alight Documentation

Release 0.14.0

Oleg Nechaev

Apr 16, 2017

Contents

Main functional, read Alight in 7 steps

Contents:

Contents 2:

CHAPTER 1

API

alight.debug

Turn on a logging debug information

- alight.debug.scan - logging scan operations
- alight.debug.directive - logging binding directives
- alight.debug.parser - logging parsing expressions
- alight.debug.watch - logging function $watch
- alight.debug.watchText - logging function $watchText

alight.bootstrap(option)

It lets you a few ways to bind DOM, read more

alight.bind(scope, element, option)

Bind a change detector to the DOM element, alias **alight.applyBindings**

- scope - an instance of Scope (or ChangeDetector)
- element - DOM element
- option.skip_top = false - Skip binding the top DOM element
- option.skip_attr = ['al-repeat', 'al-app'] - Skip attributes for a binding, for the top element
- option.attachDirective = {} - Attach a custom directive

```
var scope = alight.Scope();
var element = document.body;
alight.bind(scope, element);
```

alight.Scope()

Create a new scope

alight.ChangeDetector([scope])

Create a new change detector, read more

alight.ctrl

Dictionary of controllers, alias for **alight.controllers**

alight.filters

Dictionary of filters

alight.directives

Dictionary of directives, short alias **alight.d**

alight.text

Collection of text directives

alight.hook

Different hooks

alight.nextTick(callback)

Execute the function on next tick

CHAPTER 2

Events

A way to bind events.

Listing 2.1: Syntax

```
<div al-on.eventname.modifier="expression"> </div>
<div @eventname.modifier="expression"> </div>
```

Modifiers for keydown, keypress, keyup

- Filter by extra key: alt, ctrl (control), meta, shift
- Filter by keycode: enter, tab, delete, backspace, esc, space, up, down, left, right, <any number of key code>

Special modifiers for all events

- stop - calls stopPropagation
- prevent - calls preventDefault
- nostop - voids stopPropagation for click, dblclick, submit
- noprevent - voids preventDefault for click, dblclick, submit
- noscan - voids $scan
- throttle-<number>
- debounce-<number>

Modificators “prevent” and “stop” are on by default for click, dblclick and submit. Available arguments in expressions: **$event, $element, $value**

Examples

- al-on.keyup="onKeyup($event)"
- al-on.keydown.tab="onTab()"
- @keyup="onKeyup($event)"
- @keyup="key=$event.keyCode"
- @keydown.tab="onTab()"
- @keydown.enter="onEnter($event)"
- @keydown.13="onEnter($event)"
- @keydown.13.prevent="onEnter($event)"
- @keydown.control.enter="onEnter($event)"
- @keydown.control.shift.enter="onEnter($event)"
- @input="value=$event.target.value"
- @submit="onSubmit()"
- @submit.noprevent="onSubmit()"
- @click="onClick($event)"
- @click.noprevent="onClick()"
- @click.noprevent.stop="onClick()"
- @mousemove.noscan="onMousemove($event)"
- @mousemove.throttle-300="onMousemove($event)"
- @mousemove.debounce-300="onMousemove($event)"

You can make aliases and filters for events

It lets you:

- bind a few events to one expression
- makes custom modifiers (filters)
- make aliases for events

Listing 2.2: A few ways to make aliases

```
alight.hooks.eventModifier['enter'] = 'keydown blur';
alight.hooks.eventModifier['enter'] = ['keydown', 'blur'];
alight.hooks.eventModifier['enter'] = (event, env) => {}
alight.hooks.eventModifier['enter'] = {
  event: 'keydown blur',        // can be omitted
  fn: (event, env) => {}        // can be omitted
  init: (scope, element) => {}  // can be omitted
}
alight.hooks.eventModifier.enter = {
  event: ['keydown', 'blur'],
  fn: (event, env) => {
```

```
    env.stop = true;  // stop the event
  }
}
```

Examples

- Throttle and Debounce for event-input and mousemove
- Custom modifier/alias

CHAPTER 3

Direct directive

It lets you call js function with ability of a directive. Syntax: `function-name!="arguments"` A function can be global (optional) or in scope. You can pass arguments, also available builtins arguments: `this (scope), $element, $env`. By default a function receives scope, element, value, env.

Listing 3.1: Example

```
<p main-app!></p>
<p some-func!="this, $element, 123"></p>
```

Listing 3.2: js part

```
function mainApp(data, element, value, env) {
    data.someFunc = function(data, element, value) {
    }
}
```

Example on jsfiddle

CHAPTER 4

Change Detector

It lets you observe changes in your scope

ChangeDetector.watch(name, callback, option)

Set the tracking variable. Also you can track system events, it returns object with method stop()

Name:

- **<expression>** - Expression/model
- **<a function>** - Track the result of the function, the function are called every iteration of $scan.
- **"$destroy"** - Track a destroying scope
- **"$any"** - Track a modifying any object
- **"$finishScan"** - a callback is executed as soon as $scan finish work
- **"$finishBinding"** - the callback is called when a binding process finishes, sample
- **"$finishScanOnce"**
- **"$onScanOnce"** - the callback is called in scan loop

Callback:

The callback is called with two parameters. The first parameter contains the object after it has been changed, the second parameter contains the object before it has been changed. If the watched variable is not initialized, the second parameter is undefined.

Option:

- **option** = true or **option.isArray** = true - watch an array
- **option.readOnly** = true - You can use it if the *callback* doesn't modify the scope. (an optimization option).
- **option.deep** = true | integer - a deep comparison for the object, watches 10 hierarchy depth levels by default, as alternative an integer which contains the depth of levels to watch.

- **option.isArray**
- **option.OneTime**
- **option.onStop**

Optimization tip: If *callback* returns '$scanNoChanges' then $scan will not run extra-loop (like readonly watch)

ChangeDetector.watchGroup(expression, callback)

Watches a group of expressions. There are two approaches which can be used:

1. It returns a group object. You need to manually add more watcher objects to the group (Example).
2. Use an array as expression and watch it's items (Example).

expression:

- array of <expressions> - items which will be watched

callback

- will be executed when any item of the watched group has been changed

ChangeDetector.compile(expression, option)

Compile an expression

option:

- **option.input** - list of input arguments
- **option.no_return** - a function will not return any result (compile a statment)
- **option.string** - result of method will convert to string

Listing 4.1: Example of $compile

```
var scope = {};
var cd = alight.ChangeDetector(scope)
var fn = cd.compile('"hello " + title')
scope.title = 'linux'
fn(scope) // return "hello linux"
scope.title = 'macos'
fn(scope) // return "hello macos"
```

Listing 4.2: Example with input

```
var fn = cd.compile('title + v', { input:['v'] })
fn(scope, ' X') // return "macos X"
```

Listing 4.3: Example with no_return

```
var fn = cd.compile('title = v', { input:['v'], no_return:true })
fn(scope, 'linux') // scope.title = "linux"
```

ChangeDetector.eval(expression)

Execute an expression

ChangeDetector.watchText(tpl, callback)

Track the template

ChangeDetector.new([scope])

Create a child ChangeDetector, if scope is omitted, then it will used parent scope

ChangeDetector.destroy()

Destroy the Scope.

ChangeDetector.scan(callback or option)

Starts the search for changes, returns a watch statistic

- **callback** - Method will be called when $scan finishes a work, even if $scan has already started from other a place.
- **option.callback** - see above
- **option.skipWatch** - skip specific watch
- **option.late =** *(true/false)* - If there is a few $scan commands, Angular Light will call only last one.

Listing 4.4: Example with $scan

```
var scope = {};
var cd = alight.ChangeDetector(scope);
cd.watch('title', function(value) {
    console.log('title =', value)
}); // make observing
scope.title = 'new'
cd.scan()
// print title = new
scope.title = 'linux'
cd.scan()
// print title = linux
cd.scan()
// do nothing
```

ChangeDetector.getValue(name)

Take the value of the variable, also you can use ChangeDetector.eval

ChangeDetector.setValue(name, value)

Set the value of the variable

Listing 4.5: Example with $setValue

```
var scope = {}
scope.var = 1;
scope.path.var = 2;
scope.path[scope.key] = 3;

// equal
var scope = {}
var cd = alight.ChangeDetector(scope);

cd.setValue('var', 1);
cd.setValue('path.var', 2);
cd.setValue('path[key]', 3);
```

CHAPTER 5

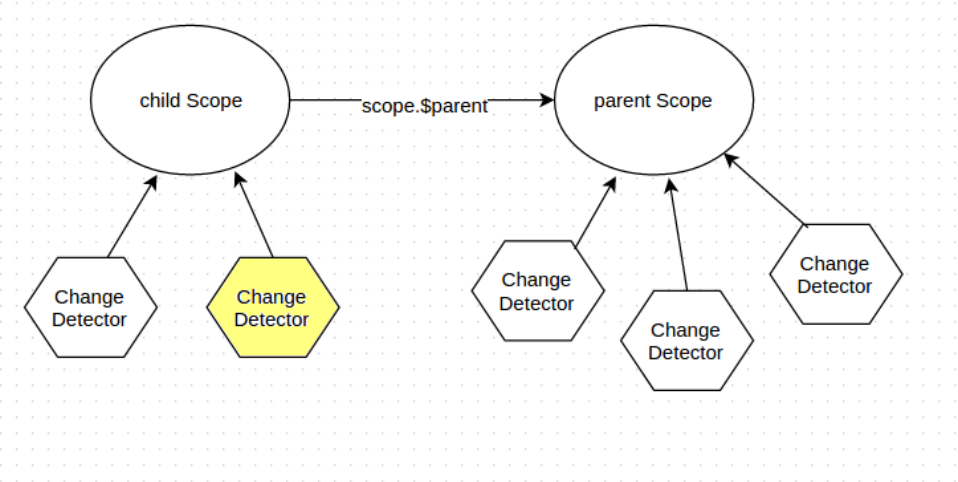
How does it work?

- Scope is a object with user's data which can have a lot of information, it doesn't have own functional, $scope.$watch is a just a link to activeChangeDetector.watch
- ChangeDetector - is a dirty-checking tool which implement "watch", "scan" etc.
- one ChangeDetector instance observes only one object (e.g. scope)
- Different directives makes own ChangeDetectors and observe your data (your scope), so a few (10, 20) change detectors can observe the same scope. e.g. al-repeat observes every item of array, al-if and al-include make own CD for child template with the same scope.

You can't call $watch or $new anytime. When you call scope.$parent.$watch, your parent can have a few CD, and your parent doesn't know which CD should serves your watch-expression. It's why you should take needed CD and call CD.watch.

On the other side you can call scope.$watch (inside link function) because in this moment one of the CD is active, so scope.$watch calls activeCD.watch

Any other scope's methods doesn't have this problem and they can be called any time, ($eval, $getValue, $setValue etc)


child Scope
scope.$parent
parent Scope
Change Detector
Change Detector
Change Detector
Change Detector
Change Detector

CHAPTER 6

Create Directives

Directives should be placed in **alight.directives.prefix**, you can choose any prefix, for example al-text (*alight.directives.al.text*), bo-text (*alight.directives.bo.text*), also you can place your directives in *scope.$ns.directives* they will be private.

A hyphen should be changed to underscores, example: `<input al-my-some-directive />` it will be in alight.directives.al. **mySomeDirective**

The prefixes are needed in order to be able to catch the lack of used directives. For example, if you use the directive "al-texta", but it does not exists (a mistake in the name or js-file isn't loaded), then aLight will throw an exception.

An example of directive **al-text**, the directive is called when the binding process comes to an DOM-element

Listing 6.1: Example of directive al-text

```
alight.directives.al.text = function(scope, element, expression, env) {
    // Track the given expression
    this.watch(expression, function(value) {
        // set a text to the DOM-element
        $(element).text(value)
    });
};
```

Input arguments:

- **scope** - current Scope
- **cd** - change detector
- **element** - element of DOM
- **name** - value of attribute
- **env** - access to different options
- env. **attrName** - name of attribute (directive)
- env. **attributes** - list of attributes

- env. **takeAttr(name, skip=true)** - take a value of the attribute, if skip=true then the attribute will skip a binding process, sample
- env. **skippedAttr()** - list of not active attributes
- env. **stopBinding** = false - stop binding for child elements
- env. **changeDetector** - access to ChangeDetector
- env. **parentChangeDetector** - access to parent ChangeDetector (if scope was changed)

Attributes of directive:

- **priority** - you can set priority for a directive
- **template** - custom template
- **templateUrl** - link to template
- **scope** (false/true) - if it is true, there will be a new, empty scope
- **ChangeDetector** (false/true/'root') - create a new change detector
- **restrict** = 'A', can be 'AEM', 'A' matches attribute name, 'E' matches element name, 'M' matches class name
- **link** - the method is called after template, scope
- **init** - the method is called when the directive is made, before template, scope. You usually need **link** instead of it.
- **stopBinding** (false/true) - stop binding for child elements
- *anyAttr* - you can make a custom attribute, look. directive preprocessor

Listing 6.2: Directives with attributes

```
alight.directives.al.stop = {
    priority: -10,
    template: '<b></b>',
    stopBinding: true,
    link: function(scope, element, name, env) {
    }
};
```

If "stopBinding" is true, then the process of binding will skip child DOM-elements, it is necessary for the directives which are themselves controlled subsidiary of DOM, such as al-repeat, al-controller, al-include, al-stop, etc.

When you try to bind the same element, you have to skip used directives, otherwise you will have a recursion problem. You can use option **skip_attr: env.skippedAttr()** for this, **env.skippedAttr()** gives you a list of used directives.

```
alight.directives.al.stop = {
    stopBinding: true,
    link: function(scope, element, name, env) {
        alight.bind(scope, element, {
            skip_attr: env.skippedAttr()
        })
    }
};
```

CHAPTER 7

Inheritance of directives

If you want make inheritance of a directive, you need call a parent directive after that you can replace methods of the directive. For example, **al-value** has a few methods:

- onDom - binding to DOM
- updateModel - updateing the model
- watchModel - $watch model
- updateDom - updateting DOM
- initDom - set first value to DOM, updateDom(init_value)
- start - It's called when the directive is ready

Make a directive al-value with deferred updating of model:

Listing 7.1: Inherit al-value

```
alight.directives.al.valueDelay = function(scope, cd, element, value, env) {
    // create a source directive
    var dir = alight.directives.al.value(scope, cd, element, value, env);

    // save the old method for update the model
    var oldUpdate = dir.updateModel;
    var timer = null;

    // change the method
    dir.updateModel = function() {
        if(timer) clearTimeout(timer);
        timer = setTimeout(function() {
            timer = null;

            // call the default method for update the model
            oldUpdate();
        }, 500);
    }
```

```
    return dir;
}
```

Examples of inheritance

Examples below for v0.10, but in v0.11 it works similar * al-value -> al-value-delay * al-show -> al-show-slow * al-repeat, add “$even”, “$odd” into the directive * al-repeat, change input expression my-repeat=”list.foreach item” * al-repeat, change rendering

CHAPTER 8

Directives

Controls

- al-model - two-way binding to user input elements (Example). This directive is a proxy for al-checked, al-radio, al-value and al-select.
- al-checked, todo sample
- al-radio sample1 sample2
- al-value, todo sample
- al-select + al-option, example

Special directives

- al-app, init application with current element, examples
- al-cloak, hide current element until activate the application, examples
- al-html
- **al-if**, sample with animation
- al-ifnot, sample with animation
- al-init
- al-repeat
- al-stop, stops a bind process for the element and his children.

CHAPTER 9

Base filters

date

- To convert the date to string
- Input argument: date

Listing 9.1: example

```
<div>{{when | date 'yyyy-mm-dd' }}</div>
```

filter

- To filter the list
- Input argument: variable

Example

slice

- To slice the list
- input arguments: numbers

Listing 9.2: example

```
<div al-repeat="it in list | slice a"></div>
<div al-repeat="it in list | slice a,b"></div>
```

toArray

- converts an object to array (for al-repeat)
- input arguments: key, value

Listing 9.3: example

```
<div al-repeat="item in object | toArray key,value track by key">
```

Example

orderBy

- sorts an array by key (for al-repeat)
- input arguments: key, reverse

Listing 9.4: example

```
<div al-repeat="item in array | orderBy key,reverse">
```

Example

throttle

- makes delay for output, pattern *debounce*
- input arguments: delay

Listing 9.5: example

```
<input al-value="link" type="text" />
<p>{{link | throttle 300 | loadFromServer}}</p>
```

Example

json

- display data in json style

Listing 9.6: example

```
{{data.value | json}}
{{this | json}}
```

storeTo

- store result value to scope

Listing 9.7: example

```
{{data.value | modifier | storeTo key}}
```

CHAPTER 10

Create filters

Overview

A filter can be placed in **scope** or in **alight.filters**, you can return Promise for deferred call.

Listing 10.1: Example filter

```
alight.filters.wrap = function(value, text) {
    return text + value + text
}
```

Example on jsfiddle

Input arguments

- expression - an input expression
- scope - scope

The filter should return a handler/function, one instance of filter.

CHAPTER 11

Text bindings

It's a way to draw information to DOM:

```
<div>Hello {{name}}!</div>
<a href="http://example.com/{{link}}>link</a>
```

Also you can use method "bind once" for optimization, for that you should append "=" to begin of expression.

```
<div>Hello {{=name}}!</div>
<a href="http://example.com/{{=link}}>link</a>
```

Also you can use one time binding

If you can't use tags `{{ }}`, you can change this to `{# #}`, `{< >}`, `## ##` or something like this, length of tags should be equal 2 symbols:

```
alight.utils.pars_start_tag = '{#';
alight.utils.pars_finish_tag = '#}';
```

For complex text bindings you can use text directives

CHAPTER 12

Text directives

Overview

An able to control a declarative data binding in the HTML

Listing 12.1: example how to use text directives

```
<div al-app>
    counter {{#counter}}
</div>
```

Listing 12.2: text directive counter

```
alight.text.counter = function(callback, expression, scope) {
    var n = 0;
    setInterval(function(){
        n++;
        callback(n)      // set result
        scope.$scan()        // $digest
    }, 1000);
}
```

Input arguments

- **callback** - a function to set a value
- **expression** - expression of directive
- **scope** - scope
- **env** - extra functional
- env. **finally** - a function to set the final value, after that $watch will be removed.

- env. **setter** = callback
- env. **setterRaw** = send value directly

Examples

- Information output delay
- An counter
- An counter with setterRaw
- Sample with bindonce

CHAPTER 13

One-time binding

Overview

Waits when an expression has a value (a non-undefined value), then stops watching. You need append "::" in front of expression for using One-Time binding. It works with $watch, $watchText, directives and declarative bindings.

Listing 13.1: example

```
<div class="red {{::class}}"> {{::text}} </div>
<div al-show="::visible"></div>
<li al-repeat="it in ::list">...</li>
and so on
```

Examples

- Sample with declarative bindings
- Sample with al-repeat

CHAPTER 14

Isolated Angular Light

Angular Light can be invisible/inaccessible for alien code, you should make a function **alightInitCallback(alightBuilder)**, it has to be callable for Angular Light. Angular Light starts the function and gives "alightBuilder()" as an argument on load.

It's useful feature, when your webapp will be used on alien page. It lets use different versions of Angular Light on the same page.

- Example 1
- Example 2

CHAPTER 15

A few ways to bind a model to the DOM

1. Auto binding, al-app

alight() is called on start system, it takes each element with **al-app** and bind it.

Listing 15.1: html

```
<div al-app al-init="title='Hello!'">
    <input al-value="title" type="text" class="form-control" />
    <p>{{title}}</p>
</div>
```

Example on jsfiddle

2. Manual binding with alight()

You can bind custom elements with **alight()**

Listing 15.2: html

```
<div id="app" al-init="title='Hello!'">
    <input al-value="title" type="text" class="form-control" />
    <p>{{title}}</p>
</div>
```

Listing 15.3: javascript

```
alight('#app');  // bind to DOM
```

Example on jsfiddle

3. To bind to element with no DOM

Listing 15.4: html

```
<div id="app"></div>
```

Listing 15.5: javascript

```
var tag = document.createElement('div');  // element
tag.innerHTML = '<input al-value="title" type="text" class="form-control" /><p>{
↪{title}}</p>';  // template

alight(tag, {title: 'Hello!'});  // binding

document.querySelector('#app').appendChild(tag);  // append to DOM
```

Example on jsfiddle

4. Manual binding #2

Listing 15.6: html

```
<div id="app">
    <input al-value="name" type="text" />
    {{name}} <br/>
    <button @click="click()">Set Hello</button>
</div>
```

Listing 15.7: javascript

```
var data = {
    name: 'Some text',
    click: function() {
        data.name = 'Hello';
    }
};

alight('#app', data);
```

Example on jsfiddle

CHAPTER 16

Directive preprocessor

Directive preprocessor lets you control process of creating directives. You can make custom attributes and make different transformations.

Objects:

- **alight.directivePreprocessor** - a default preprocessor, you can change it
- **alight.hooks.directive** - a list of handlers, you can append/remove them

Listing 16.1: Example how to create attribute 'bold'

```
alight.hooks.directive.ext.splice(1, 0, {
    code: 'bold',  // not necessary
    fn: function() {
        if(this.directive.bold) this.element.innerHTML = '<b>' + this.element.
↪innerHTML + '</b>'
    }
})
```

Listing 16.2: How to use it

```
alight.directives.al.example = {
    bold: true
}
```

- Example on plunkr
- Article [ru]: «»

CHAPTER 17

Component

- Component is an isolated part of application, it has own change detector.
- Component has an isolated scope.
- Defined properties (:rating, :max in example below) are transferred to scope of component.
- You can choose a template returning template/templateId/templateUrl.
- Events onStart and onDestroy.
- If you use a prefix for name, then it throw error if a component doesn't exist (e.g. al-component).
- You can expose api from component
- Type of watchers for props: copy, array, deep and true (or empty).

Listing 17.1: Syntax

```
alight.component('rating', (scope, element, env) => {
  return {
    template,
    templateId,
    templateUrl,
    props,
    onStart,
    onDestroy,
    api
  };
})
```

Listing 17.2: Example of componenet

```
<rating :rating="rating" :max="max" @change="rating=$value"></rating>
```

Listing 17.3: Example of componenet

```
alight.component('rating', (scope, element, env) => {
  scope.change = (value) => scope.$dispatch('change', value)
    return {template: `<ul class="rating"><li al-repeat="_ in max" al-class="filled:
↪$index<rating" @click="change($index+1)">\u2605</li></ul>`};
})
```

If you want to define callback on property change or change type of observing of a property, you can do this in "props"

Listing 17.4: Configuring properties

```
alight.component('somecomponent', (scope, element, env) => {
  return {
    props: {
        prop0: function(value) {},
        prop1: {
            watchMode: 'array'
        },
        prop2: {
            watchMode: 'deep',
            onChange: function(value) {}
        }
    }
  };
})
```

Listing 17.5: Props as list

```
alight.component('somecomponent', (scope, element, env) => {
  return {
    props: ['prop1', 'prop2']
  };
})
```

Example on jsfiddle

CHAPTER 18

FAQ

How can I take a scope?

for Angular Light v0.10 and older

- Have a scope by tags
- Publish a scope from controller
- Publish a scope using a directive
- Take a scope by a name of controller
- A way to take Scope by element - alight.getScopeByElement(element)
- A way to take Scope by element with a directive
- How to call Alert from al-click
- **Where can I take a debug version** Here you can get release and debug of any version, also you can use bower: `bower install alight`
- **Where is $http service** Angular Light doesn't have it, you can use jQuery.ajax or anyone that you usually use. But there is `alight.f$.ajax` for inner purpose.
- **Angular Light updates a whole DOM or it tracks changes and update only changed elements?** Only changed elements.
- **I need Angular Light runs my function when a bindings process will finish.** You can observe it `scope.$scan("$finishBinding", callback)`
- **Is it possible to pass jquery $element to function from al-click?** You can use $event that contains element, al-click="someFunc($event)"
- **How can I use al-repeat not as attribute, but as a comment?** al-repeat supports comment binding, example
- **Why al-show with an empty array doesn't hide my element: `al-show="model.array"`?** Because there is javascript and !!model.array always give you true, you can use `al-show="model.array.length"`

- **Where is "$odd, $even, $first, $last, $rest" for al-repeat?** You can append any attributes, example how to append $odd $even
- **How to sort an array?** Manual sort Filter orderBy Sort props of an object Call a method
- **Can I rename directives?** Yes, `alight.directives.al.myKeypress = alight.directives.al.keypress`
- **How to call my function when Scope.$scan finish digets process?** You can pass your function as callback `Scope.$scan(callback)`
- **How to redraw bind-once?** Example
- **Where is DI?** Angular Light doesn't have it, but you can make it for yourself, example
- **Why al-repeat displays not all items?** Probably a input list contains doubles, in this case you should declare ID - using **track by**

```
<div al-repeat="item in list track by $index">
```

```
<div al-repeat="item in list track by items.id">
```

Look at al-repeat