API > @angular/forms

NgForm DIRECTIVE

Creates a top-level `FormGroup` instance and binds it to a form to track aggregate form value and validation status.

See more...

Exported from

- **FormsModule**

Selectors

form:not([ngNoForm]):not([formGroup])

ng-form

[ngForm]

Properties

Property	Description
`submitted: boolean`	***Read-Only*** Returns whether the form submission has been triggered.
`form: FormGroup`	The `FormGroup` instance created for this form.
`@Output()` `ngSubmit: EventEmitter`	Event emitter for the "ngSubmit" event

`@Input('ngFormOptions')`
`options: {`
`updateOn?:`
`FormHooks;`
`}`
	Tracks options for the `NgForm` instance.
	
	updateOn: Sets the default `updateOn` value for all child `NgModels` below it unless explicitly set by a child `NgModel` using `ngModelOptions`). Defaults to 'change'. Possible values: `'change'` | `'blur'` | `'submit'`.

`formDirective: Form`	***Read-Only*** The directive instance.

Property	Description
`control: FormGroup`	***Read-Only*** The internal `FormGroup` instance.
`path: string[]`	***Read-Only*** Returns an array representing the path to this group. Because this directive always lives at the top level of a form, it is always an empty array.
`controls: { [key: string]: AbstractControl; }`	***Read-Only*** Returns a map of the controls in this group.

Inherited from `ControlContainer`

`name: string | number | null`

`formDirective: Form | null`

`path: string[] | null`

Inherited from `AbstractControlDirective`

```
abstract control: AbstractControl | null

value: any

valid: boolean | null

invalid: boolean | null

pending: boolean | null

disabled: boolean | null

enabled: boolean | null

errors: ValidationErrors | null

pristine: boolean | null

dirty: boolean | null

touched: boolean | null

status: string | null

untouched: boolean | null

statusChanges: Observable<any> | null

valueChanges: Observable<any> | null

path: string[] | null

validator: ValidatorFn | null

asyncValidator: AsyncValidatorFn | null
```

Template variable references

Identifier	Usage
`ngForm`	`#myTemplateVar="ngForm"`

Description

As soon as you import the `FormsModule`, this directive becomes active by default on all `<form>` tags. You don't need to add a special selector.

You optionally export the directive into a local template variable using `ngForm` as the key (ex: `#myForm="ngForm"`). This is optional, but useful. Many properties from the underlying `FormGroup` instance are duplicated on the directive itself, so a reference to it gives you access to the aggregate value and validity status of the form, as well as user interaction properties like `dirty` and `touched`.

To register child controls with the form, use `NgModel` with a `name` attribute. You may use `NgModelGroup` to create sub-groups within the form.

If necessary, listen to the directive's `ngSubmit` event to be notified when the user has triggered a form submission. The `ngSubmit` event emits the original form submission event.

In template driven forms, all `<form>` tags are automatically tagged as `NgForm`. To import the `FormsModule` but skip its usage in some forms, for example, to use native HTML5 validation, add the `ngNoForm` and the `<form>` tags won't create an `NgForm` directive. In reactive forms, using `ngNoForm` is unnecessary because the `<form>` tags are inert. In that case, you would refrain from using the `formGroup` directive.

Listening for form submission

The following example shows how to capture the form values from the "ngSubmit" event.

```
import {Component} from '@angular/core';
import {NgForm} from '@angular/forms';

@Component({
  selector: 'example-app',
  template: `
    <form #f="ngForm" (ngSubmit)="onSubmit(f)"
novalidate>
      <input name="first" ngModel required
#first="ngModel">
      <input name="last" ngModel>
      <button>Submit</button>
    </form>

    <p>First name value: {{ first.value }}</p>
    <p>First name valid: {{ first.valid }}</p>
    <p>Form value: {{ f.value | json }}</p>
    <p>Form valid: {{ f.valid }}</p>
  `,
})
export class SimpleFormComp {
  onSubmit(f: NgForm) {
    console.log(f.value);  // { first: '', last: '' }
    console.log(f.valid);  // false
  }
}
```

Setting the update options

The following example shows you how to change the "updateOn" option from its default using ngFormOptions.

```
<form [ngFormOptions]="{updateOn: 'blur'}">
   <input name="one" ngModel>  <!-- this ngModel will update on blur -->
</form>
```

Native DOM validation UI

In order to prevent the native DOM form validation UI from interfering with Angular's form validation, Angular automatically adds the `novalidate` attribute on any `<form>` whenever `FormModule` or `ReactiveFormModule` are imported into the application. If you want to explicitly enable native DOM validation UI with Angular forms, you can add the `ngNativeValidate` attribute to the `<form>` element:

```
<form ngNativeValidate>
  ...
</form>
```

Methods

addControl()

Method that sets up the control directive in this group, re-calculates its value and validity, and adds the instance to the internal list of directives.

```
addControl(dir: NgModel): void
```

Parameters

dir	NgModel	The NgModel directive instance.

Returns

void

getControl()

Retrieves the FormControl instance from the provided NgModel directive.

```
getControl(dir: NgModel): FormControl
```

Parameters

dir	NgModel	The NgModel directive instance.

Returns

FormControl

removeControl()

Removes the `NgModel` instance from the internal list of directives

```
removeControl(dir: NgModel): void
```

Parameters

dir	`NgModel`	The `NgModel` directive instance.

Returns

`void`

addFormGroup()

Adds a new `NgModelGroup` directive instance to the form.

```
addFormGroup(dir: NgModelGroup): void
```

Parameters

dir	`NgModelGroup`	The `NgModelGroup` directive instance.

Returns

`void`

removeFormGroup()

Removes the `NgModelGroup` directive instance from the form.

```
removeFormGroup(dir: NgModelGroup): void
```

Parameters

dir	`NgModelGroup`	The `NgModelGroup` directive instance.

Returns

`void`

getFormGroup()

Retrieves the `FormGroup` for a provided `NgModelGroup` directive instance

```
getFormGroup(dir: NgModelGroup): FormGroup
```

Parameters

dir	`NgModelGroup`	The `NgModelGroup` directive instance.

Returns

`FormGroup`

updateModel()

Sets the new value for the provided `NgControl` directive.

```
updateModel(dir: NgControl, value: any): void
```

Parameters

dir	`NgControl`	The `NgControl` directive instance.
value	`any`	The new value for the directive's control.

Returns

`void`

setValue()

Sets the value for this `FormGroup` .

```
setValue(value: { [key: string]: any; }): void
```

Parameters

value	`object`	The new value

Returns

`void`

onSubmit()

Method called when the "submit" event is triggered on the form. Triggers the `ngSubmit` emitter to emit the "submit" event as its payload.

```
onSubmit($event: Event): boolean
```

Parameters

`$event`	`Event`	The "submit" event object

Returns

`boolean`

onReset()

Method called when the "reset" event is triggered on the form.

```
onReset(): void
```

Parameters

There are no parameters.

Returns

`void`

resetForm()

Resets the form to an initial value and resets its submitted status.

```
resetForm(value: any = undefined): void
```

Parameters

value	any	The new value for the form. Optional. Default is `undefined`.

Returns

`void`

Inherited from AbstractControlDirective

```
reset(value: any = undefined): void
```

```
hasError(errorCode: string, path?: string | (string | number)[]): boolean
```

```
getError(errorCode: string, path?: string | (string | number)[]): any
```