ValueChanges in Angular Forms

7 Comments / 7 minutes of reading / March 9, 2023

← StatusChanges Angular Tutorial FormControl →

The ValueChanges is an event raised by the Angular forms whenever the value of the FormControl, FormGroup, or FormArray changes. It returns an observable so that you can subscribe to it. The observable get the latest value of the control. It allows us to track changes made to the value in real-time and respond to them. For example, we can use it to validate the value, calculate the computed fields, etc.

Table of Contents

How to use ValueChanges

Angular Forms has three building blocks. FormControl, FormGroup & FormArray. All of these controls extend the AbstractControl base class. The AbstractControl base class implements the ValueChanges event

We can subscribe to ValueChanges by getting the reference of the control and subscribing to it as shown below

```

this.reactiveForm.get("firstname").valueChanges.subscribe(x => {
   console.log('firstname value changed')
   console.log(x)
})

```

You can also subscribe to the top-level form as shown below.

```

this.reactiveForm.valueChanges.subscribe(x => {
    console.log('form value changed')
    console.log(x)
})

```

ValueChanges Example

Create a reactive form as shown below

```

reactiveForm = new FormGroup({
   firstname: new FormControl('', [Validators.required]),
   lastname: new FormControl(),
   address: new FormGroup({
     city: new FormControl(),
     street: new FormControl(),
     pincode: new FormControl()
   })
 })
```

ValueChanges of FormControl

You can subscribe to ValueChanges of a single FormControl as shown below. Here in selectedValue variable, we will get the latest value of the firstname . You can also retrieve the latest value of the firstname using this.reactiveForm.get("firstname").value

```

this.reactiveForm.get("firstname").valueChanges.subscribe(selectedValue => {
  console.log('firstname value changed')
  console.log(selectedValue)                              //latest value of firstname
  console.log(this.reactiveForm.get("firstname").value)   //latest value of firstname
})

```

ValueChanges shows the previous value

But, the top-level form is not yet updated at this point, hence this.reactiveForm.value still shows the previous value of the firstname .

The valueChanges event for the firstname fires immediately **after** the new values are updated but **before** the change is bubbled up to its parent. Hence the this.reactiveForm.value still shows the previous value.

```

this.reactiveForm.get("firstname").valueChanges.subscribe(selectedValue => {
  console.log('firstname value changed')
  console.log(selectedValue)
  console.log(this.reactiveForm.get("firstname").value)
  console.log(this.reactiveForm.value)   //still shows the old first name
})

```

You can work around this by waiting for the next tick using `setTimeout` as shown below.

```

this.reactiveForm.get("firstname").valueChanges.subscribe(selectedValue => {
  console.log('firstname value changed')
  console.log(selectedValue)
  console.log(this.reactiveForm.get("firstname").value)
  console.log(this.reactiveForm.value)    //shows the old first name

  setTimeout(() => {
    console.log(this.reactiveForm.value)   //shows the latest first name
  })

})

```

ValueChanges of FormGroup

The `ValueChanges` event of `FormGroup` or `FormArray` is fired, whenever the value of any of its child controls value changes. For Example, the following `ValueChanges` will fire even whenever the value of the *city*, *state* & *Pincode* changes.

```

this.reactiveForm.get("address").valueChanges.subscribe(selectedValue  => {
  console.log('address changed')
  console.log(selectedValue)
})

```

ValueChanges of Form

The following example show we can subscribe to the changes made to the entire form.

```

this.reactiveForm.valueChanges.subscribe(selectedValue  => {
  console.log('form value changed')
  console.log(selectedValue)
})

```

EmitEvent & ValueChanges

The ValueChanges event is fired even when the values of the control are changed programmatically. In some circumstances, you might not want to raise the ValueChanges event. To do that we can use the emitEvent: false

In the following example, the ValueChanges event is **not fired** at all, even though the value of the firstname is changed.

```

this.reactiveForm.get("firstname").setValue("", { emitEvent: false });

```

You can use emitEvent: false with the setValue, patchValue, markAsPending, disable, enable, updateValueAndValidity & setErrors methods.

OnlySelf & ValueChanges

When onlySelf: true the changes will only affect only this FormControl and change is **not** bubbled up to its parent. Hence the ValueChanges event of the parent FormGroup does not fire.

For Example, the following code will result in the ValueChanges of the firstname. but not of its parent (i.e. top-level form)

```

this.reactiveForm.get("firstname").setValue("", { onlySelf: true });

```

You can use the onlySelf: true with the setValue, patchValue, markAsUntouched, markAsDirty, markAsPristine, markAsPending, disable, enable, and updateValueAndValidity methods

Complete Source Code

[tabby title="reactive.component.ts"]

```

import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms'
import { timeout } from 'q';


@Component({
  templateUrl: './reactive.component.html',
})
export class ReactiveComponent implements OnInit {
  title = 'Reactive Forms';
```

```

reactiveForm = new FormGroup({
  firstname: new FormControl('', [Validators.required]),
  lastname: new FormControl(),
  address: new FormGroup({
    city: new FormControl(),
    street: new FormControl(),
    pincode: new FormControl()
  })
})

onSubmit() {
  console.log(this.reactiveForm.value);
}

ngOnInit() {

  this.reactiveForm.get("firstname").valueChanges.subscribe(selectedValue => {
    console.log('firstname value changed')
    console.log(selectedValue)
    console.log(this.reactiveForm.get("firstname").value)
    console.log(this.reactiveForm.value)

    setTimeout(() => {
      console.log(this.reactiveForm.value)
    })

  })

  this.reactiveForm.get("address").valueChanges.subscribe(selectedValue => {
    console.log('address changed')
    console.log(selectedValue)
  })

  this.reactiveForm.valueChanges.subscribe(selectedValue => {
    console.log('form value changed')
    console.log(selectedValue)
  })
}



setValue() {

  let contact = {
    firstname: "Rahul",
    lastname: "Dravid",
    address: {
      city: "Bangalore",
        street: "Brigade Road",
        pincode: "600070"
      }
    };

    this.reactiveForm.setValue(contact);
  }

  setAddress() {

    this.reactiveForm.get("address").setValue(
      {
        city: "Bangalore",
        street: "Brigade Road",
        pincode: "600070"
      }
    );
  }

  setFirstname() {
    this.reactiveForm.get("firstname").setValue("Saurav")
  }

  withoutOnlySelf() {
    this.reactiveForm.get("firstname").setValue("");
  }
  withOnlySelf() {
    this.reactiveForm.get("firstname").setValue("", { onlySelf: true });
  }

  withEmitEvent() {
    this.reactiveForm.get("firstname").setValue("Sachin");
  }
  withoutEmitEvent() {
    this.reactiveForm.get("firstname").setValue("", { emitEvent: false });
  }

  reset() {
    this.reactiveForm.reset();
  }

}

```

[tabby title=”reactive.component.html”]

```

<h3>{{title}}</h3>

<div style="float: left; width:50%;">

  <form [formGroup]="reactiveForm" (ngSubmit)="onSubmit()" novalidate>

    <p>
      <label for="firstname">First Name </label>
      <input type="text" id="firstname" name="firstname" formControlName="firstname">
      <label for="lastname">Last Name </label>
      <input type="text" id="lastname" name="lastname" formControlName="lastname">
    </p>

    <div formGroupName="address">

      <p>
        <label for="city">City</label>
        <input type="text" class="form-control" name="city" formControlName="city">
        <label for="street">Street</label>
        <input type="text" class="form-control" name="street" formControlName="street">
        <label for="pincode">Pin Code</label>
        <input type="text" class="form-control" name="pincode" formControlName="pincod
      </p>

    </div>


    <button>Submit</button>
    <div>
      <button type="button" (click)="setValue()">SetValue</button>
      <button type="button" (click)="setAddress()">Address</button>
      <button type="button" (click)="setFirstname()">First Name</button>
    </div>
    <div>
      <button type="button" (click)="withoutOnlySelf()">Without Only Self</button>
      <button type="button" (click)="withOnlySelf()">With Only Self</button>
    </div>
    <div>
      <button type="button" (click)="withouEmitEvent()">Without EmitEvent</button>
      <button type="button" (click)="withEmitEvent()">With EmitEvent</button>
    </div>

  </form>
</div>

<div style="float: right; width:50%;">

  <h3>Form Status</h3>
  <b>status : </b>{{reactiveForm.status}}
  <b>valid : </b>{{reactiveForm.valid}}
  <b>invalid : </b>{{reactiveForm.invalid}}
  <b>touched : </b>{{reactiveForm.touched}}
  <b>untouched : </b>{{reactiveForm.untouched}}
  <b>pristine : </b>{{reactiveForm.pristine}}
  <b>dirty : </b>{{reactiveForm.dirty}}
  <b>disabled : </b>{{reactiveForm.disabled}}
  <b>enabled : </b>{{reactiveForm.enabled}}


  <h3>Form Value</h3>
  {{reactiveForm.value |json}}

</div>

```

[tabbyending]

[tabby title="app.component.html"]

```

<h3>Angular ValueChanges Example</h3>

<ul>
  <li>
    <a [routerLink]="['/template']" routerLinkActive="router-link-active" >Template</a>
  </li>
  <li>
    <a [routerLink]="['/reactive']" routerLinkActive="router-link-active" >Reactive</a>
  </li>
```

```
</ul>

<router-outlet></router-outlet>

```

[tabby title="app.component.ts"]

```

import { Component} from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
}

```

[tabby title="app.module.ts"]

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { TemplateComponent } from './template-component';
import { ReactiveComponent } from './reactive.component';

@NgModule({
  declarations: [
    AppComponent,TemplateComponent,ReactiveComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

[tabbyending]

ValueChanges in Template Driven Forms

ValueChanges event can also be used in template-driven forms. All you need to do is to get the reference to the Form Model in the component as shown below.

```

@ViewChild('templateForm',null) templateForm: NgForm;

```

You can refer to the example code below

[tabby title="template-component.ts"]

```

import { Component, ViewChild, ElementRef, OnInit, OnDestroy } from '@angular/core';
import { NgForm } from '@angular/forms';


@Component({
  templateUrl: './template.component.html',
})
export class TemplateComponent implements OnInit {

  title = 'Template driven forms';

  @ViewChild('templateForm',null) templateForm: NgForm;

  contact: contact;

  onSubmit() {
    console.log(this.templateForm.value);
  }

  ngOnInit() {

    setTimeout(() => {

      this.templateForm.control.get("firstname").valueChanges.subscribe(selectedValue =
        console.log('firstname value changed')
        console.log(selectedValue)
        console.log(this.templateForm.control.get("firstname").value)
        console.log(this.templateForm.control.value)

        setTimeout(() => {
          console.log(this.templateForm.control.value)
        })
      })

      this.templateForm.control.get("address").valueChanges.subscribe(selectedValue =>
        console.log('address changed')
        console.log(selectedValue)
      })

      this.templateForm.valueChanges.subscribe(selectedValue => {
        console.log('form value changed')
        console.log(selectedValue)
      })

    });

  }


 setValue() {
    let contact = {
      firstname: "Rahul",
      lastname: "Dravid",
      address: {
        city: "Bangalore",
        street: "Brigade Road",
        pincode: "600070"
      }
    };

    this.templateForm.setValue(contact);
  }

  setAddress() {
    let address= {
      city: "Bangalore",
      street: "Brigade Road",
      pincode: "600070"
    };

    this.templateForm.control.get("address").setValue(address);

  };

  setFirstname() {
    this.templateForm.control.get("firstname").setValue("Saurav")
  }


  withoutOnlySelf() {
    this.templateForm.control.get("firstname").setValue("");
  }
  withOnlySelf() {
    this.templateForm.control.get("firstname").setValue("", { onlySelf: true });
  }

  withouEmitEvent() {
    this.templateForm.control.get("firstname").setValue("Sachin");
  }
  withEmitEvent() {
    this.templateForm.control.get("firstname").setValue("", { emitEvent: false });
  }

  reset() {
    this.templateForm.reset();
  }

}


export class contact {
  firstname:string;
  lastname:string;
  gender:string;
  email:string;
  isMarried:boolean;
  country:string;
  address: {
    city:string;
    street:string;
    pincode:string;
  }
}

```

[tabby title="template-component.html"]

```

<h3>{{title}}</h3>

<div style="float: left; width:50%;">
  <form #templateForm="ngForm" (ngSubmit)="onSubmit(templateForm)">

    <p>
      <label for="firstname">First Name </label>
      <input type="text" id="firstname" name="firstname" #fname="ngModel" ngModel>

    </p>
    <p>
      <label for="lastname">Last Name </label>
      <input type="text" id="lastname" name="lastname" ngModel>
    </p>

    <div ngModelGroup="address">

      <p>
        <label for="city">City</label>
        <input type="text" id="city" name="city" ngModel>
        <label for="street">Street</label>
        <input type="text" id="street" name="street" ngModel>
        <label for="pincode">Pin Code</label>
        <input type="text" id="pincode" name="pincode" ngModel>
      </p>

    </div>

    <button>Submit</button>
    <div>
      <button type="button" (click)="setValue()">SetValue</button>
      <button type="button" (click)="setAddress()">Address</button>
      <button type="button" (click)="setFirstname()">First Name</button>
    </div>
    <div>
      <button type="button" (click)="withoutOnlySelf()">Without Only Self</button>
      <button type="button" (click)="withOnlySelf()">With Only Self</button>
    </div>
    <div>
      <button type="button" (click)="withouEmitEvent()">Without EmitEvent</button>
      <button type="button" (click)="withEmitEvent()">With EmitEvent</button>
    </div>

  </form>
</div>

<div style="float: right; width:50%;">
  <h3>Form Status</h3>
  <b>status : </b>{{templateForm.status}}
  <b>valid : </b>{{templateForm.valid}}
  <b>invalid : </b>{{templateForm.invalid}}
  <b>touched : </b>{{templateForm.touched}}
  <b>untouched : </b>{{templateForm.untouched}}
  <b>pristine : </b>{{templateForm.pristine}}
  <b>dirty : </b>{{templateForm.dirty}}
  <b>disabled : </b>{{templateForm.disabled}}
  <b>enabled : </b>{{templateForm.enabled}}

  <h3>Form Value</h3>
  {{templateForm.value | json }}
</div>>

```

[tabbyending]

Summary

In this tutorial, we learned how to make use of ValueChanges in Angular Forms. The ValueChanges event is fired whenever the value of the FormControl, FormGroup, or FormArray changes. It is observable and we can subscribe to it. We can then use it to validate the forms. update the computed fields, etc. The ValueChanges event does not fire depending on how we set `emitEvent` or `onlySelf`, when updating the value and validity of the form controls.