# gx-dynamic-form

## Abbreviations

* GXFom: gx-dynamic-form Angular component. This Angular component is put in the HTML template to draw the HTML.
* GXLib: This entire library project.
* GXTemplate: The user created JSON template structured as JSON array which is responsible to draw the HTML form at run time. The objects defined in the GXJson array are drawn in the final HTML form.
* GXObject: A JSON object which is member of above GXTemplate (Json array).
* GXControl: Final HTML control created from GXObject. GXControl is a resultant Angular component created out of GXObject.
* GXContainer: Each GXControl is contained in a single container which is mostly a div HTML tag. We refer to this container div as GXContainer
* GXMain, GXHelper: GXControl is a set of controls contained in GXContainer. There is generally one main control in GXControl which we call as GXMain. For example GXControl created from an input type GXObject will have several controls such as <input><span>, <label>, <button>, <abbr>, <small> etc. The <input> control is main control and other controls are helper controls. This input control is GXMain and other controls are GXHelper of the GXControl.

The purpose of GXLib is to use user provided GXTemplate (each member of GXTemplate is GXObject) and convert it to GXForm. Various controls in GXForm are GXControl HTML. GXObject is finally converted to GXControl in GXForm.

* GXControl signature: This is bare minimum GXControl layout structure just showing the hierarchy of nested controls.

## Introduction

GXLib is a library to create forms from json templates (GXTemplate). The templates can be stored in database which are available at run time. GXLib is meant to be used with Angular 2 /4 / 5 / 6 / 7+. GXLib creates on the fly form with controls dynamically created from GXTemplate. The GXObject of GXTemplate regulate the look and feel and behavior of controls in the resulting HTML form (GXForm).

Apart from providing default click events and validations the GXLib also enables the developer to write custom validations and custom click events by using the services of GXLib. The library provides its own set of controls, but developer is free to create custom controls and then register those controls with GXLib; the custom controls such created then behave like build-in controls and can be used like any other provided built-in control. The custom user created controls such created can be used in GXTemplate as GXObject.

Purpose of library is to quickly create forms based on user provided GXTemplate. The styles and layouts of controls are highly customizable. The library is a boon for creating simple forms for feedback, login, user profile and many more. You can store several JSON templates in database, provide them at run time through HTTP requests, create forms based on them and save data at server on click of submit buttons.

## Features at a glance

## Dependencies

Angular material

Bootstrap 4

## Quick start guide

## Error handling

## Control reference guide

**input**

GXObject properties

Sample GXObject

GXControl signature

textarea

button

group

array

buttongroup

anchor

checkbox

radio

select

checkboxgroup

mat-checkbox

mat-radio

mat-select

mat-textarea

mat-input

mat-datepicker

mat-button

mat-checkboxgroup

mat-autocomplete

name-value-input

## Id and name attribute of GXControl

For GXObject the **id** attribute is optional. If you do not provide an id for a GXObject a new id is automatically created at runtime in the format xid1, xid2, xid3…. Similarly, *name* attribute is also optional. If you do not provide a name attribute, then its *id* will be treated as *name*. Recall that when form is submitted then the name of a control becomes property of output json of the form. So if you do not provide id or name for a GXObject, then output json against form submit will be something like

{

“xid1”: “some value”

“xid2”: “some value 1”

“xid3”: “some value 2”

}

If above JSON makes little sense to you then please provide the values for at least name *attribute*.

The id property of GXObject is promoted as CSS class attribute in resulting GXControl for styling purposes. Each GXControl in GXForm is contained in a div or some other container tag. A css class with name as id is assigned to this container. So, it is recommended to provide an id for a GXControl if you want to style it in custom manner.

## Validations

### Custom validations

## Style guide

To effectively use proper styling with GXLib it is recommended to brush up the **specificity** concept in css. A ready refresh knowledge link is [here](https://www.w3schools.com/css/css_specificity.asp). GXlib does not have any hard-coded styles within it. GXLib provides a file gx-form.basic.scss wherein many classes are provided with ::ng-deep scss operator. You are suggested to include this file in app.component.ts as shown below. This enables all the classes provided within the file to reach at each level of resultant DOM tree. You are free to add / modify classes in the gx-form.basic.scss file. How to include the .scss file in app.component.ts:

@**Component**({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.scss', './gx-form.basic.scss']

})

You can also create custom scss templates which can be used with GXForm. Those custom templates should be added to the styleUrl’s array. After including .scss file in styleUrls you must add the relevant classes in the meta.client tag of the GXTemplate as follows:

{

type: 'meta'

, client: {

title: 'Test form'

, class: 'basic-style-1 form-custom'

…

You can append many classes in the ‘class’ property of above meta.client.class property. All those classes will be available to GXForm and therefore to the entire DOM tree. This is the beauty of :: ng-deep modifier.

You can write classes in a .scss file, include the .scss file in the app.component.ts and then use the classes at component level by using ‘class’ attribute of the JSON object as follows. You can also make use of style attribute which is an object. The style attribute will be promoted as ngStyle. This is Angular directive which after compilation will add up raw inline styles to the component with highest specificity. Please note that style object provided in GXObject will be added to style attribute of GXMain of GXControl and not to GXContainer or GXHelper.

{

type: 'input'

, subtype: 'text'

, id: 'id2'

, name: 'myName1'

, label: 'First name1'

, placeholder: 'First name1'

, value: ''

, class: 'label-name'

, style: { width: '40%' }

, remove: true

, validation: {

required: { message: '$ is required' }

}

}

Id of each GXControl is also promoted to the class attribute of the corresponding GXControl. But the class attribute is applied to the GXContainer and not to GXMain or GXHelper controls. Hence if say a GXObject has an id **myObjectId,** and you have created a class with name myObjectId and included that class in the .scss file with ::ng-deep operator, then the class myObjectId will be applicable to the corresponding GXContainer which may be a div.

### 

### How to apply styles from GXObject

First of all you need to create a class and make that class available to GXForm. Steps to make a class available to GXForm:

* Create a .scss file and write all your classes in that .scss file. Name the .scss file for example myscss.scss. Your all top level css classes should have ::ng-deep decorator otherwise those classes will not be available to GXControl at compile time.

Example:

::ng-deep .basic-style-1 {

margin: 0px auto;

max-width: 600px;

font: 13px "Lucida Sans Unicode", "Lucida Grande", sans-serif;

border: 2px solid green;

small {

color: red;

}

legend {

font-weight: bold;

margin: 5px;

}

div {

margin: 5px 8px 0 8px;

}

i {

margin-left: 5px;

}

label {

margin-right: 5px;

}

fieldset {

border: 1px groove threedface; *// padding: 5px;*

margin: 5px;

}

}

::ng-deep .custom1 {…}

::ng-deep .custom2 {…}

::ng-deep .custom3 {…}

* In GXTemplate in the meta.client.class property append all individual classes separated by a space.

Example:

const form1 = [

{

type: 'meta'

, client: {

title: 'Test form'

, class: 'basic-style-1 custom1 custom2 custom3'

}

, id: 'gx-form1'

}

,

{

type: 'input'

, subtype: 'text'

, id: 'id1'

, name: 'myName'

, label: 'First name2'

, placeholder: 'First name2'

, value: ''

*// , style: { 'margin-top': '8px', 'width': '100%' }*

*// , remove: true*

, validation: {

required: { message: '$ is required' }

}

}

]

Once you have made a class available to GXForm now that class can be used in GXTemplate in following ways: (There is a fourth way also to directly include the style in GXMain which is described at the end).

#### Id promoted as class in GXContainer

The id of GXObject is promoted as class attribute of GXContainer of GXControl. So if you have provided a class with the same name as id to GXForm as shown above then this class will be automatically applied to GXContainer. In this case you need not have to provide any other specific entry for that class in GXTemplate. This is easiest and recommended way of using a css class. Since the class is applied to GXContainer, you can write the class in a manner to control the behavior of GXHelper and GXMain controls.

Example:

1. In GXTemplate I defined an input GXObject, id is myName1:

{

type: 'input'

, subtype: 'text'

, id: 'myName1'

, label: 'First name'

, placeholder: 'First name'

, value: ''

, validation: {

required: { message: '$ is required' }

}

}

1. I made the class myName1 available in meta.client.class property ofGXTemplate:

{

type: 'meta'

, client: {

title: 'Test form'

, class: 'basic-style-1 myName1'

, validationtext: ' indicates required field'

}

, id: 'gx-form1'

}

1. The format of input GXControl in GXLib is following. This trimmed down control for explanation purpose. See how myClass1 is applied to GXContainer which is a <div>

<div *class*="myName1">

<label>

<abbr>\*</abbr>

</label>

<input>

<button>

<i></i>

</button>

<app-gx-error></app-gx-error>

</div>

#### class property of GXObject as string

You can control the styles in more granular manner in this way. GXObject can have a property named as ‘class’. Value for this property can be string or an object. If value is a string say myClass1 then GXForm will look for myClass1 class and append it to classes of GXContainer.

Example:

1. I assume that a class myClass1 is already made available to GXForm.
2. In GXObject I defined an input type with class property as string

{

type: 'input'

, subtype: 'text'

, id: 'id1'

, name: 'myName'

, label: 'First name'

, placeholder: 'First name'

, value: ''

, class: 'myClass1'

}

1. Now the class myClass1 will be used by input GXControl signature as below

<div ngC*lass*="myClass1">

<label>

<abbr>\*</abbr>

</label>

<input>

<button>

<i></i>

</button>

<app-gx-error></app-gx-error>

</div>

After compilation of above HTML signature snippet myClass1 will be appended to already existing other classes of <div> container. In the class myClass1 you can write styles to control the behavior of all GXHelper and GXMain controls. In above <label>, <abbr>, <button>, <i> are GXHelper controls and <input> is the GXMain control.

#### class property of GXObject as object

When you provide class property of GXObject as an object you have got more control over GXHelper and GXMain controls. In fact you can define sub classes for each of GXContainer, GXMain and GXHelper controls. You can inspect the signature of GXControl and provide a class for every HTML control may it be helper or main control.

Example:

1. I assume that class myClass1 is already made available to GXForm.
2. I inspect the signature of input GXObject which is as follows:

<div>

<label>

<abbr>\*</abbr>

</label>

<input>

<button>

<i></i>

</button>

<app-gx-error></app-gx-error>

</div>

You can see that <div> is GXContainer, <label>, <abbr>, <button>, <i> are GXHelper and <input> is GXMain.

1. I set the GXObject class as object as follows:

{

type: 'input'

, subtype: 'text'

, id: 'id1'

, name: 'myName'

, label: 'First name2'

, placeholder: 'First name2'

, value: ''

, class: {

, div: ‘myDiv1’

, label:’myLabel1’

, input: ‘myInput1’

, button: ‘myButton1

, i: ‘myIcon1

, abbr: ‘myAbbr1’

}

}

Now if the classes myLabel1, myInput1, myButton1, myIcon1 and myAbbr1 are made available to GXControl then following GXControl signature will be created before compilation:

<div class=’myDiv1’>

<label class=’myLabel1’>

<abbr class=’myAbbr1’>\*</abbr>

</label>

<input class = ‘myInput1’>

<button class = ‘myButton1’>

<i class = ‘myIcon1’></i>

</button>

<app-gx-error></app-gx-error>

</div>

#### Directly applying styles

This is the fourth way when you can apply styles from GXObject directly to individual controls of GXControl signature. This has got nothing to do with css class.

Provide style property in GXObject. This style property can be a direct style object or a nested object of styles for each individual member component of GXControl. The object style is promoted to GXMain control of the GXControl signature. The nested object style is promoted to each individual GXHelper, GXContainer and GXMain control in the same way as in class above.

Example 1: style is a simple object consisting of style values

{

type: 'input'

, subtype: 'text'

, id: 'id1'

, name: 'myName'

, label: 'First name2'

, placeholder: 'First name2'

, value: ''

*, style: { 'margin-top': '8px', 'width': '100%' }*

}

The above will be promoted to following. Remember that GXMain is <input> control and single object style is promoted to GXMain.

<div class=’myDiv1’>

<label class=’myLabel1’>

<abbr class=’myAbbr1’>\*</abbr>

</label>

<input ngStyle = “margin-top: 8px, width:100%”>

<button class = ‘myButton1’>

<i class = ‘myIcon1’></i>

</button>

<app-gx-error></app-gx-error>

</div>

Example 2: style is a nested object

{

type: 'input'

, subtype: 'text'

, id: 'id1'

, name: 'myName'

, label: 'First name2'

, placeholder: 'First name2'

, value: ''

*, style: {*

*‘div’: {'margin-top': '8px', 'width': '100%' }*

*, ‘label’: {‘color’: ‘red’}*

*, ‘abbr’: {‘color’: ‘red’}*

*, ‘input’: {‘padding’: ‘5px;’}*

*, ‘button’: {‘margin’: ‘5px’}*

*}*

}

The above style of GXObject will be promoted to following GXControl signature:

<div ngStyle = “*margin-top: 8px, width: 100%*”>

<label ngStyle = “color:red”>

<abbr ngStyle = “color:red”>\*</abbr>

</label>

<input ngStyle = “padding: 5px”>

<button ngStyle = “margin: 5px;”>

<i class = ‘myIcon1’></i>

</button>

<app-gx-error></app-gx-error>

</div>

## Questions and Answers

## Custom controls

## Technical aspects

## How to