gModel.js?message=docs(ngModel.NgModelController)%3A%20describe%20your%20change...#L22)

/iew Source (https://github.com/angular/angular.js/tree/master/src/ng/directive/ngModel.js#L22)

ngModel.NgModelController

- type in module ng (api/ng)

`NgModelController` provides API for the `ngModel` (api/ng/directive/ngModel) directive. The controller contains services for data-binding, validation, CSS updates, and value formatting and parsing. It purposefully does not contain any logic which deals with DOM rendering or listening to DOM events. Such DOM related logic should be provided by other directives which make use of **`NgModelController`** for data-binding to control elements. Angular provides this DOM logic for most `input` (api/ng/directive/input) elements. At the end of this page you can find a custom control example (api/ng/type/ngModel.NgModelController#custom-control-example) that uses `ngModelController` to bind to `contenteditable` elements.

Methods

`$render();`

Called when the view needs to be updated. It is expected that the user of the ng-model directive will implement this method.

The `$render()` method is invoked in the following situations:

- `$rollbackViewValue()` is called. If we are rolling back the view value to the last committed value then `$render()` is called to update the input control.
- The value referenced by `ng-model` is changed programmatically and both the `$modelValue` and the `$viewValue` are different from last time.

Since `ng-model` does not do a deep watch, `$render()` is only invoked if the values of `$modelValue` and `$viewValue` are actually different from their previous value. If `$modelValue` or `$viewValue` are objects (rather than a string or number) then `$render()` will not be invoked if you only change a property on the objects.

`$isEmpty(value);`

This is called when we need to determine if the value of an input is empty.

For instance, the required directive does this to work out if the input has data or not.

The default `$isEmpty` function checks whether the value is **`undefined`** , `''` , **`null`** or **`NaN`** .

You can override this for input directives whose concept of being empty is different from the default. The `checkboxInputType` directive does this because in its case a value of **`false`** implies empty.

Parameters

Param	Type	Details
value	* ()	The value of the input to check for emptiness.

Returns

boolean ()	True if `value` is "empty".

`$setValidity(validationErrorKey, isValid);`

Change the validity state, and notify the form.

This method can be called within $parsers/$formatters or a custom validation implementation. However, in most cases it should be sufficient to use the `ngModel.$validators` and `ngModel.$asyncValidators` collections which will call `$setValidity` automatically.

Parameters

Param	Type	Details
validationErrorKey	string ()	Name of the validator. The `validationErrorKey` will be assigned to either `$error[validationErrorKey]` or `$pending[validationErrorKey]` (for unfulfilled `$asyncValidators`), so that it is available for data-binding. The `validationErrorKey` should be in camelCase and will get

		converted into dash-case for class name. Example: `myError` will result in `ng-valid-my-error` and `ng-invalid-my-error` class and can be bound to as `{{someForm.someControl.$error.myError}}` .
isValid	boolean ()	Whether the current state is valid (true), invalid (false), pending (undefined), or skipped (null). Pending is used for unfulfilled `$asyncValidators` . Skipped is used by Angular when validators do not run because of parse errors and when `$asyncValidators` do not run because any of the `$validators` failed.

`$setPristine();`

Sets the control to its pristine state.

This method can be called to remove the `ng-dirty` class and set the control to its pristine state (`ng-pristine` class). A model is considered to be pristine when the control has not been changed from when first compiled.

`$setDirty();`

Sets the control to its dirty state.

This method can be called to remove the `ng-pristine` class and set the control to its dirty state (`ng-dirty` class). A model is considered to be dirty when the control has been changed from when first compiled.

`$setUntouched();`

Sets the control to its untouched state.

This method can be called to remove the `ng-touched` class and set the control to its untouched state (`ng-untouched` class). Upon compilation, a model is set as untouched by default, however this function can be used to restore that state if the model has already been touched by the user.

`$setTouched();`

Sets the control to its touched state.

This method can be called to remove the `ng-untouched` class and set the control to its touched state (`ng-touched` class). A model is considered to be touched when the user has first focused the control element and then shifted focus away from the control (blur event).

`$rollbackViewValue();`

Cancel an update and reset the input element's value to prevent an update to the `$modelValue`, which may be caused by a pending debounced event or because the input is waiting for a some future event.

If you have an input that uses `ng-model-options` to set up debounced events or events such as blur you can have a situation where there is a period when the `$viewValue` is out of synch with the ngModel's `$modelValue`.

In this case, you can run into difficulties if you try to update the ngModel's `$modelValue` programmatically before these debounced/future events have resolved/occurred, because Angular's dirty checking mechanism is not able to tell whether the model has actually changed or not.

The `$rollbackViewValue()` method should be called before programmatically changing the model of an input which may have such events pending. This is important in order to make sure that the input field will be updated with the new model value and any pending operations are cancelled.

app.js () index.html () Edit in Plunker

```
angular.module('cancel-update-example', [])

.controller('CancelUpdateController', ['$scope', function($scope) {
  $scope.resetWithCancel = function(e) {
    if (e.keyCode == 27) {
      $scope.myForm.myInput1.$rollbackViewValue();
      $scope.myValue = '';
    }
  };
  $scope.resetWithoutCancel = function(e) {
    if (e.keyCode == 27) {
      $scope.myValue = '';
    }
  };
}]);
```

Try typing something in each input. See that the model only updates when you blur off the input.

Now see what happens if you start typing then press the Escape key

With $rollbackViewValue()

myValue: ""

Without $rollbackViewValue()

myValue: ""

`$validate();`

Runs each of the registered validators (first synchronous validators and then asynchronous validators). If the validity changes to invalid, the model will be set to **`undefined`** , unless `ngModelOptions.allowInvalid` (api/ng/directive/ngModelOptions) is **`true`** . If the validity changes to valid, it will set the model to the last available valid modelValue, i.e. either the last parsed value or the last value set from the scope.

`$commitViewValue();`

Commit a pending update to the `$modelValue` .

Updates may be pending by a debounced event or because the input is waiting for a some future event defined in `ng-model-options` . this method is rarely needed as **`NgModelController`** usually handles calling this in response to input events.

`$setViewValue(value, trigger);`

Update the view value.

This method should be called when an input directive want to change the view value; typically, this is done from within a DOM event handler.

For example input (api/ng/directive/input) calls it when the value of the input changes and select (api/ng/directive/select) calls it when an option is selected.

If the new `value` is an object (rather than a string or a number), we should make a copy of the object before passing it to `$setViewValue` . This is because `ngModel` does not perform a deep watch of objects, it only looks for a change of identity. If you only change the property of the object then ngModel will not realise that the object has changed and will not invoke the `$parsers` and `$validators` pipelines.

For this reason, you should not change properties of the copy once it has been passed to `$setViewValue` . Otherwise you may cause the model value on the scope to change incorrectly.

When this method is called, the new `value` will be staged for committing through the `$parsers` and `$validators` pipelines. If there are no special `ngModelOptions` (api/ng/directive/ngModelOptions) specified then the staged value sent directly for processing, finally to be applied to `$modelValue` and then the **expression** specified in the `ng-model` attribute.

Lastly, all the registered change listeners, in the `$viewChangeListeners` list, are called.

In case the ngModelOptions (api/ng/directive/ngModelOptions) directive is used with `updateOn` and the **`default`** trigger is not listed, all those actions will remain pending until one of the `updateOn` events is triggered on the DOM element. All these actions will be debounced if the ngModelOptions (api/ng/directive/ngModelOptions) directive is used with a custom debounce for this particular event.

Note that calling this function does not trigger a `$digest` .

Parameters

Param	Type	Details
value	string ()	Value from the view.
trigger	string ()	Event that triggered the update.

Properties

$viewValue

string () Actual string value in the view.

$modelValue

* () The value in the model that the control is bound to.

$parsers

Array.<Function> ()

Array of functions to execute, as a pipeline, whenever the control reads value from the DOM. The functions are called in array order, each passing its return value through to the next. The last return value is forwarded to the `$validators` (api/ng/type/ngModel.NgModelController#$validators) collection.

Parsers are used to sanitize / convert the `$viewValue` (api/ng/type/ngModel.NgModelController#$viewValue).

Returning **`undefined`** from a parser means a parse error occurred. In that case, no `$validators` (api/ng/type/ngModel.NgModelController#$validators) will run and the `ngModel` will be set to **`undefined`** unless `ngModelOptions.allowInvalid` (api/ng/directive/ngModelOptions) is set to **`true`** . The parse error is stored in `ngModel.$error.parse` .

$formatters

Array.<Function> ()

Array of functions to execute, as a pipeline, whenever the model value changes. The functions are called in reverse array order, each passing the value through to the next. The last return value is used as the actual DOM value. Used to format / convert values for display in the control.

```
function formatter(value) {
  if (value) {
    return value.toUpperCase();
  }
}
ngModel.$formatters.push(formatter);
```

$validators

Object.<string, function> ()

A collection of validators that are applied whenever the model value changes. The key value within the object refers to the name of the validator while the function refers to the validation operation. The validation operation is provided with the model value as an argument and must return a true or false value depending on the response of that validation.

```
ngModel.$validators.validCharacters = function(modelValue,
viewValue) {
  var value = modelValue || viewValue;
  return /[0-9]+/.test(value) &&
         /[a-z]+/.test(value) &&
         /[A-Z]+/.test(value) &&
         /\W+/.test(value);
};
```

$asyncValidators

Object.<string, function> ()

A collection of validations that are expected to perform an asynchronous validation (e.g. a HTTP request). The validation function that is provided is expected to return a promise when it is run during the model validation process. Once the promise is delivered then the validation status will be set to true when fulfilled and false when rejected. When the asynchronous validators are triggered, each of the validators will run in parallel and the model value will only be updated once all validators have been fulfilled. As long as an asynchronous validator is unfulfilled, its key will be added to the controllers `$pending` property. Also, all asynchronous validators will only run once all synchronous validators have passed.

Please note that if $http is used then it is important that the server returns a success HTTP response code in order to fulfill the validation and a status level of `4xx` in order to reject the validation.

```
ngModel.$asyncValidators.uniqueUsername = function(modelValue,
viewValue) {
  var value = modelValue || viewValue;

  // Lookup user by username
  return $http.get('/api/users/' + value).
     then(function resolved() {
       //username exists, this means validation fails
       return $q.reject('exists');
     }, function rejected() {
       //username does not exist, therefore this validation
passes
       return true;
     });
};
```

$viewChangeListeners

Array.<Function> ()

Array of functions to execute whenever the view value has changed. It is called with no arguments, and its return value is ignored. This can be used in place of additional $watches against the model value.

$error

Object ()

An object hash with all failing validator ids as keys.

`$pending`

Object ()	An object hash with all pending validator ids as keys.

`$untouched`

boolean ()	True if control has not lost focus yet.

`$touched`

boolean ()	True if control has lost focus.

`$pristine`

boolean ()	True if user has not interacted with the control yet.

`$dirty`

boolean ()	True if user has already interacted with the control.

`$valid`

boolean ()	True if there is no error.

`$invalid`

boolean ()	True if at least one error on the control.

`$name`

string ()	The name attribute of the control.

Example

Custom Control Example

This example shows how to use **`NgModelController`** with a custom control to achieve data-binding. Notice how different directives (`contenteditable` , `ng-model` , and `required`) collaborate together to achieve the desired result.

`contenteditable` is an HTML5 attribute, which tells the browser to let the element contents be edited in place by the user.

We are using the $sce (api/ng/service/$sce) service here and include the $sanitize (api/ngSanitize) module to automatically remove "bad" content like inline event listener (e.g. `<span onclick="...">`). However, as we are using `$sce` the model can still decide to provide unsafe content if it marks that content using the `$sce` service.

style.css () script.js () index.html () protractor.js () Edit in Plunker

```
[contenteditable] {
  border: 1px solid black;
  background-color: white;
  min-height: 20px;
}

.ng-invalid {
  border: 1px solid red;
}
```

Change me!

Change me!

Super-powered by Google ©2010-2015 (v1.4.0-beta.5 karmic-stabilization (https://github.com/angular/angular.js/blob/master/CHANGELOG.md#1.4.0-beta.5)) Back to top