



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Reactive Forms and Validation

Hosted by Alex Thalhammer

Pro

Auto generated
object tree

Simple to use

Contra

Dynamic Forms?

Control?

Testing?

Lot of code in
HTML-template

Reactive Forms



ReactiveFormsModule

```
@Component({  
  imports: [  
    ReactiveFormsModule,  
    CommonModule,  
    [...]  
,  
    [...]  
])  
export class FlightEditComponent{}
```



@LX_T

Reactive Forms

```
export class FlightSearchComponent {  
  readonly searchForm: FormGroup;  
  [...]  
}
```

Reactive Forms

```
export class FlightSearchComponent {  
  readonly searchForm: FormGroup;  
  
  constructor(...) {  
    const fromControl = new FormControl('Graz');  
    const toControl = new FormControl('Hamburg');  
    this.searchForm = new FormGroup({ from: fromControl, to: toControl});  
  
    [...]  
  }  
}
```

Reactive Forms

```
export class FlightSearchComponent {  
  readonly searchForm: FormGroup;  
  
  constructor(...) {  
    const fromControl = new UntypedFormControl('Graz');  
    const toControl = new UntypedFormControl('Hamburg');  
    this.searchForm = new FormGroup({ from: fromControl, to: toControl});  
  
    [...]  
  }  
}
```

Reactive Forms

```
export class FlightSearchComponent {  
  
  readonly searchForm: FormGroup;  
  
  constructor(...) {  
    const fromControl = new FormControl('Graz');  
    const toControl = new FormControl('Hamburg');  
    this.searchForm = new FormGroup({ from: fromControl, to: toControl});  
  
    fromControl.validator = Validators.required;  
    [...]  
  }  
}
```

Reactive Forms

```
export class FlightSearchComponent {  
  
  readonly searchForm: FormGroup;  
  
  constructor(...) {  
    const fromControl = new FormControl('Graz');  
    const toControl = new FormControl('Hamburg');  
    this.searchForm = new FormGroup({ from: fromControl, to: toControl});  
  
    fromControl.validator =  
      Validators.compose([Validators.required, Validators.minLength(3)]);  
  }  
}
```

Reactive Forms

```
export class FlightSearchComponent {  
  
    readonly searchForm: FormGroup;  
  
    constructor(...) {  
        [...]  
  
        fromControl.validator =  
            Validators.compose([Validators.required, Validators.minLength(3)]);  
  
        fromControl.asyncValidator = Validators.composeAsync(...);  
    }  
}
```

FormBuilder

```
export class FlightSearchComponent {  
  
  readonly searchForm: FormGroup;  
  
  constructor(private readonly fb: FormBuilder, ...) {  
    this.searchForm = this.fb.group({  
      from: ['Graz', Validators.required],  
      to: ['Hamburg', Validators.required]  
    });  
    [...]  
  }  
}
```

FormBuilder

```
export class FlightSearchComponent {  
  
  readonly searchForm: FormGroup;  
  
  constructor(private readonly fb: FormBuilder, ...) {  
    this.searchForm = this.fb.group({  
      from: ['Graz', [Validators.required, Validators.minLength(3)]],  
      to: ['Hamburg', Validators.required]  
    });  
    [...]  
  }  
}
```



@LX_T

FormBuilder

```
export class FlightSearchComponent {  
  
  readonly searchForm: FormGroup;  
  
  constructor(private readonly fb: FormBuilder, ...) {  
    this.searchForm = this.fb.group(  
      from: ['Graz', [Validators.required, Validators.minLength(3)], /* asyncValidator */ ],  
      to: ['Hamburg', Validators.required]  
    );  
    [...]  
  }  
}
```

FormBuilder

```
export class FlightSearchComponent {  
  protected readonly searchForm = inject(FormBuilder).group({  
    from: ['Graz', [Validators.required, Validators.minLength(3)]],  
    to: ['Hamburg', Validators.required]  
  });  
}
```

API

```
this.searchForm.valueChanges.subscribe((newFormValue) => {  
  console.debug('form value has changed', newFormValue);  
});
```

```
this.searchForm.controls['from'].valueChanges.subscribe((newFromValue) => {  
  console.debug('from input has changed ', newFromValue);  
});
```

```
let fromValue = this.searchForm.controls['from'].value; // directly via control  
const toValue = this.searchForm.value['to'];
```

```
const formValue = this.searchForm.value;  
fromValue = formValue['from']; // via form value object
```

Reactive Forms

```
<form [formGroup]="searchForm">  
  <input formControlName="from" id="from" class="form-control">  
  [...]  
</form>
```



@LX_T

Reactive Forms

```
<form [formGroup]="searchForm">
  <label for="from">From</label>
  <input formControlName="from" id="from" class="form-control">

  @if (searchForm.controls['from'].invalid) {
    <div>...Error...</div>
  }

  [...]

</form>
```



@LX_T

DEMO

LAB

Validators for Reactive Forms



Reactive validators === functions

A simple validator

```
export function validateCity(c: AbstractControl): ValidationErrors | null {
  if (c.value === 'Graz' || c.value === 'Hamburg') {
    return null;
  }

  return { city: true };
}
```

Apply validators

```
this.form = fb.group({  
  from: [  
    'Graz',  
    [  
      validate  
    ],  
    [  
      /* asyncValidator */  
    ]  
  ],  
  to: ['Hamburg', Validators.required]  
});
```

Parameterizable validators

```
export function validateWithParams(validCities: string[]): ValidatorFn {  
  [...]  
}  
}
```

Parameterizable validators

```
export function validateWithParams(validCities: string[]): ValidatorFn {  
  return (c: AbstractControl): ValidationErrors | null => {  
    [...]  
  };  
}
```

Parameterizable validators

```
export function validateWithParams(validCities: string[]): ValidatorFn {  
  return (c: AbstractControl): ValidationErrors | null => {  
    if (c.value && !validCities.includes(c.value)) {  
      return {  
        city: {  
          actualCity: c.value,  
          validCities: validCities.join(', ')  
        }  
      };  
    }  
  
    return null;  
  };  
}
```

Use validators

```
this.form = fb.group({  
  from: [  
    'Graz',  
    [Validators.required, validateWithParams(['Graz', 'Hamburg'])],  
  ],  
  to: ['Hamburg', Validators.required]  
});
```

DEMO

Asynchronous validators

```
export function cityValidatorAsync(flightService): AsyncValidatorFn {  
  return (ctrl: AbstractControl): Observable<ValidationErrors | null> => {  
    [...]  
    return observable;  
  }  
}
```

Use async validators

```
this.form = fb.group({
  from: [
    'Graz',
    [Validators.required, validateWithParams(['Graz', 'Hamburg'])],
    cityValidatorAsync(this.flightService)
  ],
  to: ['Hamburg', Validators.required]
});
```

DEMO

Multifield validators

```
export function validateMultiField(...): ValidationFn {  
  return (control: AbstractControl): ValidationErrors | null {  
    const formGroup = control as FormGroup;  
    [...]  
  }  
};
```

Use multiform validators

```
this.editForm = this.fb.group({ ... });

this.editForm.validator = validateRoundTrip;

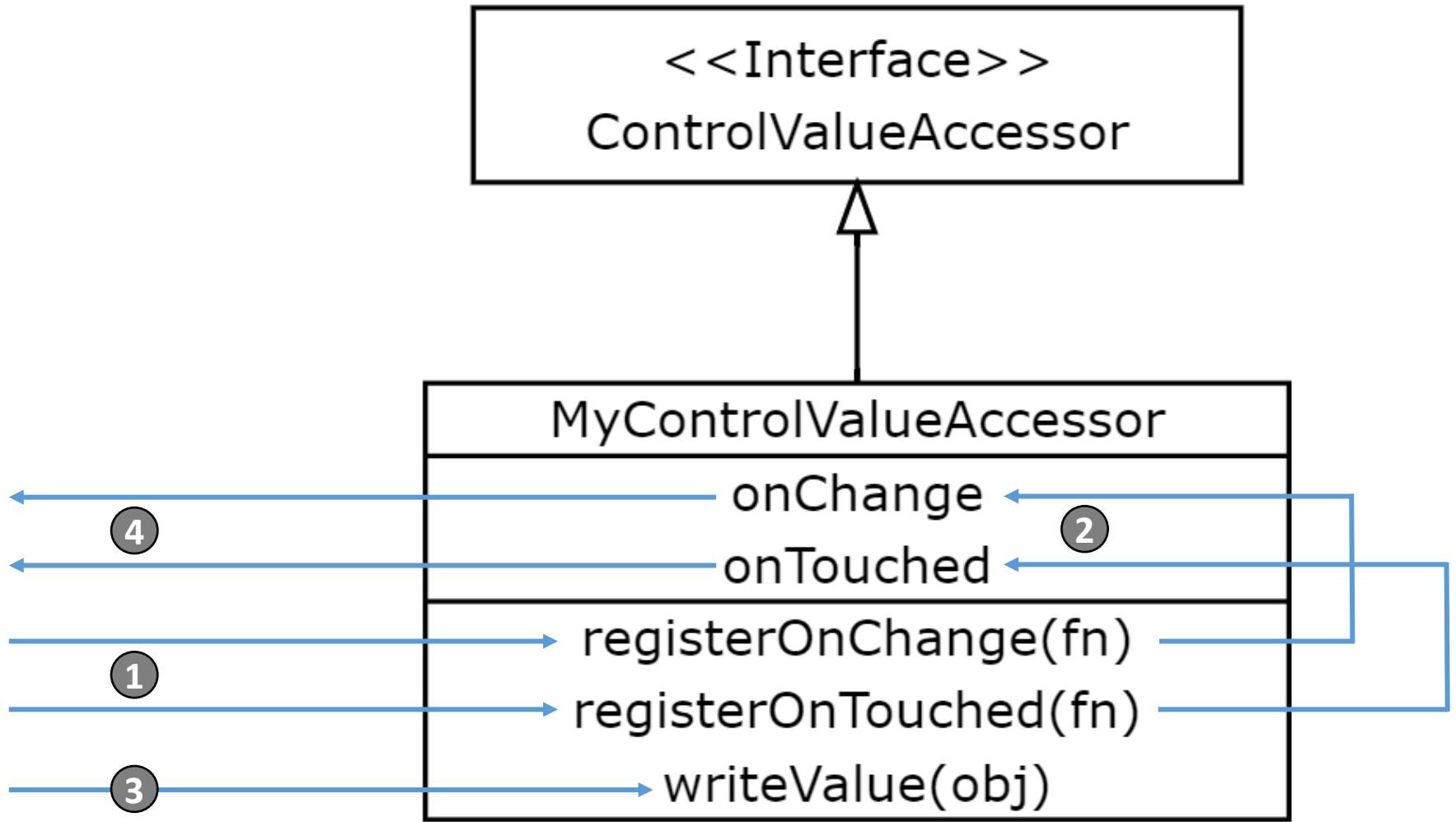
this.editForm.validator = validators.compose([validateRoundTrip, ...])
```

```
this.editForm = this.fb.group(
  { ... },
  { validators: validateRoundTrip }
);
```

DEMO

LAB

Custom Form Controls with Control Value Accessors



@LX_T

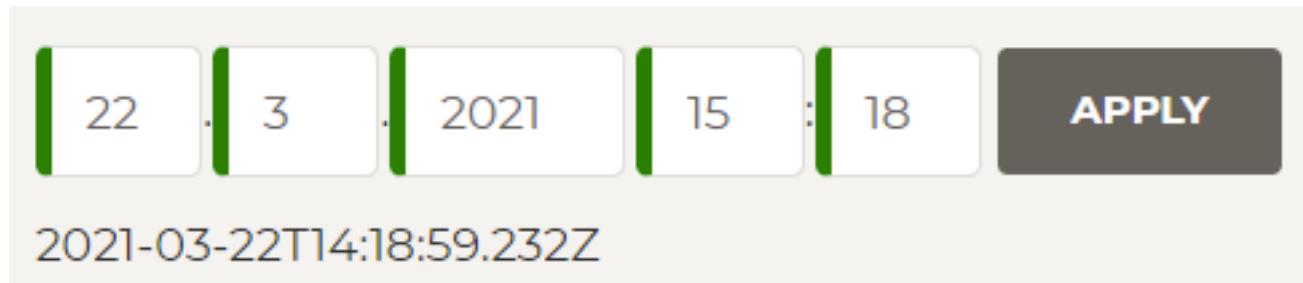
Case Study #3: Formatting/Parsing Dates



```
<input [(ngModel)]="date" appDate name="date">
```

DEMO

Case Study: DateControl



```
<app-date [(ngModel)]="date"></app-date>
```

LAB