmallErrorMessage(
  'user_password',
  'Password must contain uppercase letter',
  0
);

// Append to DOM
jQuery('#password-container').append(errorElement);
```

Features:

- Automatic element reuse if already exists
- Unique ID generation
- Data attributes for easy targeting

DOM Management Functions

`addErrorMessageFieldDom()`

Appends or updates validation error messages for a form field in the DOM with automatic styling.

Signature:


typescript

```
function addErrorMessageFieldDom(
  elmtfield: JQuery<HTMLElement>,
  errormessagefield?: string[],
  className_container_ErrorMessage: string = "border border-3 border-light"
): void
```

Parameters:

Parameter	Type	Description
`elmtfield`	`JQuery<HTMLElement>`	jQuery reference to the form field
`errormessagefield`	`string[]?`	Array of error messages (empty array clears errors)
`className_container_ErrorMessage`	`string`	CSS classes for error container

Returns: `void`

Example:

javascript

```
// Display errors
const emailField = jQuery('#user_email');
addErrorMessageFieldDom(emailField, [
  'This field is required.',
  'Must be a valid email address.'
]);

// Clear errors
addErrorMessageFieldDom(emailField, []);
```

Behavior:

- Adds `is-invalid` class to field
- Creates container div below the field
- Removes errors when passed empty array
- Handles multiple messages with separators

`handleErrorsManyForm()`

Manages validation errors for an entire form, supporting nested field names.

Signature:

typescript

```
function handleErrorsManyForm(
  formName: string,
  formId: string,
  errors: Record<string, string[]>
): void
```

Parameters:

Parameter	Type	Description
formName	string	Prefix used in field IDs (e.g., "user")
formId	string	DOM ID of the form element
errors	Record<string, string[]>	Object with field names as keys, error arrays as values

Returns: `void`

Example:

javascript

```
// Display multiple form errors
handleErrorsManyForm('user', 'user_form', {
  email: ['Email is required', 'Invalid format'],
  password: ['Password too short'],
  'address.city': ['City is required'],
  'address.country': ['Country is required']
});

// Clear all errors (pass empty object)
handleErrorsManyForm('user', 'user_form', {});
```

Field Naming Convention:

```
Form field ID: user_email          → Error key: email
Form field ID: user_address_city   → Error key: address.city
Form field ID: user_phone_country  → Error key: phone.country
```

Use Cases:

- Server-side form validation responses
- Multi-step form validation
- Comprehensive error clearing

`clearErrorInput()`

Removes all error messages and validation styling from a specific form field.

Signature:


typescript

```
function clearErrorInput(
  inputFieldJQuery: JQuery<HTMLElement>
): void
```

Parameters:

Parameter	Type	Description
`inputFieldJQuery`	`JQuery<HTMLElement>`	jQuery reference to the form field

Returns: `void`

Example:


javascript

```
// Clear errors for email field
clearErrorInput(jQuery('#user_email'));

// Clear errors on input event
jQuery('#user_email').on('input', function() {
  clearErrorInput(jQuery(this));
});
```

Operations:

- Removes `is-invalid` class
- Deletes error container from DOM
- Handles edge cases (missing IDs, etc.)

Utility Functions

`getInputPatternRegex()`

Extracts and converts the `pattern` attribute from an input element to a JavaScript RegExp object.

Signature:


typescript

```
function getInputPatternRegex(
  children: HTMLElement | JQuery<HTMLElement>,
  formParentName: string,
  flag: string = 'i'
): RegExp | undefined
```

Parameters:

Parameter	Type	Description
children	HTMLElement | JQuery<HTMLElement>	The input or textarea element
formParentName	string	Form name for logging context
flag	string	Regex flags: g, i, m, u, y, s

Returns: RegExp object or undefined if pattern not found

Example:

javascript

```
// HTML
// <input id="email" name="email" pattern="^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$">

const input = document.querySelector('#email');
const regex = getInputPatternRegex(input, 'LoginForm', 'i');

if (regex?.test('user@example.com')) {
  console.log('✅ Valid email!');
} else {
  console.log('❌ Invalid email!');
}
```

React Integration:

jsx

```
import React, { useRef } from 'react';
import { getInputPatternRegex } from 'form-validator';

export function EmailValidator() {
  const inputRef = useRef(null);

  const validate = () => {
    if (inputRef.current) {
      const regex = getInputPatternRegex(inputRef.current, 'MyForm', 'i');
      const isValid = regex?.test(inputRef.current.value);
      console.log(isValid ? '✅ Valid' : '❌ Invalid');
    }
  };

  return (
    <>
      <input
        ref={inputRef}
        type="email"
        pattern="^[^\s@]+@[^\s@]+\.[^\s@]+$"
      />
      <button onClick={validate}>Validate</button>
    </>
  );
}
```

`getAttr()`

Safely retrieves and optionally parses attribute values from DOM elements.

Signature:

typescript

```
function getAttr<T = unknown>(
  element: HTMLElement | null | undefined | JQuery<HTMLElement>,
  name: string,
  defaults: unknown = null,
  toJson: boolean = false
): T
```

Parameters:

Parameter	Type	Description
element	HTMLElement | JQuery<HTMLElement>?	DOM element to query
name	string	Attribute name to retrieve
defaults	unknown	Default value if not found
toJson	boolean	Parse JSON if true

Returns: Attribute value or default

Example:


javascript

```javascript
// Simple attribute
const placeholder = getAttr(
  jQuery('#email'),
  'placeholder',
  'Enter email'
);

// JSON attribute
const config = getAttr(
  jQuery('#form'),
  'data-config',
  { theme: 'light' },
  true
);
// HTML: <form id="form" data-config='{"theme":"dark"}'>
// Result: { theme: "dark" }
```

Use Cases:

- Reading data attributes
- Parsing JSON configurations
- Safe attribute access with fallbacks

`stringToRegex()`

Converts a string representation into a RegExp object with optional flags.

Signature:


typescript

```
function stringToRegex(
  regexString: string | null | undefined,
  flags: FlagRegExp = 'iu'
): RegExp | undefined
```

Parameters:

Parameter	Type	Description
regexString	string?	String representation of regex pattern
flags	FlagRegExp	Regex flags (default: 'iu')

Returns: RegExp object or undefined

Example:

javascript

```
// Basic usage
const regex = stringToRegex('^[a-z0-9]+@[a-z0-9]+\\.[a-z]{2,}$', 'gi');

if (regex?.test('USER@EXAMPLE.COM')) {
  console.log('Valid email format');
}

// With fallback
const pattern = getUserPattern() || '^[a-zA-Z]+$';
const userRegex = stringToRegex(pattern, 'i');
```

Type Definitions

`ValidatorErrorFieldProps`

typescript

```
interface ValidatorErrorFieldProps {
  messageerror: string | string[];
  classnameerror?: string[];
  id: string;
  separator_join: string;
}
```

`FlagRegExp`

typescript

```
type FlagRegExp =
  | 'g' | 'i' | 'm' | 'u' | 'y' | 's'
  | 'gi' | 'iu' | 'gim'
  | [other flag combinations];
```

`MediaType`

typescript

```
type MediaType = "video" | "document" | "image";
```

`FormInputType`

typescript

```
type FormInputType =
  | "fqdn" | "file" | "radio" | "checkbox" | "number"
  | "text" | "email" | "password" | "url" | "select"
  | "textarea" | "date" | "tel";
```

`HTMLFormChildrenElement`

typescript

```
type HTMLFormChildrenElement =
  | HTMLInputElement
  | HTMLTextAreaElement
  | HTMLSelectElement;
```

`DataInput`

typescript

```
type DataInput =
  | string
  | string[]
  | number
  | null
  | undefined
  | File
  | FileList
  | Date;
```

Examples

Example 1: Real-Time Form Validation

javascript

```
import {
  addErrorMessageFieldDom,
  clearErrorInput,
  getInputPatternRegex
} from 'form-validator';

const emailInput = jQuery('#email');
const passwordInput = jQuery('#password');

// Real-time email validation
emailInput.on('blur', function() {
  const value = jQuery(this).val();
  const errors = [];

  if (!value) {
    errors.push('Email is required');
  } else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(value)) {
    errors.push('Invalid email format');
  }

  if (errors.length > 0) {
    addErrorMessageFieldDom(emailInput, errors);
  } else {
    clearErrorInput(emailInput);
  }
});

// Real-time password validation
passwordInput.on('input', function() {
  const value = jQuery(this).val();
  const errors = [];

  if (value.length < 8) {
    errors.push('At least 8 characters required');
  }
  if (!/[A-Z]/.test(value)) {
    errors.push('Must contain uppercase letter');
  }
  if (!/[0-9]/.test(value)) {
    errors.push('Must contain number');
  }

  if (errors.length > 0) {
    addErrorMessageFieldDom(passwordInput, errors);
  } else {
```

```
    clearErrorInput(passwordInput);
  }
});
```

Example 2: Server-Side Validation Response

javascript

```
import { handleErrorsManyForm } from 'form-validator';

// Simulating API response
fetch('/api/register', {
  method: 'POST',
  body: JSON.stringify(formData)
})
.then(res => res.json())
.then(data => {
  if (data.errors) {
    // Handle validation errors from server
    handleErrorsManyForm('registration', 'registration_form', {
      username: data.errors.username || [],
      email: data.errors.email || [],
      password: data.errors.password || [],
      'profile.bio': data.errors.profile_bio || []
    });
  } else {
    // Clear all errors on success
    handleErrorsManyForm('registration', 'registration_form', {});
    console.log('Form submitted successfully!');
  }
});
```

Example 3: React Component with Validation

jsx

```
import React, { useState, useRef } from 'react';
import {
  addErrorMessageFieldDom,
  clearErrorInput,
  getInputPatternRegex
} from 'form-validator';

export function RegistrationForm() {
  const [loading, setLoading] = useState(false);
  const emailRef = useRef(null);
  const passwordRef = useRef(null);
  const confirmRef = useRef(null);

  const validateField = (ref, validations) => {
    const errors = [];
    const value = ref.current?.value;

    for (const validation of validations) {
      if (!validation.rule(value)) {
        errors.push(validation.message);
      }
    }

    const $field = jQuery(ref.current);
    if (errors.length > 0) {
      addErrorMessageFieldDom($field, errors);
      return false;
    } else {
      clearErrorInput($field);
      return true;
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);

    const isEmailValid = validateField(emailRef, [
      {
        rule: (v) => !!v,
        message: 'Email is required'
      },
      {
        rule: (v) => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(v),
        message: 'Invalid email format'
```

```
  }
 ]);

 const isPasswordValid = validateField(passwordRef, [
  {
   rule: (v) => v?.length >= 8,
   message: 'At least 8 characters required'
  },
  {
   rule: (v) => /[A-Z]/.test(v),
   message: 'Must contain uppercase letter'
  },
  {
   rule: (v) => /[0-9]/.test(v),
   message: 'Must contain number'
  }
 ]);

 const doPasswordsMatch = validateField(confirmRef, [
  {
   rule: (v) => v === passwordRef.current?.value,
   message: 'Passwords do not match'
  }
 ]);

 if (isEmailValid && isPasswordValid && doPasswordsMatch) {
  try {
   const response = await fetch('/api/register', {
    method: 'POST',
    body: JSON.stringify({
     email: emailRef.current?.value,
     password: passwordRef.current?.value
    })
   });

   if (response.ok) {
    console.log('✅ Registration successful!');
   }
  } catch (error) {
   console.error('Registration failed:', error);
  }
 }

 setLoading(false);
};
```

```
 return (
  <form onSubmit={handleSubmit} id="registration_form">
   <div className="form-group mb-3">
    <label htmlFor="email">Email:</label>
    <input
     ref={emailRef}
     id="email"
     type="email"
     className="form-control"
     placeholder="Enter your email"
    />
   </div>

   <div className="form-group mb-3">
    <label htmlFor="password">Password:</label>
    <input
     ref={passwordRef}
     id="password"
     type="password"
     className="form-control"
     placeholder="Enter password"
    />
   </div>

   <div className="form-group mb-3">
    <label htmlFor="confirm">Confirm Password:</label>
    <input
     ref={confirmRef}
     id="confirm"
     type="password"
     className="form-control"
     placeholder="Confirm password"
    />
   </div>

   <button type="submit" disabled={loading} className="btn btn-primary">
    {loading ? 'Registering...' : 'Register'}
   </button>
  </form>
 );
}
```

Example 4: Custom Error Styling

javascript

```javascript
import { addErrorMessageFieldDom } from 'form-validator';

const field = jQuery('#username');

// Use custom styling
addErrorMessageFieldDom(
  field,
  ['Username is already taken', 'Minimum 4 characters required'],
  'bg-light-danger border border-2 border-danger rounded p-3'
);

// Custom classes example (with Bootstrap utilities)
addErrorMessageFieldDom(
  field,
  ['Username too short'],
  'alert alert-danger alert-dismissible fade show'
);
```

Best Practices

✔ Do

- **Clear errors on focus**: Remove error messages when user starts typing

javascript

```javascript
field.on('focus', () => clearErrorInput(jQuery(this)));
```

- **Validate on blur**: Perform validation when user leaves the field

javascript

```javascript
field.on('blur', () => validateAndShowErrors(jQuery(this)));
```

- **Use semantic HTML**: Always provide ID attributes to form fields


html

```
<input id="user_email" type="email" required>
```

- **Combine multiple validators**: Stack validations for comprehensive checks


javascript

```
const errors = [];
if (!value) errors.push('Required');
if (value.length < 8) errors.push('Too short');
if (!hasNumber) errors.push('Need number');
addErrorMessageFieldDom(field, errors);
```

- **Handle edge cases**: Always check for null/undefined


javascript

```
const value = field.val() || '';
if (!value.trim()) {
  // Handle empty field
}
```

✕ Don't

- **Don't forget field IDs**: All fields must have unique IDs


javascript

```
// ✕ Bad
<input type="email">

// ✓ Good
<input id="user_email" type="email">
```

- **Don't mix validation approaches**: Use Form Validator consistently


javascript

```
// ✕ Inconsistent
addErrorMessageFieldDom(field1, errors);
field2.classList.add('error'); // Don't mix approaches

// ✓ Consistent
addErrorMessageFieldDom(field1, errors);
addErrorMessageFieldDom(field2, errors);
```

- **Don't ignore TypeScript warnings**: Use proper types

typescript

```
// ✕ Loose typing
const result: any = getAttr(element, 'data-config');

// ✓ Type safe
const result: Record<string, unknown> = getAttr(
  element,
  'data-config',
  {},
  true
);
```

Common Issues & Solutions

Issue: jQuery not found

Problem:

```
Uncaught ReferenceError: jQuery is not defined
```

Solution: Ensure jQuery is loaded before Form Validator:

html

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script src="form-validator.js"></script>
```

Issue: Errors not displaying

Problem: Error container created but not visible

Solution: Check field ID exists and is unique:


javascript

```
console.log(jQuery('#field_id').attr('id')); // Should print the ID
```

Issue: Multiple error messages duplicating

Problem: Old errors not cleared before adding new ones

Solution: Clear errors first:


javascript

```
// ✕ May duplicate
addErrorMessageFieldDom(field, newErrors);

// ✓ Correct approach
clearErrorInput(field);
addErrorMessageFieldDom(field, newErrors);
```

Browser Support

Browser	Support
Chrome	✓ Latest 2 versions
Firefox	✓ Latest 2 versions
Safari	✓ Latest 2 versions
Edge	✓ Latest 2 versions
IE 11	⚠ Limited (requires polyfills)

Contributing

We welcome contributions! Please follow these steps:

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/amazing-feature`)
3. Commit your changes (`git commit -m 'Add amazing feature'`)
4. Push to the branch (`git push origin feature/amazing-feature`)
5. Open a Pull Request

Support & Contact

Created by: AGBOKOUDJO Franck

📧 **Email:** internationaleswebservices@gmail.com

📱 **Phone:** +229 0167 25 18 86

🔗 **LinkedIn:** INTERNATIONALES WEB APPS & SERVICES

💀 **GitHub:** @Agbokoudjo

🏢 **Company:** INTERNATIONALES WEB APPS & SERVICES

License

Made with ❤ for the developer community

Last Updated: November 2024
Version: 1.0.0