API Report File for "@angular/forms"

> *Do not edit this file. It is a report generated by [API Extractor](https://api-extractor.com/).*

```ts
import { AfterViewInit } from '@angular/core';
import { ChangeDetectorRef } from '@angular/core';
import { ElementRef } from '@angular/core';
import { EventEmitter } from '@angular/core';
import * as i0 from '@angular/core';
import { InjectionToken } from '@angular/core';
import { Injector } from '@angular/core';
import { ModuleWithProviders } from '@angular/core';
import { Observable } from 'rxjs';
import { OnChanges } from '@angular/core';
import { OnDestroy } from '@angular/core';
import { OnInit } from '@angular/core';
import { Renderer2 } from '@angular/core';
import { SimpleChanges } from '@angular/core';
import { Version } from '@angular/core';

// @public
export abstract class AbstractControl {
    constructor(validators: ValidatorFn | ValidatorFn[] | null, asyncValidators: AsyncValidatorFn | AsyncValidatorFn[] | null);
    addAsyncValidators(validators: AsyncValidatorFn | AsyncValidatorFn[]): void;
    addValidators(validators: ValidatorFn | ValidatorFn[]): void;
    get asyncValidator(): AsyncValidatorFn | null;
    set asyncValidator(asyncValidatorFn: AsyncValidatorFn | null);
    clearAsyncValidators(): void;
    clearValidators(): void;
    get dirty(): boolean;
    disable(opts?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    get disabled(): boolean;
    enable(opts?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    get enabled(): boolean;
    readonly errors: ValidationErrors | null;
    get(path: Array<string | number> | string): AbstractControl | null;
    getError(errorCode: string, path?: Array<string | number> | string): any;
    getRawValue(): any;
    hasAsyncValidator(validator: AsyncValidatorFn): boolean;
    hasError(errorCode: string, path?: Array<string | number> | string): boolean;
    hasValidator(validator: ValidatorFn): boolean;
    get invalid(): boolean;
    markAllAsTouched(): void;
    markAsDirty(opts?: {
        onlySelf?: boolean;
    }): void;
    markAsPending(opts?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    markAsPristine(opts?: {
        onlySelf?: boolean;
    }): void;
    markAsTouched(opts?: {
        onlySelf?: boolean;
    }): void;
    markAsUntouched(opts?: {
        onlySelf?: boolean;
    }): void;
    get parent(): FormGroup | FormArray | null;
    abstract patchValue(value: any, options?: Object): void;
    get pending(): boolean;
    readonly pristine: boolean;
    removeAsyncValidators(validators: AsyncValidatorFn | AsyncValidatorFn[]): void;
    removeValidators(validators: ValidatorFn | ValidatorFn[]): void;
    abstract reset(value?: any, options?: Object): void;
    get root(): AbstractControl;
    setAsyncValidators(validators: AsyncValidatorFn | AsyncValidatorFn[] | null):
void;
    setErrors(errors: ValidationErrors | null, opts?: {
        emitEvent?: boolean;
    }): void;
    // (undocumented)
    setParent(parent: FormGroup | FormArray): void;
    setValidators(validators: ValidatorFn | ValidatorFn[] | null): void;
    abstract setValue(value: any, options?: Object): void;
    readonly status: FormControlStatus;
    readonly statusChanges: Observable<FormControlStatus>;
    readonly touched: boolean;
    get untouched(): boolean;
    get updateOn(): FormHooks;
    updateValueAndValidity(opts?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    get valid(): boolean;
    get validator(): ValidatorFn | null;
    set validator(validatorFn: ValidatorFn | null);
    readonly value: any;
    readonly valueChanges: Observable<any>;
}

// @public
export abstract class AbstractControlDirective {
    get asyncValidator(): AsyncValidatorFn | null;
    abstract get control(): AbstractControl | null;
    get dirty(): boolean | null;
    get disabled(): boolean | null;
    get enabled(): boolean | null;
    get errors(): ValidationErrors | null;
    getError(errorCode: string, path?: Array<string | number> | string): any;
    hasError(errorCode: string, path?: Array<string | number> | string): boolean;
    get invalid(): boolean | null;
    get path(): string[] | null;
    get pending(): boolean | null;
    get pristine(): boolean | null;
    reset(value?: any): void;
    get status(): string | null;
    get statusChanges(): Observable<any> | null;
    get touched(): boolean | null;
    get untouched(): boolean | null;
    get valid(): boolean | null;
    get validator(): ValidatorFn | null;
    get value(): any;
    get valueChanges(): Observable<any> | null;
}

// @public
export interface AbstractControlOptions {
    asyncValidators?: AsyncValidatorFn | AsyncValidatorFn[] | null;
    updateOn?: 'change' | 'blur' | 'submit';
    validators?: ValidatorFn | ValidatorFn[] | null;
}

// @public
export class AbstractFormGroupDirective extends ControlContainer implements OnInit,
OnDestroy {
    get control(): FormGroup;
    get formDirective(): Form | null;
    // (undocumented)
    ngOnDestroy(): void;
    // (undocumented)
    ngOnInit(): void;
    get path(): string[];
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<AbstractFormGroupDirective, never, never,
{}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<AbstractFormGroupDirective, never>;
}

// @public
export interface AsyncValidator extends Validator {
    validate(control: AbstractControl): Promise<ValidationErrors | null> |
Observable<ValidationErrors | null>;
}
```

```
// @public
export interface AsyncValidatorFn {
    // (undocumented)
    (control: AbstractControl): Promise<ValidationErrors | null> |
Observable<ValidationErrors | null>;
}

// @public
export class CheckboxControlValueAccessor extends BuiltInControlValueAccessor
implements ControlValueAccessor {
    writeValue(value: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<CheckboxControlValueAccessor,
"input[type=checkbox][formControlName],input[type=checkbox]
[formControl],input[type=checkbox][ngModel]", never, {}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<CheckboxControlValueAccessor, never>;
}

// @public
export class CheckboxRequiredValidator extends RequiredValidator {
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<CheckboxRequiredValidator,
"input[type=checkbox][required][formControlName],input[type=checkbox][required]
[formControl],input[type=checkbox][required][ngModel]", never, {}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<CheckboxRequiredValidator, never>;
}

// @public
export const COMPOSITION_BUFFER_MODE: InjectionToken<boolean>;

// @public
export abstract class ControlContainer extends AbstractControlDirective {
    get formDirective(): Form | null;
    name: string | number | null;
    get path(): string[] | null;
}

// @public
export interface ControlValueAccessor {
    registerOnChange(fn: any): void;
    registerOnTouched(fn: any): void;
    setDisabledState?(isDisabled: boolean): void;
    writeValue(obj: any): void;
}

// @public
export class DefaultValueAccessor extends BaseControlValueAccessor implements
ControlValueAccessor {
    constructor(renderer: Renderer2, elementRef: ElementRef, _compositionMode:
boolean);
```

```
    writeValue(value: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<DefaultValueAccessor,
"input:not([type=checkbox])
[formControlName],textarea[formControlName],input:not([type=checkbox])
[formControl],textarea[formControl],input:not([type=checkbox])
[ngModel],textarea[ngModel],[ngDefaultControl]", never, {}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<DefaultValueAccessor, [null, null, {
optional: true; }]>;
}

// @public
export class EmailValidator extends AbstractValidatorDirective {
    email: boolean | string;
    // (undocumented)
    enabled(input: boolean): boolean;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<EmailValidator, "[email]
[formControlName],[email][formControl],[email][ngModel]", never, { "email": "email";
}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<EmailValidator, never>;
}

// @public
export interface Form {
    addControl(dir: NgControl): void;
    addFormGroup(dir: AbstractFormGroupDirective): void;
    getControl(dir: NgControl): FormControl;
    getFormGroup(dir: AbstractFormGroupDirective): FormGroup;
    removeControl(dir: NgControl): void;
    removeFormGroup(dir: AbstractFormGroupDirective): void;
    updateModel(dir: NgControl, value: any): void;
}

// @public
export class FormArray extends AbstractControl {
    constructor(controls: AbstractControl[], validatorOrOpts?: ValidatorFn |
ValidatorFn[] | AbstractControlOptions | null, asyncValidator?: AsyncValidatorFn |
AsyncValidatorFn[] | null);
    at(index: number): AbstractControl;
    clear(options?: {
        emitEvent?: boolean;
    }): void;
    // (undocumented)
    controls: AbstractControl[];
    getRawValue(): any[];
    insert(index: number, control: AbstractControl, options?: {
        emitEvent?: boolean;
    }): void;
    get length(): number;
    patchValue(value: any[], options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    push(control: AbstractControl, options?: {
        emitEvent?: boolean;
    }): void;
    removeAt(index: number, options?: {
        emitEvent?: boolean;
    }): void;
    reset(value?: any, options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    setControl(index: number, control: AbstractControl, options?: {
        emitEvent?: boolean;
    }): void;
    setValue(value: any[], options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
}

// @public
export class FormArrayName extends ControlContainer implements OnInit, OnDestroy {
    constructor(parent: ControlContainer, validators: (Validator | ValidatorFn)[],
asyncValidators: (AsyncValidator | AsyncValidatorFn)[]);
    get control(): FormArray;
    get formDirective(): FormGroupDirective | null;
    name: string | number | null;
    ngOnDestroy(): void;
    ngOnInit(): void;
    get path(): string[];
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<FormArrayName, "[formArrayName]", never,
{ "name": "formArrayName"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<FormArrayName, [{ optional: true; host:
true; skipSelf: true; }, { optional: true; self: true; }, { optional: true; self:
true; }]>;
}

// @public
export class FormBuilder {
    array(controlsConfig: any[], validatorOrOpts?: ValidatorFn | ValidatorFn[] |
AbstractControlOptions | null, asyncValidator?: AsyncValidatorFn |
AsyncValidatorFn[] | null): FormArray;
    control(formState: any, validatorOrOpts?: ValidatorFn | ValidatorFn[] |
FormControlOptions | null, asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[] |
null): FormControl;
    group(controlsConfig: {
        [key: string]: any;
    }, options?: AbstractControlOptions | null): FormGroup;
    // @deprecated
    group(controlsConfig: {
        [key: string]: any;
    }, options: {
        [key: string]: any;
    }): FormGroup;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<FormBuilder, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<FormBuilder>;
}

// @public
export interface FormControl extends AbstractControl {
    readonly defaultValue: any;
    patchValue(value: any, options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
        emitModelToViewChange?: boolean;
        emitViewToModelChange?: boolean;
    }): void;
    registerOnChange(fn: Function): void;
    registerOnDisabledChange(fn: (isDisabled: boolean) => void): void;
    reset(formState?: any, options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    setValue(value: any, options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
        emitModelToViewChange?: boolean;
        emitViewToModelChange?: boolean;
    }): void;
}

// @public (undocumented)
export const FormControl: ɵFormControlCtor;

// @public
export class FormControlDirective extends NgControl implements OnChanges, OnDestroy
{
    constructor(validators: (Validator | ValidatorFn)[], asyncValidators:
(AsyncValidator | AsyncValidatorFn)[], valueAccessors: ControlValueAccessor[],
_ngModelWarningConfig: string | null);
    get control(): FormControl;
    form: FormControl;
    set isDisabled(isDisabled: boolean);
    // @deprecated (undocumented)
    model: any;
    // (undocumented)
    ngOnChanges(changes: SimpleChanges): void;
    // (undocumented)
    ngOnDestroy(): void;
    get path(): string[];
    // @deprecated (undocumented)
    update: EventEmitter<any>;
    viewModel: any;
    viewToModelUpdate(newValue: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<FormControlDirective, "[formControl]",
["ngForm"], { "form": "formControl"; "isDisabled": "disabled"; "model": "ngModel";
}, { "update": "ngModelChange"; }, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<FormControlDirective, [{ optional: true;
self: true; }, { optional: true; self: true; }, { optional: true; self: true; }, {
optional: true; }]>;
}

// @public
export class FormControlName extends NgControl implements OnChanges, OnDestroy {
    constructor(parent: ControlContainer, validators: (Validator | ValidatorFn)[],
asyncValidators: (AsyncValidator | AsyncValidatorFn)[], valueAccessors:
ControlValueAccessor[], _ngModelWarningConfig: string | null);
    readonly control: FormControl;
    get formDirective(): any;
    set isDisabled(isDisabled: boolean);
    // @deprecated (undocumented)
    model: any;
    name: string | number | null;
    // (undocumented)
    ngOnChanges(changes: SimpleChanges): void;
    // (undocumented)
    ngOnDestroy(): void;
    get path(): string[];
    // @deprecated (undocumented)
    update: EventEmitter<any>;
    viewToModelUpdate(newValue: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<FormControlName, "[formControlName]",
never, { "name": "formControlName"; "isDisabled": "disabled"; "model": "ngModel"; },
{ "update": "ngModelChange"; }, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<FormControlName, [{ optional: true; host:
true; skipSelf: true; }, { optional: true; self: true; }, { optional: true; self:
true; }, { optional: true; self: true; }, { optional: true; }]>;
}

// @public
export interface FormControlOptions extends AbstractControlOptions {
    initialValueIsDefault?: boolean;
}

// @public
```

```
export type FormControlStatus = 'VALID' | 'INVALID' | 'PENDING' | 'DISABLED';

// @public
export class FormGroup extends AbstractControl {
    constructor(controls: {
        [key: string]: AbstractControl;
    }, validatorOrOpts?: ValidatorFn | ValidatorFn[] | AbstractControlOptions |
null, asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[] | null);
    addControl(name: string, control: AbstractControl, options?: {
        emitEvent?: boolean;
    }): void;
    contains(controlName: string): boolean;
    // (undocumented)
    controls: {
        [key: string]: AbstractControl;
    };
    getRawValue(): any;
    patchValue(value: {
        [key: string]: any;
    }, options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    registerControl(name: string, control: AbstractControl): AbstractControl;
    removeControl(name: string, options?: {
        emitEvent?: boolean;
    }): void;
    reset(value?: any, options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
    setControl(name: string, control: AbstractControl, options?: {
        emitEvent?: boolean;
    }): void;
    setValue(value: {
        [key: string]: any;
    }, options?: {
        onlySelf?: boolean;
        emitEvent?: boolean;
    }): void;
}

// @public
export class FormGroupDirective extends ControlContainer implements Form, OnChanges,
OnDestroy {
    constructor(validators: (Validator | ValidatorFn)[], asyncValidators:
(AsyncValidator | AsyncValidatorFn)[]);
    addControl(dir: FormControlName): FormControl;
    addFormArray(dir: FormArrayName): void;
    addFormGroup(dir: FormGroupName): void;
    get control(): FormGroup;
    directives: FormControlName[];
    form: FormGroup;
    get formDirective(): Form;
    getControl(dir: FormControlName): FormControl;
    getFormArray(dir: FormArrayName): FormArray;
    getFormGroup(dir: FormGroupName): FormGroup;
    // (undocumented)
    ngOnChanges(changes: SimpleChanges): void;
    // (undocumented)
    ngOnDestroy(): void;
    ngSubmit: EventEmitter<any>;
    onReset(): void;
    onSubmit($event: Event): boolean;
    get path(): string[];
    removeControl(dir: FormControlName): void;
    removeFormArray(dir: FormArrayName): void;
    removeFormGroup(dir: FormGroupName): void;
    resetForm(value?: any): void;
    readonly submitted: boolean;
    updateModel(dir: FormControlName, value: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<FormGroupDirective, "[formGroup]",
["ngForm"], { "form": "formGroup"; }, { "ngSubmit": "ngSubmit"; }, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<FormGroupDirective, [{ optional: true;
self: true; }, { optional: true; self: true; }]>;
}

// @public
export class FormGroupName extends AbstractFormGroupDirective implements OnInit,
OnDestroy {
    constructor(parent: ControlContainer, validators: (Validator | ValidatorFn)[],
asyncValidators: (AsyncValidator | AsyncValidatorFn)[]);
    name: string | number | null;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<FormGroupName, "[formGroupName]", never,
{ "name": "formGroupName"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<FormGroupName, [{ optional: true; host:
true; skipSelf: true; }, { optional: true; self: true; }, { optional: true; self:
true; }]>;
}

// @public
export class FormsModule {
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<FormsModule, never>;
    // (undocumented)
    static ɵinj: i0.ɵɵInjectorDeclaration<FormsModule>;
    // (undocumented)
    static ɵmod: i0.ɵɵNgModuleDeclaration<FormsModule, [typeof i1_2.NgModel, typeof
i2_2.NgModelGroup, typeof i3_2.NgForm], never, [typeof
i4_2.ɵInternalFormsSharedModule, typeof i1_2.NgModel, typeof i2_2.NgModelGroup,
typeof i3_2.NgForm]>;
}

// @public
export class MaxLengthValidator extends AbstractValidatorDirective {
    maxlength: string | number | null;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<MaxLengthValidator, "[maxlength]
[formControlName],[maxlength][formControl],[maxlength][ngModel]", never, {
"maxlength": "maxlength"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<MaxLengthValidator, never>;
}

// @public
export class MaxValidator extends AbstractValidatorDirective {
    max: string | number | null;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<MaxValidator, "input[type=number][max]
[formControlName],input[type=number][max][formControl],input[type=number][max]
[ngModel]", never, { "max": "max"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<MaxValidator, never>;
}

// @public
export class MinLengthValidator extends AbstractValidatorDirective {
    minlength: string | number | null;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<MinLengthValidator, "[minlength]
[formControlName],[minlength][formControl],[minlength][ngModel]", never, {
"minlength": "minlength"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<MinLengthValidator, never>;
}

// @public
export class MinValidator extends AbstractValidatorDirective {
    min: string | number | null;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<MinValidator, "input[type=number][min]
[formControlName],input[type=number][min][formControl],input[type=number][min]
[ngModel]", never, { "min": "min"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<MinValidator, never>;
}

// @public
export const NG_ASYNC_VALIDATORS: InjectionToken<(Function | Validator)[]>;

// @public
export const NG_VALIDATORS: InjectionToken<(Function | Validator)[]>;
```

```
// @public
export const NG_VALUE_ACCESSOR: InjectionToken<readonly ControlValueAccessor[]>;

// @public
export abstract class NgControl extends AbstractControlDirective {
    name: string | number | null;
    valueAccessor: ControlValueAccessor | null;
    abstract viewToModelUpdate(newValue: any): void;
}

// @public
export class NgControlStatus extends AbstractControlStatus {
    constructor(cd: NgControl);
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<NgControlStatus, "[formControlName],
[ngModel],[formControl]", never, {}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<NgControlStatus, [{ self: true; }]>;
}

// @public
export class NgControlStatusGroup extends AbstractControlStatus {
    constructor(cd: ControlContainer);
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<NgControlStatusGroup, "[formGroupName],
[formArrayName],[ngModelGroup],[formGroup],form:not([ngNoForm]),[ngForm]", never,
{}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<NgControlStatusGroup, [{ optional: true;
self: true; }]>;
}

// @public
export class NgForm extends ControlContainer implements Form, AfterViewInit {
    constructor(validators: (Validator | ValidatorFn)[], asyncValidators:
(AsyncValidator | AsyncValidatorFn)[]);
    addControl(dir: NgModel): void;
    addFormGroup(dir: NgModelGroup): void;
    get control(): FormGroup;
    get controls(): {
        [key: string]: AbstractControl;
    };
    form: FormGroup;
    get formDirective(): Form;
    getControl(dir: NgModel): FormControl;
    getFormGroup(dir: NgModelGroup): FormGroup;
    // (undocumented)
    ngAfterViewInit(): void;
    ngSubmit: EventEmitter<any>;
    onReset(): void;
    onSubmit($event: Event): boolean;
    options: {
        updateOn?: FormHooks;
    };
    get path(): string[];
    removeControl(dir: NgModel): void;
    removeFormGroup(dir: NgModelGroup): void;
    resetForm(value?: any): void;
    setValue(value: {
        [key: string]: any;
    }): void;
    readonly submitted: boolean;
    updateModel(dir: NgControl, value: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<NgForm,
"form:not([ngNoForm]):not([formGroup]),ng-form,[ngForm]", ["ngForm"], { "options":
"ngFormOptions"; }, { "ngSubmit": "ngSubmit"; }, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<NgForm, [{ optional: true; self: true; }, {
optional: true; self: true; }]>;
}

// @public
export class NgModel extends NgControl implements OnChanges, OnDestroy {
    constructor(parent: ControlContainer, validators: (Validator | ValidatorFn)[],
asyncValidators: (AsyncValidator | AsyncValidatorFn)[], valueAccessors:
ControlValueAccessor[], _changeDetectorRef?: ChangeDetectorRef | null | undefined);
    // (undocumented)
    readonly control: FormControl;
    get formDirective(): any;
    isDisabled: boolean;
    model: any;
    name: string;
    // (undocumented)
    static ngAcceptInputType_isDisabled: boolean | string;
    // (undocumented)
    ngOnChanges(changes: SimpleChanges): void;
    // (undocumented)
    ngOnDestroy(): void;
    options: {
        name?: string;
        standalone?: boolean;
        updateOn?: FormHooks;
    };
    get path(): string[];
    update: EventEmitter<any>;
    viewModel: any;
    viewToModelUpdate(newValue: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<NgModel, "
[ngModel]:not([formControlName]):not([formControl])", ["ngModel"], { "name": "name";
"isDisabled": "disabled"; "model": "ngModel"; "options": "ngModelOptions"; }, {
"update": "ngModelChange"; }, never>;
```

```
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<NgModel, [{ optional: true; host: true; },
{ optional: true; self: true; }, { optional: true; self: true; }, { optional: true;
self: true; }, { optional: true; }]>;
}

// @public
export class NgModelGroup extends AbstractFormGroupDirective implements OnInit,
OnDestroy {
    constructor(parent: ControlContainer, validators: (Validator | ValidatorFn)[],
asyncValidators: (AsyncValidator | AsyncValidatorFn)[]);
    name: string;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<NgModelGroup, "[ngModelGroup]",
["ngModelGroup"], { "name": "ngModelGroup"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<NgModelGroup, [{ host: true; skipSelf:
true; }, { optional: true; self: true; }, { optional: true; self: true; }]>;
}

// @public
export class NgSelectOption implements OnDestroy {
    constructor(_element: ElementRef, _renderer: Renderer2, _select:
SelectControlValueAccessor);
    id: string;
    // (undocumented)
    ngOnDestroy(): void;
    set ngValue(value: any);
    set value(value: any);
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<NgSelectOption, "option", never, {
"ngValue": "ngValue"; "value": "value"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<NgSelectOption, [null, null, { optional:
true; host: true; }]>;
}

// @public
export class NumberValueAccessor extends BuiltInControlValueAccessor implements
ControlValueAccessor {
    registerOnChange(fn: (_: number | null) => void): void;
    writeValue(value: number): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<NumberValueAccessor, "input[type=number]
[formControlName],input[type=number][formControl],input[type=number][ngModel]",
never, {}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<NumberValueAccessor, never>;
}

// @public
export class PatternValidator extends AbstractValidatorDirective {
    pattern: string | RegExp;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<PatternValidator, "[pattern]
[formControlName],[pattern][formControl],[pattern][ngModel]", never, { "pattern":
"pattern"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<PatternValidator, never>;
}

// @public
export class RadioControlValueAccessor extends BuiltInControlValueAccessor
implements ControlValueAccessor, OnDestroy, OnInit {
    constructor(renderer: Renderer2, elementRef: ElementRef, _registry:
RadioControlRegistry, _injector: Injector);
    fireUncheck(value: any): void;
    formControlName: string;
    name: string;
    // (undocumented)
    ngOnDestroy(): void;
    // (undocumented)
    ngOnInit(): void;
    onChange: () => void;
    registerOnChange(fn: (_: any) => {}): void;
    value: any;
    writeValue(value: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<RadioControlValueAccessor,
"input[type=radio][formControlName],input[type=radio][formControl],input[type=radio]
[ngModel]", never, { "name": "name"; "formControlName": "formControlName"; "value":
"value"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<RadioControlValueAccessor, never>;
}

// @public
export class RangeValueAccessor extends BuiltInControlValueAccessor implements
ControlValueAccessor {
    registerOnChange(fn: (_: number | null) => void): void;
    writeValue(value: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<RangeValueAccessor, "input[type=range]
[formControlName],input[type=range][formControl],input[type=range][ngModel]", never,
{}, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<RangeValueAccessor, never>;
}

// @public
export class ReactiveFormsModule {
    static withConfig(opts: {
        warnOnNgModelWithFormControl: 'never' | 'once' | 'always';
    }): ModuleWithProviders<ReactiveFormsModule>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<ReactiveFormsModule, never>;
    // (undocumented)
    static ɵinj: i0.ɵɵInjectorDeclaration<ReactiveFormsModule>;
    // (undocumented)
    static ɵmod: i0.ɵɵNgModuleDeclaration<ReactiveFormsModule, [typeof
i5_2.FormControlDirective, typeof i6_2.FormGroupDirective, typeof
i7_2.FormControlName, typeof i8_2.FormGroupName, typeof i8_2.FormArrayName], never,
[typeof i4_2.ɵInternalFormsSharedModule, typeof i5_2.FormControlDirective, typeof
i6_2.FormGroupDirective, typeof i7_2.FormControlName, typeof i8_2.FormGroupName,
typeof i8_2.FormArrayName]>;
}

// @public
export class RequiredValidator extends AbstractValidatorDirective {
    // (undocumented)
    enabled(input: boolean): boolean;
    required: boolean | string;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<RequiredValidator, ":not([type=checkbox])
[required][formControlName],:not([type=checkbox])[required]
[formControl],:not([type=checkbox])[required][ngModel]", never, { "required":
"required"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<RequiredValidator, never>;
}

// @public
export class SelectControlValueAccessor extends BuiltInControlValueAccessor
implements ControlValueAccessor {
    set compareWith(fn: (o1: any, o2: any) => boolean);
    registerOnChange(fn: (value: any) => any): void;
    // (undocumented)
    value: any;
    writeValue(value: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<SelectControlValueAccessor,
"select:not([multiple])[formControlName],select:not([multiple])
[formControl],select:not([multiple])[ngModel]", never, { "compareWith":
"compareWith"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<SelectControlValueAccessor, never>;
}

// @public
export class SelectMultipleControlValueAccessor extends BuiltInControlValueAccessor
implements ControlValueAccessor {
    set compareWith(fn: (o1: any, o2: any) => boolean);
    registerOnChange(fn: (value: any) => any): void;
    value: any;
    writeValue(value: any): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<SelectMultipleControlValueAccessor,
"select[multiple][formControlName],select[multiple][formControl],select[multiple]
[ngModel]", never, { "compareWith": "compareWith"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<SelectMultipleControlValueAccessor, never>;
}

// @public
export type UntypedFormArray = FormArray;

// @public (undocumented)
export const UntypedFormArray: UntypedFormArrayCtor;

// @public
export class UntypedFormBuilder extends FormBuilder {
    // (undocumented)
    array(controlsConfig: any[], validatorOrOpts?: ValidatorFn | ValidatorFn[] |
AbstractControlOptions | null, asyncValidator?: AsyncValidatorFn |
AsyncValidatorFn[] | null): UntypedFormArray;
    // (undocumented)
    control(formState: any, validatorOrOpts?: ValidatorFn | ValidatorFn[] |
FormControlOptions | null, asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[] |
null): UntypedFormControl;
    // (undocumented)
    group(controlsConfig: {
        [key: string]: any;
    }, options?: AbstractControlOptions | null): UntypedFormGroup;
    // @deprecated (undocumented)
    group(controlsConfig: {
        [key: string]: any;
    }, options: {
        [key: string]: any;
    }): UntypedFormGroup;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<UntypedFormBuilder, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<UntypedFormBuilder>;
}

// @public
export type UntypedFormControl = FormControl;

// @public (undocumented)
export const UntypedFormControl: UntypedFormControlCtor;

// @public
export type UntypedFormGroup = FormGroup;

// @public (undocumented)
export const UntypedFormGroup: UntypedFormGroupCtor;

// @public
```

```
export type ValidationErrors = {
    [key: string]: any;
};

// @public
export interface Validator {
    registerOnValidatorChange?(fn: () => void): void;
    validate(control: AbstractControl): ValidationErrors | null;
}

// @public
export interface ValidatorFn {
    // (undocumented)
    (control: AbstractControl): ValidationErrors | null;
}

// @public
export class Validators {
    static compose(validators: null): null;
    // (undocumented)
    static compose(validators: (ValidatorFn | null | undefined)[]): ValidatorFn |
null;
    static composeAsync(validators: (AsyncValidatorFn | null)[]): AsyncValidatorFn |
null;
    static email(control: AbstractControl): ValidationErrors | null;
    static max(max: number): ValidatorFn;
    static maxLength(maxLength: number): ValidatorFn;
    static min(min: number): ValidatorFn;
    static minLength(minLength: number): ValidatorFn;
    static nullValidator(control: AbstractControl): ValidationErrors | null;
    static pattern(pattern: string | RegExp): ValidatorFn;
    static required(control: AbstractControl): ValidationErrors | null;
    static requiredTrue(control: AbstractControl): ValidationErrors | null;
}

// @public (undocumented)
export const VERSION: Version;

// (No @packageDocumentation comment for this package)
```