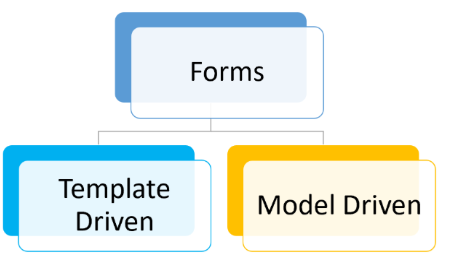
**Angular Forms Module**

Angular forms module provides all the above [services](https://www.tektutorialshub.com/angular/angular-services/) out of the box. It binds the form field to the [Angular component](https://www.tektutorialshub.com/angular/angular-component/) class. It tracks changes made to the form fields so that we can respond accordingly. The Angular forms provide the [built-in validators](https://www.tektutorialshub.com/angular/angular-reactive-forms-validation/) to validate the inputs

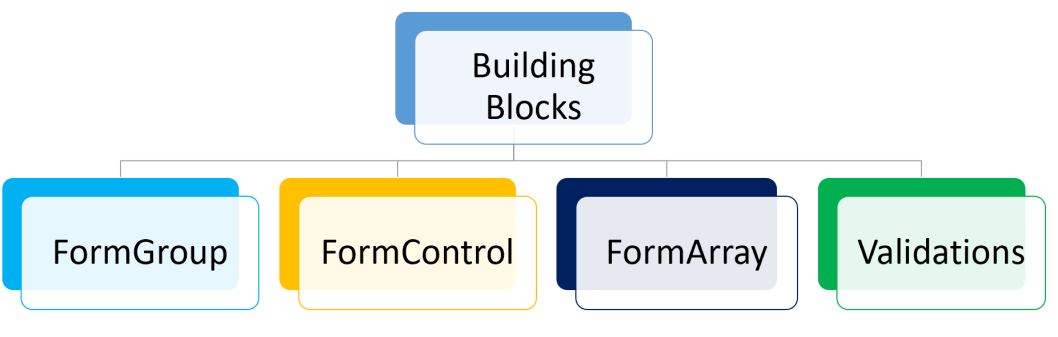
. You can create your own [custom validator.](https://www.tektutorialshub.com/angular/custom-validator-in-angular-reactive-form/) It presents the validation errors to the user. Finally, it encapsulates all the input fields into an object structure when the user submits the form.

Angular takes two approaches to build the forms. In this article, you will learn about different ways of creating forms in Angular. Angular provides two ways to create forms – Template Driven and Model-Driven.



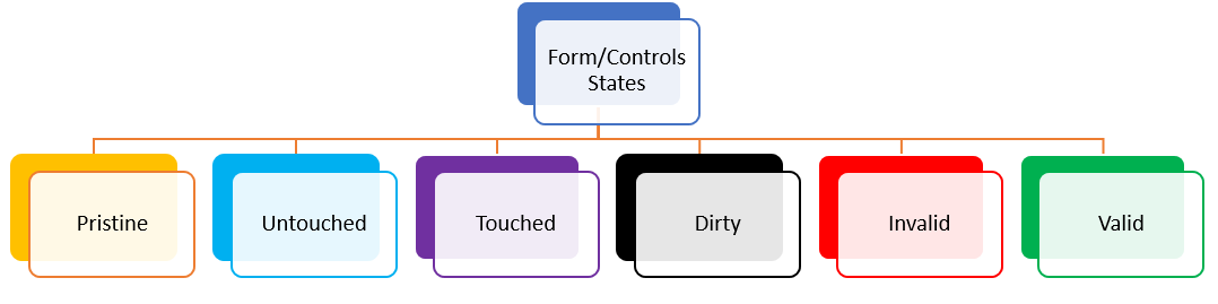
## Angular Form Building Blocks

The basic building blocks of an Angular form are FormsGroup, FormControl, FormArray, and Validations.



## Angular Form and Form Controls States

Angular Forms and their controls do change in their states as the user starts interaction with the form input controls. This state transition is helpful to get the information about the form and its input control state. Based upon the get information you can show or hide error messages and even you can check the validity of the form.



### States Transition

1. untouched => touched
2. invalid => valid
3. pristine => dirty

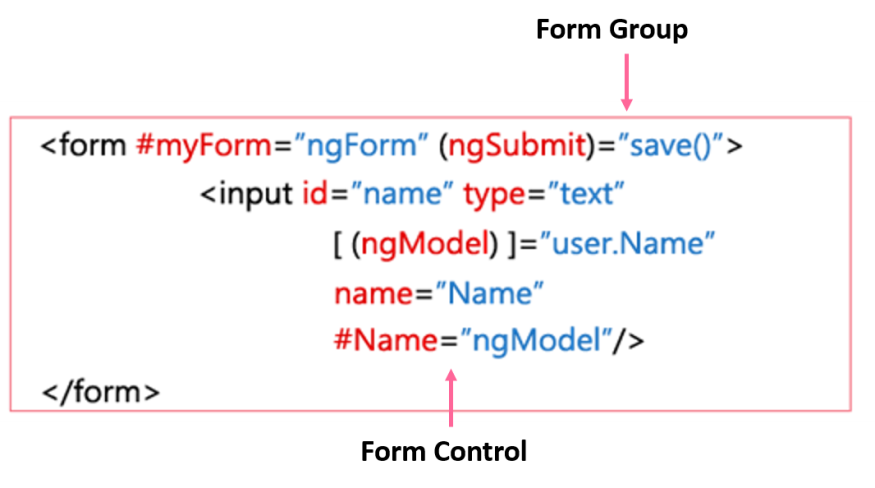
### Template Driven Form

In template-driven forms, we don't create Angular form control objects but Angular directives create them for us using the information from our data binding configuration.

We don't have to push and pull data values because Angular handles that for you using the ngModel directive and Angular updates the mutable data model according to user changes as they happen while using the form.

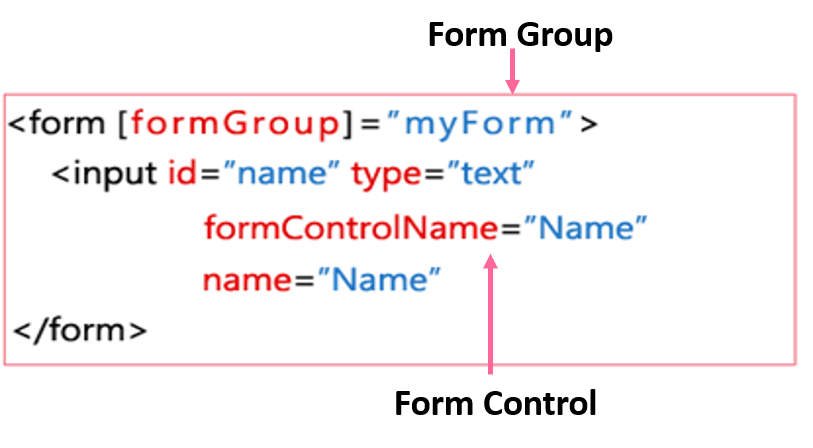
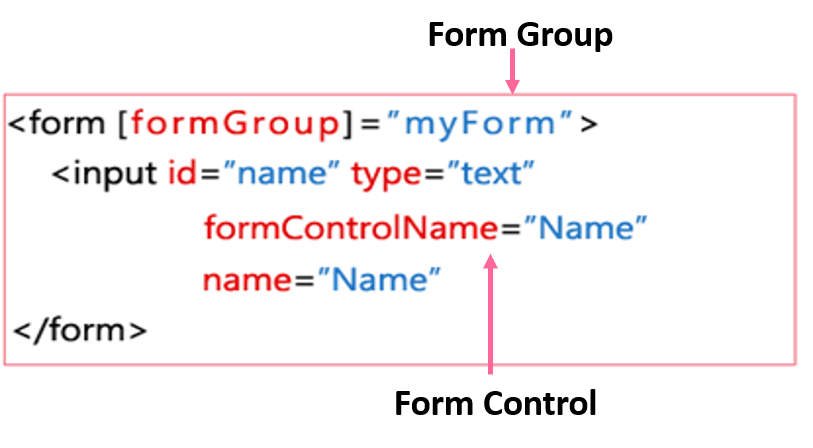
Template-driven Form is set up and configured in HTML Code. Template-driven Form is easy to use and suits simple form.

Template-driven Form uses directives (ngForm, ngModel) and reference name(#refName) for creating forms.

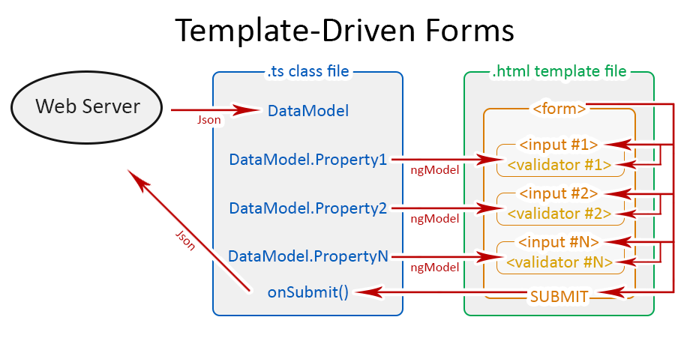


## Model-Driven Form

The model-driven form is set up and configured in the component class. It is based on a reactive style of programming where you use the underlying APIs FormControl and FormGroup that track the value and validation status. Reactive forms offer ease of testing and validation.



**Template-driven forms approach**

In [Template-driven approach](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) is the easiest way to build the Angular forms. The logic of the form is placed in the Html template. The approach here is similar to what we did in AngularJs. 

**Model-driven forms approach**

In [Reactive Forms](https://www.tektutorialshub.com/angular/angular-reactive-forms/) or [Model-driven](https://www.tektutorialshub.com/angular/angular-reactive-forms/) approach, the logic of the form is defined in the component as an object. The Model-driven approach has more benefits as it makes the testing of the component easier.

In this approach, the representation of the form is created in the component class. This form model is then bound to the HTML elements. it is done using the special markups.

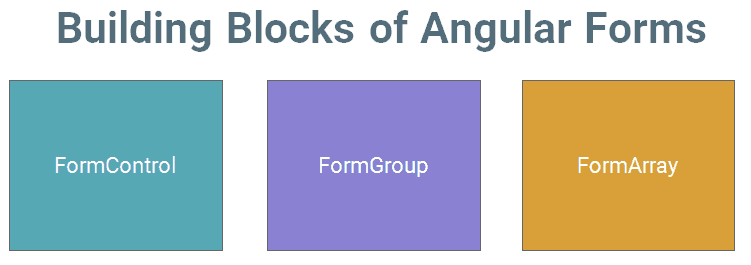
[Template-driven forms](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) in Angular allows us to create sophisticated looking forms easily without writing any javascript code. The model-driven forms are created in component class, where Form fields are created as properties of our component class.  This makes it easier to test.

Here is the list of tutorials on how to build a form using the template-driven approach & reactive or model-driven approach

**Learn more**

**Building Blocks of Angular Forms**

The Angular Forms module consists of three Building blocks, irrespective of whether you are using [Template-driven](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) or [Reactive forms](https://www.tektutorialshub.com/angular/angular-reactive-forms/) approach.



**FormControl**

A [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) represents a single input field in an Angular form.

Consider a simple Text input box

|  |  |
| --- | --- |
| 1  2  3 | **First Name : <input type="text" name="firstname" />** |

As a developer, you would like to know the current value in the Text box. You would also be like to know if the value is valid or not

. If the user has changed the value(dirty) or is it unchanged. You would like to be notified when the user changes value.

The [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) is an object that encapsulates all this information related to the single input element. It Tracks the value and validation status of each of these control

The [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) is just a class. **A [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) is created for each form field.** We can refer them in our component class and inspect its properties and methods

You can use [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) to set the value of the Form field, find the status of form field like (valid/invalid, pristine/dirty, touched/untouched ) etc & add validation rules to it.

The above input field is created using the [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) as shown below

|  |  |
| --- | --- |
| **1**  **2**  **3** | **let firstname= new FormControl(); //Creating a FormControl in a Reactive forms** |

Then, you can retrieve the current value in the input field using the value property

|  |  |
| --- | --- |
|  | firstname.value   //Returns the value of the first name field |

You can check the validation status of the First Name element as shown below

|  |  |
| --- | --- |
| 6 | firstname.errors      // returns the list of errors  firstname.dirty       // true if the value has changed (dirty)  firstname.touched     // true if input field is touched  firstname.valid       // true if the input value has passed all the validation |

**FormGroup**

[FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) is a collection of [FormControls](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) . Each [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) is a property in a [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/). with the control name as the key.

Often forms have more than one field. It is helpful to have a simple way to manage the Form controls together.

Consider the following Form. we have three input fields street, city & Pincode.

|  |  |
| --- | --- |
|  | city : <input type="text" name="city" >  Street : <input type="text" name="street" >  PinCode : <input type="text" name="pincode" > |

All of the above input fields are represented as the separate [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/). If we wanted to check the validity of our form, we have to check the validity of each and every [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) for validity.

Imagine Form having large no of fields. It is cumbersome to loop over large no of [FormControls](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) and check for validity.

[FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) solve’s this issue by providing a wrapper interface around a collection of [FormControls](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) A [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) tracks the status of each child [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) and aggregates the values into one object. with each control name as the key

We can group these input fields under the group address as shown below

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **let address= new FormGroup({**  **street : new FormControl(""),**  **city : new FormControl(""),**  **pinCode : new FormControl("")**  **})** |

**In the above example, the address is our FormGroup, consisting of 3 Form Controls city, street, and Pincode.**

**Now we can check the validity of the entire group together.**

**For example, if the state is invalid, then the address [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) returns the invalid state.**

You can read the value of an address using the value method, which returns the JSON object as shown below

|  |
| --- |
| address.value |

The Return value

|  |  |
| --- | --- |
|  | address {      street :"",      city:"",      Pincode:""  } |

You can access child control as

|  |  |
| --- | --- |
|  | address.get("street") |

Check the Validation status as follows

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | address.errors     // returns the list of errors  address.dirty      // true if the value of one of the child control has changed (dirty)  address.touched    // true if one of the child control is touched  address.valid      // true if all the child controls passed the validation |

A typical Angular Form can have more than one [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/). A [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) can also contain another [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/).

The Angular form is itself a [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/)

**FormArray**

[FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/) is an array of form controls. It is similar to [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) except for one difference. In [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) each [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) is a property with the control name as the key. In [FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/)is an array of form controls.

We define the [FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/)as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | contactForm = new FormGroup( {      name: new FormControl(''),      cities:new FormArray([        new FormControl('Mumbai'),        new FormControl('Delhi')      ])    }); |

You can get the reference to the cities from the contactForm.get method

|  |  |
| --- | --- |
| 1  2  3  4  5 | cities() :FormArray {      return this.contactForm.get("cities") as FormArray    } |

Check the Validation status as follows

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | cities.errors     // returns the list of errors  cities.dirty      // true if the value of one of the child control has changed (dirty)  cities.touched    // true if one of the child control is touched  cities.valid      / |

## What is Template-driven form?

In Template Driven Forms we specify behaviors/validations using directives and attributes in our Html template and let it work behind the scenes

. **All things happen in Templates hence very little code is required in the component class.** This is different from the reactive forms, where we define the logic and controls in the component class.

The Template-driven forms

1. The form is set up using **ngForm directive**
2. controls are set up using the **ngModel directive**
3. ngModel also provides the two-way data binding
4. The Validations are configured in the template via directives

Template-driven forms are

1. Contains little code in the component class
2. Easier to set up

While they are

1. Difficult to add controls dynamically
2. Unit testing is a challenge

## Create the Example Application

Use ng new to create a new application

|  |  |
| --- | --- |
| Step:1 | ng new tdf  --routing=true --style=css |

Run ng serve and verify if everything is installed correctly.

### Import FormsModule

**To work with Template-driven forms, we must import the FormsModule. We usually import it in root module app.module.ts**

or in a [shared module](https://www.tektutorialshub.com/angular/angular-folder-structure-best-practices/#shared-module). The FormsModule contains all the form directives and constructs for working with forms

Open the app.module.ts and add the import { FormsModule } from '@angular/forms'; to it.

And also add the FormsModule to the imports metadata property array

|  |  |
| --- | --- |
|  | import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { FormsModule } from '@angular/forms';        //import FormsModule    import { AppRoutingModule } from './app-routing.module';  import { AppComponent } from './app.component';    @NgModule({    declarations: [      AppComponent    ],    imports: [      BrowserModule,      AppRoutingModule,      FormsModule                    //Add in Imports Array    ],    providers: [],    bootstrap: [AppComponent]  })  export class AppModule { } |

### HTML Form

The first task is to build the template. The following is a regular HTML form. We enclose it in a <form> tag. We have included two text input (FirstName & LastName), a email (email), a radio button (gender), a checkbox (isMarried), and a select list (country). These are form elements.

|  |
| --- |
| **<form>**    **<p>**  **<label for="firstname">First Name</label>**  **<input type="text" id="firstname" name="firstname">**  **</p>**    **<p>**  **<label for="lastname">Last Name</label>**  **<input type="text" id="lastname" name="lastname">**  **</p>**    **<p>**  **<label for="email">Email </label>**  **<input type="text" id="email" name="email">**  **</p>**    **<p>**  **<label for="gender">Geneder</label>**  **<input type="radio" value="male" id="gender" name="gender"> Male**  **<input type="radio" value="female" id="gender" name="gender"> Female**  **</p>**    **<p>**  **<label for="isMarried">Married</label>**  **<input type="checkbox" id="isMarried" name="isMarried">**  **</p>**    **<p>**  **<label for="country">country </label>**  **<select name="country" id="country">**  **<option selected="" value=""></option>**  **<option [ngValue]="c.id" \*ngFor="let c of countryList">**  **{{c.name}}**  **</option>**  **</select>**  **</p>**    **<p>**  **<button type="submit">Submit</button>**  **</p>**    **</form>** |

**Component Class**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import { Component } from '@angular/core';    @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {    title = 'Template driven forms';      countryList:country[] = [      new country("1", "India"),      new country('2', 'USA'),      new country('3', 'England')    ];  }    export class country {    id:string;    name:string;      constructor(id:string, name:string) {      this.id=id;      this.name=name;    }  } |

### ngForm

Once, we have a form with few form elements, the angular automatically converts it into a Template-driven form. This is done by the ngForm directive.

The ngForm directive is what makes the Angular template-driven forms work. But we do not need to add it explicitly. Angular adds it automatically

When we include FormsModule, the Angular is going to look out for any <form> tag in our HTML template. Angular does this via ngForm [directive](https://www.tektutorialshub.com/angular/angular-directives/). ngForm directive automatically detects the <form> tag and automatically binds to it. You do not have to do anything on your part to invoke and bind the ngForm directive.

The ngForm does the following

1. Binds itself to the <Form> directive
2. Creates a top-level FormGroup instance
3. CreatesFormControl instance for each of child control, which has ngModel directive.
4. CreatesFormGroup instance for each of the  NgModelGroup directive.

We can export the ngForm instance into a local template variable using ngForm as the key (ex: #contactForm="ngForm"). This allows us to access the many properties and methods of ngForm using the template variable contactForm

Hence, update the form element as shown below.

|  |  |
| --- | --- |
|  | Step:2 <form #contactForm="ngForm"> |

### FormControl

The FormControl is the basic building block of the [Angular Forms](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/). It represents a single input field in an [Angular form](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/)**.** The [Angular Forms Module](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/#angular-forms-module) binds the input element to a FormControl. We use the FormControl instance to track the value, user interaction and validation status of an individual form element. Each individual Form element is a FormControl

We have six form elements in our HTML template. They are firstName, lastname, email, gender, isMarried & country. We need to bind them to FormControl instance. We do this by using the ngModel directive. Add the ngModel directive to each control as shown below.

|  |  |
| --- | --- |
|  | <input type="text" name="firstname" ngModel> |

ngModel will use the name attribute to create the FormControl instance for each of the Form field it is attached.

### Submit Form

Now have the template ready, except for the final piece i.e submitting data to the component.

We use the ngSubmit event, to submit the form data to the component class. We use the [event binding](https://www.tektutorialshub.com/angular/angular-data-binding/) (parentheses) to bind ngSubmit to OnSubmit method in the component class. When the user clicks on the submit button, the ngSubmit event will fire

|  |  |
| --- | --- |
|  | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)"> |

We are passing the local template variable contactForm in onSubmit method. contactForm holds the reference to the ngForm directive. We can use this in our component class to extract the data from the form fields.

### Final Template

Our final template is as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">      <p>      <label for="firstname">First Name</label>      <input type="text" name="firstname" ngModel>    </p>      <p>      <label for="lastname">Last Name</label>      <input type="text" name="lastname" ngModel>    </p>      <p>      <label for="email">Email </label>      <input type="text" id="email" name="email" ngModel>    </p>      <p>      <label for="gender">Geneder</label>      <input type="radio" value="male" name="gender" ngModel> Male      <input type="radio" value="female" name="gender" ngModel> Female    </p>      <p>      <label for="isMarried">Married</label>      <input type="checkbox" name="isMarried" ngModel>    </p>      <select name="country" ngModel>      <option [ngValue]="c.id" \*ngFor="let c of countryList">        {{c.name}}      </option>    </select>      <p>      <button type="submit">Submit</button>    </p>    </form> |

### Receive Form Data

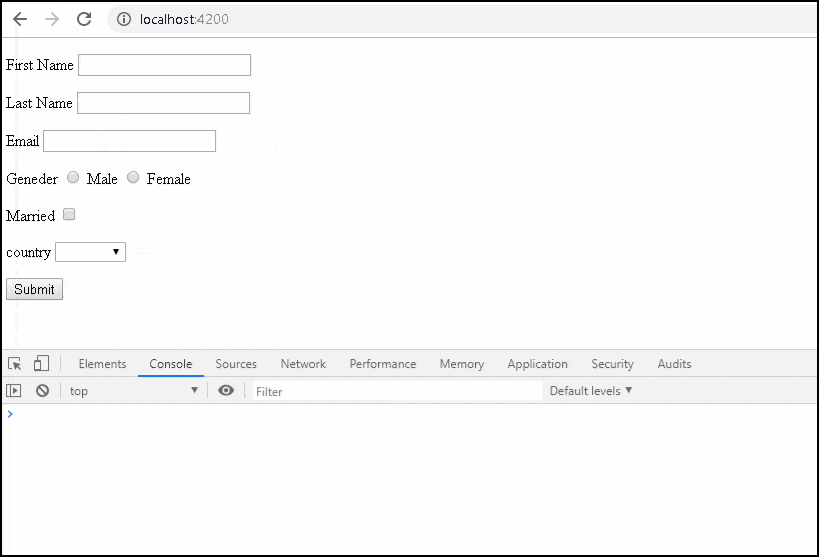
We need to receive the data in component class from our form. To do this we need to create the onSubmit method in our component class. The submit method receives the reference to the ngForm directive, which we named is as contactForm. The contactForm exposes the value method which returns the form fields as a Json object.

|  |  |
| --- | --- |
| 1  2  3  4  5 | onSubmit(contactForm) {      console.log(contactForm.value);    } |

You can print the value to the console using the console.log(contactForm.value)

Run the code now and enter some data into the form. Open the Developer Console in your browser and check the output, when you submit the data.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | country: "1"  firstname: "Sachin"  email:"sachin@gmail.com"  gender: "male"  isMarried: true  lastname: "Tendulkar" |

Angular template-driven forms in Action

## Local Variable

We can assign the ngForm,FormControl or FormGroup instance to a template local variable. This allows us to check the status of the form like whether the form is valid, submitted, and value of the form elements, etc

### ngForm

We have access to the ngForm instance via the local template variable #contactForm.

|  |  |
| --- | --- |
| 1  2  3 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)"> |

Now, we can make use of some of the properties & methods to know the status of form. For Example

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <p>    <button type="submit">Submit</button>  </p>    <pre>Value : {{contactForm.value | json }} </pre>  <pre>Valid : {{contactForm.valid}} </pre>  <pre>Touched : {{contactForm.touched  }} </pre>  <pre>Submitted : {{contactForm.submitted  }} </pre> |

value: The value property returns the object containing the value of every FormControl  
valid: Returns true if the form is Valid else returns false.  
touched: True if the user has entered a value in at least in one field.  
submitted: Returns true if the form is submitted. else false.

### FormControl

Similarly, we can also get access to the FormControl instance by assigning the ngModel to a local variable as shown below

|  |  |
| --- | --- |
| 1  2  3 | <input type="text" name="firstname" #fname="ngModel" ngModel> |

Now, the variable #fname holds the reference to the firstname FormControl. We can then access the properties of FormControl like value, valid, isvalid, tocuhed etc

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <p>    <label for="firstname">First Name </label>    <input type="text" name="firstname" #fname="ngModel" ngModel>  </p>    <pre>Value    : {{fname.value}} </pre>  <pre>valid    : {{fname.valid}} </pre>  <pre>invalid  : {{fname.invalid}} </pre>  <pre>touched  : {{fname.touched}} </pre |

value: Returns the current value of the control  
valid: Returns true if the value is Valid else false  
invalid: True if the value is invalid else false  
touched: Returns true if the value is entered in the element

## Nested FormGroup

The FormGroup is a collection of FormControl. It can also contain other FormGroup's.

The ngForm directive creates the top Level FormGroup behind the scene, when we use the <Form> directive.

|  |  |
| --- | --- |
| 1  2  3 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)"> |

We can add new [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) using the ngModelGroup directive. Let us add street, city & Pincode form controls and group them under the address FormGroup

All you need to do is to enclose the fields inside a div element with ngModelGroup directive applied on it as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <div ngModelGroup="address">        <p>        <label for="city">City</label>        <input type="text" name="city" ngModel>      </p>        <p>        <label for="street">Street</label>        <input type="text" name="street" ngModel>      </p>      <p>        <label for="pincode">Pin Code</label>        <input type="text" name="pincode" ngModel>      </p>    </div> |

Run the App and submit. The resultant object is as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | Value : {    "firstname": "Sachin",    "lastname": "Tendulkar",    "email":"sachin@gmail.com"    "gender": "male",    "isMarried": true,    "country": "1",    "address": {      "city": "Mumbai",      "street": "Fashin Street",      "pincode": "400600"    }  } |

## Setting the Initial Value

The form is usually pre-filled with some default data. In the case of editing, we have to show the user the current data..

## Validating the Form

Validating the form is another important task. We have covered it in Validation in template-driven form tutorial.

## Summary

**Angular Template-driven Forms** is simpler compared to the reactive forms. The FormsModule is imported first. Then we create the HTML form. The Angular detects the <form> tag and converts the form to the Angular Form. ngModel directive added to each form element, which converts them to FormControl. Finally, submit event is subscribed via event binding.

# How to set value in template-driven forms in Angular

[1 Comment](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#comments) / 8 minutes of reading

[**Angular Template Driven Forms**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/)

[**Reactive Forms in Angular**](https://www.tektutorialshub.com/angular/angular-reactive-forms/)

In this tutorial, we will learn how to set value in [template-driven forms in Angular](https://www.tektutorialshub.com/angular/angular-template-driven-forms/). We will learn how to set the default or initial value to form controls, dynamically set values, reset the value of the form, etc. Learn how to set the value of individual FormControl or a FormGroup and nested FormGroup

We have covered [how to create template-driven forms in the angular tutorial](https://www.tektutorialshub.com/angular/angular-template-driven-forms/). We will continue from there and in this tutorial, we will show you

Applies to: Angular 2 to the latest edition of i.e. Angular 8. Angular 9, Angular 10, Angular 11, Angular 12, Angular 13

**Table of Contents**

* [Template](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#template)
* [Set value in template-driven forms](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#set-value-in-template-driven-forms)
  + [Two-way data binding](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#two-way-data-binding)
    - [Set the default/initial value](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#set-the-defaultinitial-value)
    - [Set the value individually or dynamically](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#set-the-value-individually-or-dynamically)
    - [Reset form](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#reset-form)
  + [Template reference variable](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#template-reference-variable)
    - [Set the default or initial value](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#set-the-default-or-initial-value)
    - [Set the value individually or dynamically](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#set-the-value-individually-or-dynamically-2)
    - [Reset values](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#reset-values)
    - [Set Default Value](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#set-default-value)
    - [patch value](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#patch-value)
    - [Set value of nested FormGroup](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#set-value-of-nested-formgroup)
* [Summary](https://www.tektutorialshub.com/angular/how-to-set-value-in-template-driven-forms-in-angular/#summary)

## Template

The following is the app.component.html from the [angular template-driven forms tutorial](https://www.tektutorialshub.com/angular/angular-template-driven-forms/).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" ngModel>    </p>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" ngModel>    </p>      <p>      <label for="email">Email </label>      <input type="text" id="email" name="email"  ngModel>    </p>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" ngModel> Male      <input type="radio" value="female" id="gender" name="gender" ngModel> Female    </p>      <p>      <label for="isMarried">Married </label>      <input type="checkbox" id="isMarried" name="isMarried" ngModel>    </p>      <p>      <label for="country">country </label>      <select id="country" name="country" ngModel>        <option [ngValue]="c.id" \*ngFor="let c of countryList">          {{c.name}}        </option>      </select>    </p>      <div ngModelGroup="address">        <p>        <label for="city">City</label>        <input type="text" id="city" name="city" ngModel>      </p>        <p>        <label for="street">Street</label>        <input type="text" id="street" name="street" ngModel>      </p>      <p>        <label for="pincode">Pin Code</label>        <input type="text" id="pincode" name="pincode" ngModel>      </p>      </div>      <p>      <button type="submit">Submit</button>    </p>    </form> |

Before we set the default value, it is better to create a model class for the above form. Open the app.component.ts and add the following class

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | export class contact {    firstname:string;    lastname:string;    email:string;    gender:string;    isMarried:boolean;    country:string;    address: {      city:string;      street:string;      pincode:string;    }  } |

## Set value in template-driven forms

There are two ways you can set the value of the form elements

* Two-way data binding
* Use the template reference variable

### Two-way data binding

The [two-way data binding](https://www.tektutorialshub.com/angular/angular-data-binding/#two-way-binding).is the recommended way to set the value in the template-driven forms.

The following code uses the [(ngModel)]="contact.firstname" to bind the firstname HTML element to the contact.firstname field in the component class. The advantageous here is that any changes made in the form are automatically propagated to the component class and changes made in component class are immediately shown in the form.

|  |  |
| --- | --- |
| 1  2  3  4 | <label for="firstname">First Name </label>  <input type="text" id="firstname" name="firstname" [(ngModel)]="contact.firstname"> |

#### Set the default/initial value

To set the initial or default value all you need to populate the contact model in the [ngOnInit](https://www.tektutorialshub.com/angular/angular-ngoninit-and-ngondestroy/) method as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | ngOnInit() {        this.contact = {        firstname: "Sachin",        lastname: "Tendulkar",        email: "sachin@gmail.com",        gender: "male",        isMarried: true,        country: "2",        address: { city: "Mumbai", street: "Perry Cross Rd", pincode: "400050" }      };      } |

#### Set the value individually or dynamically

|  |  |
| --- | --- |
| 1  2  3  4  5 | changeCountry() {    this.contact.country = "1";  } |

#### Reset form

|  |  |
| --- | --- |
| 1  2  3 | <button type="button" (click)="reset(contactForm)">Reset</button> |

|  |  |
| --- | --- |
| 1  2  3  4  5 | reset(contactForm :NgForm) {    contactForm.resetForm();  } |

## app.component.ts

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86 | import { Component, ViewChild, ElementRef, OnInit } from '@angular/core';  import { NgForm } from '@angular/forms';    @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent implements OnInit {    title = 'Template driven forms';        countryList: country[] = [      new country("1", "India"),      new country('2', 'USA'),      new country('3', 'England')    ];      contact: contact;      ngOnInit() {        this.contact = {        firstname: "Sachin",        lastname: "Tendulkar",        email: "sachin@gmail.com",        gender: "male",        isMarried: true,        country: "2",        address: { city: "Mumbai", street: "Perry Cross Rd", pincode: "400050" }      };      }      onSubmit() {      console.log(this.contact);    }      setDefaults() {      this.contact = {        firstname: "Sachin",        lastname: "Tendulkar",        email: "sachin@gmail.com",        gender: "male",        isMarried: true,        country: "2",        address: { city: "Mumbai", street: "Perry Cross Rd", pincode: "400050" }      };    }      changeCountry() {      this.contact.country = "1";    }      reset(contactForm :NgForm) {      contactForm.resetForm();    }    }    export class contact {    firstname: string;    lastname: string;    email: string;    gender: string;    isMarried: boolean;    country: string;    address: {      city: string;      street: string;      pincode: string;    }  }      export class country {    id: string;    name: string;      constructor(id: string, name: string) {      this.id = id;      this.name = name;    }  } |

## app.component.html

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" [(ngModel)]="contact.firstname">    </p>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" [(ngModel)]="contact.lastname">    </p>      <p>      <label for="email">Email </label>      <input type="text" id="email" name="email"  [(ngModel)]="contact.email">    </p>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" [(ngModel)]="contact.gender"> Male      <input type="radio" value="female" id="gender" name="gender" [(ngModel)]="contact.gender"> Female      </p>      <p>      <label for="isMarried">Married </label>      <input type="checkbox" id="isMarried" name="isMarried" [(ngModel)]="contact.isMarried">    </p>      <p>      <label for="country">country </label>      <select id="country" name="country" [(ngModel)]="contact.country">        <option [ngValue]="c.id" \*ngFor="let c of countryList">          {{c.name}}        </option>      </select>    </p>      <div ngModelGroup="address">        <p>        <label for="city">City</label>        <input type="text" id="city" name="city" [(ngModel)]="contact.address.city">      </p>        <p>        <label for="street">Street</label>        <input type="text" id="street" name="street" [(ngModel)]="contact.address.street">      </p>        <p>        <label for="pincode">Pin Code</label>        <input type="text" id="pincode" name="pincode"  [(ngModel)]="contact.address.pincode">      </p>      </div>      <p>      <button type="submit">Submit</button>    </p>      <p>      <button type="button" (click)="changeCountry()">Change Country</button>      <button type="button" (click)="setDefaults()">Set Defaults</button>      <button type="button" (click)="reset(contactForm)">Reset</button>    </p>      <b>valid</b> {{contactForm.valid}}    <b>touched</b> {{contactForm.touched}}    <b>pristine</b> {{contactForm.pristine}}    <b>dirty</b> {{contactForm.dirty}}    </form> |

### Template reference variable

We have a #contactForm reference variable, which is an instance of ngForm.

|  |  |
| --- | --- |
| 1  2  3 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)"> |

We can get the reference to the #contactForm in the app.component.ts, using the viewchild

|  |  |
| --- | --- |
| 1  2  3 | @ViewChild('contactForm',null) contactForm: NgForm; |

Once we have the reference, we can use the [setValue](https://www.tektutorialshub.com/angular/setvalue-patchvalue-in-angular/) method of the ngForm to set the initial value

#### Set the default or initial value

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | ngOnInit() {       this.contact = {        firstname: "Sachin",        lastname: "Tendulkar",        email: "sachin@gmail.com",        gender: "male",        isMarried: true,        country: "2",        address: {          city: "Mumbai",          street: "Perry Cross Rd",          pincode: "400050"        }      };        setTimeout(() => {        this.contactForm.setValue(this.contact);      });      } |

Note that we are using the setTimeout That is because the form controls are yet initialized when the OnInit is fired. We will get the following error message

There are no form controls registered with this group yet. If you’re using ngModel, you may want to check next tick (e.g. use setTimeout).

#### Set the value individually or dynamically

You can also set the value individually using the setValue method of the individual FormControl.

You will get the reference to the individual FormControl from the controls collection of the ngForm. Once you get the reference use the setValue on the FormControl instance to change the value.

For Example, this code will change the country to India

|  |  |
| --- | --- |
| 1  2  3  4  5 | changeCountry() {     this.contactForm.controls["country"].setValue("1");  } |

Call the changeCountry method from the Template.

|  |  |
| --- | --- |
| 1  2  3 | <button type="button" (click)="changeCountry()">Change Country</button> |

#### Reset values

You can reset the form to empty value using the reset or resetForm method of the ngForm. These also resets the form status like dirty, valid, pristine & touched, etc

|  |  |
| --- | --- |
| 1  2  3  4  5 | reset() {    this.contactForm.reset();  } |

|  |  |
| --- | --- |
| 1  2  3  4  5 | resetForm() {     this.contactForm.resetForm();  } |

#### Set Default Value

You can invoke the setValue anytime to set the form back to the default value. This will set the entire form to the value held by the contact form.

|  |  |
| --- | --- |
| 1  2  3  4  5 | setDefaults() {      this.contactForm.setValue(this.contact);    } |

#### patch value

You can make use of the [patchValue](https://www.tektutorialshub.com/angular/setvalue-patchvalue-in-angular/) to change the only few fields anytime. The control property of the ngForm returns the reference to the top level FormGroup. Then, you can make use of the patchValue method to change only firstname, lastname & email fields

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | patchValue() {      let obj = {        firstname: "Rahul",        lastname: "Dravid",        email: "rahul@gmail.com",      };        this.contactForm.control.patchValue(obj);      } |

#### Set value of nested FormGroup

You can update nested [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) by getting a reference to the nested FormGroup from the controls collection of ngForm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | changeAddress() {      let obj = {        city: "Bangalore",        street: "Brigade Road",        pincode: "600100"      };      let address= this.contactForm.controls["address"] as FormGroup      address.patchValue(obj);      } |

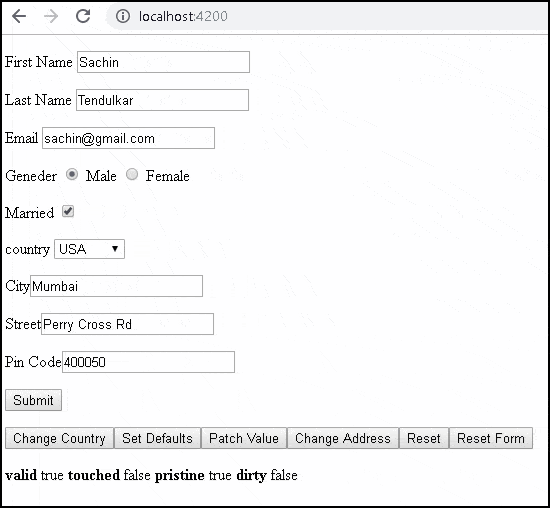
**The complete code.**

## app.component.ts

|  |
| --- |
| import { Component, ViewChild, ElementRef, OnInit } from '@angular/core';  import { NgForm, FormGroup } from '@angular/forms';    @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent implements OnInit {    title = 'Template driven forms';      @ViewChild('contactForm', null) contactForm: NgForm;      countryList: country[] = [      new country("1", "India"),      new country('2', 'USA'),      new country('3', 'England')    ];      contact: contact;      ngOnInit() {        this.contact = {        firstname: "Sachin",        lastname: "Tendulkar",        email: "sachin@gmail.com",        gender: "male",        isMarried: true,        country: "2",        address: {          city: "Mumbai",          street: "Perry Cross Rd",          pincode: "400050"        }      };        setTimeout(() => {        this.contactForm.setValue(this.contact);      });      }      onSubmit() {      console.log(this.contactForm.value);    }      setDefaults() {      this.contactForm.setValue(this.contact);    }      changeCountry() {      this.contactForm.controls["country"].setValue("1");    }      patchValue() {      let obj = {        firstname: "Rahul",        lastname: "Dravid",        email: "rahul@gmail.com",      };        this.contactForm.control.patchValue(obj);      }      changeAddress() {      let obj = {        city: "Bangalore",        street: "Brigade Road",        pincode: "600100"      };      let address= this.contactForm.controls["address"] as FormGroup      address.patchValue(obj);      }      reset() {      this.contactForm.reset();    }      resetForm() {      this.contactForm.resetForm();    }  }      export class contact {    firstname: string;    lastname: string;    email: string;    gender: string;    isMarried: boolean;    country: string;    address: {      city: string;      street: string;      pincode: string;    }  }      export class country {    id: string;    name: string;      constructor(id: string, name: string) {      this.id = id;      this.name = name;    }  } |

## app.component.html

|  |  |
| --- | --- |
| 1  67  68  69  70  71  72  73  74  75  76  77  78  79  80 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" ngModel>    </p>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" ngModel>    </p>      <p>      <label for="email">Email </label>      <input type="text" id="email" name="email"  ngModel>    </p>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" ngModel> Male      <input type="radio" value="female" id="gender" name="gender" ngModel> Female    </p>      <p>      <label for="isMarried">Married </label>      <input type="checkbox" id="isMarried" name="isMarried" ngModel>    </p>      <p>      <label for="country">country </label>        <select id="country" name="country" ngModel>        <option [ngValue]="c.id" \*ngFor="let c of countryList">          {{c.name}}        </option>      </select>      </p>      <div ngModelGroup="address">        <p>        <label for="city">City</label>        <input type="text" id="city" name="city" ngModel>      </p>        <p>        <label for="street">Street</label>        <input type="text" id="street" name="street" ngModel>      </p>      <p>        <label for="pincode">Pin Code</label>        <input type="text" id="pincode" name="pincode"  ngModel>      </p>      </div>      <p>      <button type="submit">Submit</button>    </p>      <p>      <button type="button" (click)="changeCountry()">Change Country</button>      <button type="button" (click)="setDefaults()">Set Defaults</button>      <button type="button" (click)="patchValue()">Patch Value</button>      <button type="button" (click)="changeAddress()">Change Address</button>      <button type="button" (click)="reset()">Reset</button>      <button type="button" (click)="resetForm()">Reset Form</button>    </p>      <b>valid</b> {{contactForm.valid}}    <b>touched</b> {{contactForm.touched}}    <b>pristine</b> {{contactForm.pristine}}    <b>dirty</b> {{contactForm.dirty}}    </form> |



# emplate driven form validation in Angular

[**Inject Service into Validator**](https://www.tektutorialshub.com/angular/inject-service-into-validator-in-angular/)

[**Custom Validator in Template Driven Forms**](https://www.tektutorialshub.com/angular/custom-validator-in-template-driven-forms-in-angular/)

In this tutorial, we will learn [template-driven form](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) validation in Angular. It is very common that the users will make mistakes when filling out the web form. This is where the validations come into play. The validation module must ensure that the user has provided accurate and complete information in the form fields. We must display the validation error messages to the users, disable the submit button until validation. In this tutorial, we will look at how validations are handled in [Template-driven forms](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) in Angular and learn how to use the Angular built-in validators.

Applies to: Angular 2 to the latest edition of i.e. Angular 8. Angular 9, Angular 10, Angular 11, Angular 12, Angular 13

This tutorial is a continuation of the [Angular template-driven forms tutorial](https://www.tektutorialshub.com/angular/angular-template-driven-forms/), where we built a simple form. In the next tutorial, we learned how to set the values to the form fields. We suggest you read those tutorials if you are new to Template-driven forms in Angular

**Table of Contents**

* [Template-driven Form Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#template-driven-form-validation)
* [Template](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#template)
* [Component Class](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#component-class)
* [Disabling the Browser validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#disabling-the-browser-validation)
* [Built-in Validators](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#built-in-validators)
* [Adding in Built-in Validators](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#adding-in-built-in-validators)
  + [Required Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#required-validation)
  + [Minlength Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#minlength-validation)
  + [Maxlength Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#maxlength-validation)
  + [Pattern Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#pattern-validation)
  + [Email Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#email-validation)
* [Disable Submit button](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#disable-submit-button)
* [Displaying the Validation/Error messages](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#displaying-the-validationerror-messages)
  + [Why check dirty and touched?](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#why-checknbspdirtynbspandnbsptouched)
  + [Error message](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#error-message)
* [Final Template](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#final-template)
* [Summary](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#summary)

## Template-driven Form Validation

Validations in [Template-driven forms](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) are provided by the Validation directives. The [Angular Forms Module](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/) comes with several built-in validators. You can also create your own custom Validator.

## Template

Consider the following template-driven form. It has firstname, lastname, email, gender & istoc form fields.

|  |  |
| --- | --- |
|  | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" [(ngModel)]="contact.firstname">    </p>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" [(ngModel)]="contact.lastname">    </p>      <p>      <label for="email">email </label>      <input type="text" id="email" name="email" [(ngModel)]="contact.email">    </p>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" [(ngModel)]="contact.gender"> Male      <input type="radio" value="female" id="gender" name="gender" [(ngModel)]="contact.gender"> Female    </p>      <p>      <label for="isToc">Accept TOC</label>      <input type="checkbox" id="isToc" name="isToc" [(ngModel)]="contact.isToc">    </p>      <p>      <button type="submit">Submit</button>    </p>    </form> |

**Learn more**

## ****Component Class****

|  |  |
| --- | --- |
|  | import { Component, ViewChild, ElementRef, OnInit } from '@angular/core';  import { NgForm } from '@angular/forms';    **@Component({**  **selector: 'app-root',**  **templateUrl: './app.component.html',**  **styleUrls: ['./app.component.css']**  **})**  **export class AppComponent implements OnInit  {**  **title = 'Template driven forms';**    **@ViewChild('contactForm',null) contactForm: NgForm;**    **contact:contact;**    **ngOnInit() {**    **this.contact = {**  **firstname:"",**  **lastname:"",**  **gender:"male",**  **isToc:true,**  **email:"",**  **};**    **}**    **onSubmit() {**  **console.log(this.contactForm.value);**  **}**    **}**    **export class contact {**  **firstname:string;**  **lastname:string;**  **gender:string;**  **isToc:boolean;**  **email:string;**  **}** |

## Disabling the Browser validation

First, we need to disable browser validator interfering with the Angular validator. To do that we need to add novalidate attribute on <form> element as shown below

|  |  |
| --- | --- |
|  | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)" novalidate> |

## Built-in Validators

The Built-in validators use the HTML5 validation attributes like required, minlength, maxlength & pattern. Angular interprets these validation attributes and add the validator functions to [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) instance.

## Angular Form Validations

[**Angular**](https://www.dotnettricks.com/learn/angular) Form Validations are based on HTML5 validations. Angular Form Validations validate the user input for accuracy and completeness. All validations are performed on the client side.

### Angular Forms Built-In Validation

1. required
2. minlength
3. maxlength
4. pattern
5. email – supports in Ng4
6. min
7. max

## Adding in Built-in Validators

### Required Validation

The required validator returns true only if the form control has non-empty value entered. Let us add this validator to all fields

|  |
| --- |
| **<input type="text" id="firstname" name="firstname" required [(ngModel)]="contact.firstname">** |

### Minlength Validation

This Validator requires the control value must not have less number of characters than the value specified in the validator.

For Example, minlength validator ensures that the firstname value has at least 10 characters.

|  |  |
| --- | --- |
|  | <input type="text" id="firstname" name="firstname" required minlength="10" [(ngModel)]="contact.firstname"> |

### Maxlength Validation

This Validator requires that the number of characters must not exceed the value of the attribute.

|  |  |
| --- | --- |
|  | **<input type="text" id="lastname" name="lastname" required maxlength="15" [(ngModel)]="contact.lastname">** |

### Pattern Validation

This Validator requires that the control value must match the regex pattern provided in the attribute. For example, the pattern ^[a-zA-Z]+$ ensures that the only letters are allowed (even spaces are not allowed). Let us apply this pattern to the lastName

|  |  |
| --- | --- |
|  | **<input type="text" id="lastname" name="lastname" required maxlength="15"**  **pattern="^[a-zA-Z]+$" [(ngModel)]="contact.lastname">** |

### Email Validation

This Validator requires that the control value must be a valid email address. We apply this to the email field

|  |  |
| --- | --- |
|  | **<input type="text" id="email" name="email" required email [(ngModel)]="contact.email">** |

## Disable Submit button

Now, we have successfully added the validators. You will notice that the click submit button still submits the form.

We need to disable the submit button if our form is not valid.

Angular forms module keep track of the state of our form and each of its form elements. These states are exposed to the user through FormGroup, FormArray & FormControl objects.

We get the reference to the top-level FormGroup instance by creating a template variable and bind it to ngForm. We have already done it when we had added the #contactForm="ngForm" in our form tag.

The [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) has a valid property, which is set to true if all of its child controls are valid. We use it to set the disabled attribute of the submit button.

|  |  |
| --- | --- |
|  | <button type="submit" [disabled]="!contactForm.valid">Submit</button> |

So long as contactForm.valid remains false, the submit button remains disabled.

## Displaying the Validation/Error messages

We need to provide a short and meaningful error message to the user.

Angular creates a FormControl for each and every field, which has ngModel directive applied. The FormControl exposes the state of form element like valid, dirty, touched, etc

.

There are two ways in which you can get the reference to the FormControl.

One way is to use the contactForm variable. We can use the **contactForm.controls.firstname.valid to find out if the firstname is valid.**

The other way is to create a new local variable for each FormControl For Example, the following firstname="ngModel" creates the firstname variable with the FormControl instance.

|  |  |
| --- | --- |
| 1  2  3  4 | <input type="text" id="firstname" name="firstname" required minlength="10"              #firstname="ngModel" [(ngModel)]="contact.firstname"> |

Now, we have a reference to the firstname FormControl instance, we can check its status. We use the valid property to check if the firstname has any errors.

valid: returns either invalid status or null which means a valid status

|  |  |
| --- | --- |
| 5 | <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)">     Invalid First Name  </div> |

### Why check dirty and touched?

We do not want the application to display the error when the form is displayed for the first time. We want to display errors only after the user has attempted to change the value.

**The dirty & touched properties help us do that.**

**dirty: A control is dirty if the user has changed the value in the UI.  
touched: A control is touched if the user has triggered a blur event on it.**

### Error message

The error message ” “Invalid First Name” ” is not helpful. The firstname has two validators. required and minlength

Any errors generated by the failing validation is updated in the errors object. The errors object returns the error object or null if there are no errors.

|  |  |
| --- | --- |
|  | <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)">    Invalid First Name    <div \*ngIf="firstname.errors?.[‘required’]">       First Name is required    </div>    <div \*ngIf="firstname.errors.minlength">      First Name Minimum Length is {{firstname.errors.minlength?.requiredLength}}    </div>  </div> |

Note that the minlength validators return the {{firstname.errors.minlength?.requiredLength}}, which we use the display the error message.

## Final Template

|  |  |
| --- | --- |
|  | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)" novalidate>     <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" required minlength="10" #firstname="ngModel"        [(ngModel)]="contact.firstname">    </p>    <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)" class="error">      <div \*ngIf="firstname.errors.required">        First Name is required      </div>      <div \*ngIf="firstname.errors.minlength">        First Name Minimum Length is {{firstname.errors.minlength?.requiredLength}}      </div>    </div>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" required maxlength="15" #lastname="ngModel"              pattern="^[a-zA-Z]+$"  [(ngModel)]="contact.lastname">    </p>    <div \*ngIf="!lastname?.valid && (lastname?.dirty || lastname?.touched)" class="error">      <div \*ngIf="lastname.errors.required">        Last Name is required      </div>      <div \*ngIf="lastname.errors.maxlength">        Last Name Minimum Length is {{lastname.errors.maxlength?.requiredLength}}      </div>      <div \*ngIf="lastname.errors.pattern">        Only characters are allowed      </div>    </div>          <p>      <label for="email">email </label>      <input type="text" id="email" name="email" required email #email="ngModel" [(ngModel)]="contact.email">    </p>    <div \*ngIf="!email?.valid && (email?.dirty || email?.touched)" class="error">      <div \*ngIf="email.errors.required">        Email is required      </div>      <div \*ngIf="email.errors.email">        Invalid Email Address      </div>    </div>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" #gender="ngModel" required [(ngModel)]="contact.gender">      Male      <input type="radio" value="female" id="gender" name="gender" #gender="ngModel" required        [(ngModel)]="contact.gender"> Female    </p>    <div \*ngIf="!gender?.valid && (gender?.dirty || gender?.touched)" class="error">      <div \*ngIf="gender.errors.required">        Gender is required      </div>    </div>        <p>      <label for="isToc">Accept TOC</label>      <input type="checkbox" id="isToc" name="isToc" required #isToc="ngModel" [(ngModel)]="contact.isToc">    </p>    <div \*ngIf="!isToc?.valid && (isToc?.dirty || isToc?.touched)" class="error">      <div \*ngIf="isToc.errors.required">        Please accept the TOC      </div>    </div>      <p>      <button type="submit" [disabled]="!contactForm.valid">Submit</button>    </p>      <p>{{contactForm.valid}} </p>    </form>  Class example  import { NgModule } from '@angular/core';  import { FormsModule } from '@angular/forms';  import { BrowserModule } from '@angular/platform-browser';  import { AppRoutingModule } from './app-routing.module';  import { AppComponent } from './app.component';  @NgModule({    declarations: [      AppComponent    ],    imports: [      BrowserModule,      AppRoutingModule,FormsModule    ],    providers: [],    bootstrap: [AppComponent]  })  export class AppModule { }  <!doctype html>  <html lang="en">  <head>      <title>Title</title>      <!-- Required meta tags -->      <meta charset="utf-8">      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">      <!-- Bootstrap CSS -->      <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">  </head>  <body>      <!-- Optional JavaScript -->      <!-- jQuery first, then Popper.js, then Bootstrap JS -->      <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>      <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>      <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>      <br/>      <br/>      <div class="container">          <div class="row">              <div class="form-bg">                  <form #studentForm="ngForm" (ngSubmit)="RegisterStudent(studentForm)">                      <div class="panel panel-primary">                          <div class="panel-heading">                              <h3 class="panel-title">Student Registration</h3>                          </div>                          <div class="panel-body">                              <div class="form-group">                                  <label for="firstName">First Name</label>                                  <input id="firstName" type="text" class="form-control" name="firstName" #firstname="ngModel" required [(ngModel)]="studentForm.value.firstName">                                  <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)">                                      Invalid First Name                                  </div>                                  <div \*ngIf="firstname.errors?.['required']">                                      First Name is required                                  </div>                              </div>                              <div class="form-group ">                                  <label for="lastName ">Last Name</label>                                  <input id="lastName " type="text " class="form-control " name="lastName " #lastName="ngModel" required [(ngModel)]="studentForm.value.lastName ">                                  <div \*ngIf=" !lastName?.valid && (lastName?.dirty || lastName?.touched) ">                                      Invalid First Name                                  </div>                                  <div \*ngIf="lastName.errors?.[ 'required'] ">                                      lastName is required                                  </div>                              </div>                              <div class="form-group ">                                  <label for="email ">Email</label>                                  <input id="email " type="text " class="form-control " name="email" #email="ngModel" [(ngModel)]="studentForm.value.email" pattern="^[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$" required>                                  <div \*ngIf="!email?.valid && (email?.dirty || email?.touched)" class="error">                                      <div \*ngIf="email.errors?.['required']">                                          Email is required                                      </div>                                      <div \*ngIf="email.errors?.['email']">                                          Invalid Email Address                                      </div>                                        <div \*ngIf="email.errors?.['pattern']">                                          invalid email                                      </div>                                  </div>                              </div>                              <div class=" form-group ">                                  <label for="Gender ">Gender</label><br>                                  <input type="radio" name="gender " value="male " checked [(ngModel)]="gender ">Male                                  <input type="radio" name="gender " value="female " [(ngModel)]="gender ">Female </div>                              <div class="form-group">                                  <div class="form-control">                                      <input type="checkbox " name="isaccepted " ngModel>AcceptTerms and Condition</div>                              </div>                              <div class="form-group ">                                  <div class="form-control ">                                      <select name="branch " id="branch " ngModel class="form-control ">                                          <!-- <option value="1 ">ComputerScience</option>                                          <option value="2 ">Mechanical</option>                                          <option value="3 ">ai and ml</option>                                          <option value="4 ">Electrical and Electronics</option>                                          <option value="5 ">Electronics and Communication</option> -->                                       <option \*ngFor="let branch of Branches " [value]="branch.name ">                                          {{branch.name}}                                        </option>                                      </select>                                  </div>                              </div>                          </div>                          <div class=" panel-footer ">                              <button class="btn btn-primary " type="submit ">Submit</button>                          </div>                      </div>                  </form>              </div>          </div>      </div>  </body>  </html> |

## Summary

Angular template-driven form validation uses the directives known as validators. The validators handle form validations and display validation messages. The Angular comes up with several built-in validators for this purpose. They are minlength