StatusChanges in Angular Forms

Leave a Comment / 7 minutes of reading / March 9, 2023

← SetValue & PatchValue | Angular Tutorial | ValueChanges →

The StatusChanges is an event raised by the Angular forms whenever the Angular **calculates the validation status** of the FormControl, FormGroup or FormArray. It returns an observable so that you can subscribe to it. The observable gets the **latest status** of the control. The Angular runs the validation check on every change made to the control. It also generates a list of validation errors in which case the status becomes INVALID . If there are no errors, then the status becomes VALID

Table of Contents

How to use StatusChanges

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

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

```
this.reactiveForm.get("firstname").statusChanges.subscribe(newStaus => {
   console.log('firstname status changed')
   console.log(newStaus)
})
```

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

```
this.reactiveForm.statusChanges.subscribe(newStaus => {
   console.log('form Status changed event')
   console.log(newStaus)
})
```

StatusChanges 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()
   })
 })
```

```
11
```

StatusChanges of FormControl

You can subscribe to `StatusChanges` of a single `FormControl` as shown below. Here in the newStatus variable, we will get the latest status of the firstname. You can also retreive the latest status of the firstname using `this.reactiveForm.get("firstname").status`

```
1
2 this.reactiveForm.get("firstname").statusChanges.subscribe(newStatus => {
3   console.log('firstname status changed')
4   console.log(newStatus)                          //latest status
5   console.log(this.reactiveForm.get("firstname").status)  //latest status
6 })
7
```

But, the top-level form is not yet updated at this point, hence `this.reactiveForm.status` still shows the old status of the firstname and also the form.

The `statusChanges` event for the firstname fires immediately **after** the new status is updated but **before** the change is bubbled up to its parent. Hence the `this.reactiveForm.status` still shows the old status.

```
1
2 this.reactiveForm.get("firstname").statusChanges.subscribe(newStatus=> {
```

```
  console.log('firstname status changed')
  console.log(newStatus)                          //latest status
  console.log(this.reactiveForm.get("firstname").status)  //latest status
  console.log(this.reactiveForm.status)                 //Previous status
})

```

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

```

this.reactiveForm.get("firstname").statusChanges.subscribe(newStatus=> {
  console.log('firstname status changed')
  console.log(newStatus)                          //latest status
  console.log(this.reactiveForm.get("firstname").status)  //latest status
  console.log(this.reactiveForm.status)                 //Previous status

  setTimeout(() => {
    console.log(this.reactiveForm.status)               //latest status
  })

})

```

StatusChanges of FormGroup

The `StatusChanges` event of `FormGroup` or `FormArray` is fired, whenever the status of any of its child controls are calculated. For Example, the following `StatusChanges` will fire even whenever the status of the *city*, *state* & *pincode* are calculated.

```

this.reactiveForm.get("address").statusChanges.subscribe(newStaus => {
  console.log('address status changed')
  console.log(newStaus)
})

```

StatusChanges of Form

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

```

this.reactiveForm.statusChanges.subscribe(newStaus => {
  console.log('form status changed')
  console.log(newStaus)
})

```

emitEvent & StatusChanges

The `statusChanges` event is fired even when the angular calculates the status of the control either via UI or programmatically. In some circumstances, you might not want to raise the `statusChanges` event. To do that we can use the `emitEvent: false`

In the following example, the `statusChanges` event is **not fired** at all, even though the value of the firstname is changed making it and the form `INVALID`.

```

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

```

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

onlySelf & StatusChanges

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

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

```
1
2 this.reactiveForm.get("firstname").setValue("", { onlySelf: true });
3
```

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"]

```
1
2 import { Component, OnInit } from '@angular/core';
3 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").statusChanges.subscribe(newStatus=> {
      console.log('firstname status changed')
      console.log(newStatus)
      console.log(this.reactiveForm.get("firstname").status)
      console.log(this.reactiveForm.status)

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

    })

    this.reactiveForm.get("address").statusChanges.subscribe(newStatus=> {
      console.log('address status changed')
      console.log(newStatus)
    })

    this.reactiveForm.statusChanges.subscribe(newStatus=> {
      console.log('form status changed')
      console.log(newStatus)
    })
  }

  setValue() {
```

```
53
54    let contact = {
55      firstname: "Rahul",
56      lastname: "Dravid",
57      address: {
58        city: "Bangalore",
59        street: "Brigade Road",
60        pincode: "600070"
61      }
62    };
63
64    this.reactiveForm.setValue(contact);
65  }
66
67  setAddress() {
68
69    this.reactiveForm.get("address").setValue(
70      {
71        city: "Bangalore",
72        street: "Brigade Road",
73        pincode: "600070"
74      }
75    );
76  }
77
78  setFirstname() {
79    this.reactiveForm.get("firstname").setValue("Saurav")
80  }
81
82  withoutOnlySelf() {
83    this.reactiveForm.get("firstname").setValue("");
84  }
85  withOnlySelf() {
86    this.reactiveForm.get("firstname").setValue("", { onlySelf: true });
87  }
88
89  withEmitEvent() {
90    this.reactiveForm.get("firstname").setValue("Sachin");
91  }
92  withoutEmitEvent() {
93    this.reactiveForm.get("firstname").setValue("", { emitEvent: false });
94  }
95
96  reset() {
97    this.reactiveForm.reset();
98  }
99
100 }
```

```
101
```

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

```
1
2  <h3>{{title}}</h3>
3
4  <div style="float: left; width:50%;">
5
6   <form [formGroup]="reactiveForm" (ngSubmit)="onSubmit()" novalidate>
7
8     <p>
9       <label for="firstname">First Name </label>
10      <input type="text" id="firstname" name="firstname" formControlName="firstname">
11      <label for="lastname">Last Name </label>
12      <input type="text" id="lastname" name="lastname" formControlName="lastname">
13    </p>
14
15    <div formGroupName="address">
16
17      <p>
18        <label for="city">City</label>
19        <input type="text" class="form-control" name="city" formControlName="city">
20        <label for="street">Street</label>
21        <input type="text" class="form-control" name="street" formControlName="street">
22        <label for="pincode">Pin Code</label>
23        <input type="text" class="form-control" name="pincode" formControlName="pincod
24      </p>
25
26    </div>
27
28    <button>Submit</button>
29
30  </form>
31
32  <div>
33    <button type="button" (click)="setValue()">SetValue</button>
34    <button type="button" (click)="setAddress()">Address</button>
35    <button type="button" (click)="setFirstname()">First Name</button>
36  </div>
37  <div>
38    <button type="button" (click)="withoutOnlySelf()">Without Only Self</button>
39    <button type="button" (click)="withOnlySelf()">With Only Self</button>
40  </div>
41  <div>
42    <button type="button" (click)="withouEmitEvent()">Without EmitEvent</button>
```

```
    <button type="button" (click)="withEmitEvent()">With EmitEvent</button>
  </div>


</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 StatusChanges 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]

StatusChanges in Template Driven Forms

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

```
1
2 @ViewChild('templateForm',null) templateForm: NgForm;
3
```

You can refer to the example code below

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

```
1
2 import { Component, ViewChild, ElementRef, OnInit, OnDestroy } from '@angular/core';
3 import { NgForm } from '@angular/forms';
4
5
6 @Component({
7   templateUrl: './template.component.html',
8 })
9 export class TemplateComponent implements OnInit {
10
11   title = 'Template driven forms';
12
```

```
13  @ViewChild('templateForm',null) templateForm: NgForm;
14
15  contact: contact;
16
17  onSubmit() {
18    console.log(this.templateForm.value);
19  }
20
21  ngOnInit() {
22
23    setTimeout(() => {
24
25      this.templateForm.control.get("firstname").statusChanges.subscribe(newStatus=> {
26        console.log('firstname status changed')
27        console.log(newStatus)
28        console.log(this.templateForm.control.get("firstname").status)
29        console.log(this.templateForm.control.status)
30
31        setTimeout(() => {
32          console.log(this.templateForm.control.status)
33        })
34
35      })
36
37      this.templateForm.control.get("address").statusChanges.subscribe(newStatus => {
38        console.log('address status changed')
39        console.log(newStatus)
40      })
41
42      this.templateForm.control.statusChanges.subscribe(newStatus=> {
43        console.log('form status changed')
44        console.log(newStatus)
45      })
46
47    });
48
49
50
51
52  }
53
54
55
56  setValue() {
57    let contact = {
58      firstname: "Rahul",
59      lastname: "Dravid",
60      address: {
61        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>

 </form>

 <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>



</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 `StatusChanges` in Angular Forms. The `StatusChanges` event is fired whenever the angular calculates the validity status of the FormControl, FormGroup or FormArray. It is an observable and we can subscribe to it.