[Official Changelog]

alpha 55 features

- It's beta!

alpha 52 features

- Angular is now fully camel case
  - no more converting a "myVariableName" in a class to "my-variable-name" in a template

alpha 50 features

- testing change: package angular2_testing to prepare it for publishing

alpha 47 features

- breaking changes
  - adds an "ng" prefix to all lifecycle hook methods, e.g.
    - onInit -> ngOnInit

The CoveNew undo is my right to down how to do thatnantalpha 41 features

- breaking changes
  - many API deprecations, e.g.
    - bind -> providedfddddddddddddddddddddddddddddddddddddddddddddd
  - angular2/test_lib is now called angular2/testing
    - test_lib.js -> testing.js
    - import {...} from 'angular2/test_lib' -> import {...} from 'angular2/testing'

alpha 40 features

- core: add syntax sugar to make @View optional
  - dropped the @View annotation from our samples, since you can now specify the view in the @Component annotation
  - @View still exists, but shouldn't be necessary unless you have multiple views

alpha 38 features (many changes, see [changelog])

- Breaking changes
  - in Directives, Components, etc.
    - properties -> inputs
    - events -> outputs
  - use "import angular2.dart" if you use angular API because bootstrap.dart no longer implicitly includes those APIs
  - Dart scripts should be moved from the HTML body to the head, as part of an effort to standardize what DartPad, stagehand, and dartlang.org samples/templates do
  - many more, see changelog

alpha 37 features

- core: remove the ^ syntax and make all DOM events bubbling

alpha 35 features (many changes, see changelog)

- change_detection: added an example demonstrating how to use observable models (52da220), closes #3684
- Breaking changes
    - Rename all constants to UPPER_CASE names, e.g.:
        - appComponentTypeToken => APP_COMPONENT
    - Renamed DI visibility flags, e.g.:
        - PRIVATE => Visibility.Private
    - Renamed all "annotation" references to "metadata", e.g.:
        - *Annotations => *Metadata
        - renderer.DirectiveMetadata => renderer.RendererDirectiveMetadata
    - Pipe factories have been removed and Pipe names to pipe implementations are 1-to-1
    - Instead of configuring pipes via a Pipes object, now you can configure them by providing the pipes property to the View decorator

alpha 34 features

- WebWorkers: Added a WebWorker Todo Example
- Breaking changes
    - Directives that previously injected Pipes to get iterableDiff or keyvalueDiff, now should inject IterableDiffers and KeyValueDiffers
    - Previously, if an element had a property, Angular would update that property even if there was a directive placed on the same element with the same property. Now, thdirective12 would have to explicitly update the native element by either using hostInputs or the renderer

alpha 33 features

- Http calls now fire complete() when they finish.
- Breaking change:
    - View renderer used to take normalized CSS class names (ex. fooBar for foo-bar). With this change a rendered implementation gets a class name as specified in a template, without any transformations / normalization. This change only affects custom view renderers that should be updated accordingly.

alpha 32 features(“”)

- Bootstrap no longer requires explicit reflection setup
    - The angular2.dart, reflection.dart, and reflection_capabilities.dart imports are replaced with an import of bootstrap.dart
    - The statement `reflector.reflectionCapabilities = new ReflectionCapabilities()` was deleted

alpha 31 features(“”)

- All form directives now have control attributes (e.g. touched(), value(), valid())

alpha 30 features(“”)

- Supports @Injectable on static functions

alpha 29 features (“”)

- Added the NgStyle directive (documentation is missing)
- Removed appInjector property. Change to viewInjector or hostInjector
- Routing supports deep-linking to anywhere in the app

alpha 28 features (documentation work in progress)

- Routing supports hash-based location

Features added in alpha 27 (documentation work in progress)

- If you want your emitter named differently from your event, this now works in @Component
    - outputs: ['myEmitterName: myeventname']
- Added Http class (handles e.g. HttpRequests)
- Routing supports routing to async components
- Added form features

Features added in alpha 26 (documentation work in progress)

- Elvis operator
- using pipes in List of properties (no longer a map)); new syntax
- WIP
    - testing a component using TestComponentBuilder, like this
        - NOTE: TestComponentBuilder is integrated with /test_lib, which is integrated with Guinness. To use it at the moment, you’ll need to use Guinness for your tests
    - for Forms,
        - add a Form example using status classes: ["ng-binding", "ng-untouched", "ng-pristine", "ng-invalid"]
        - use ".touched" property
        - use [(ng-model)]="foo"
        - use nested find in the form '.find("nested/two").value' and '.find(["nested"], ["two"]).value'

Elvis operator: syntactic sugar for if-then around a potential null

```
# ternary
(a != null) ? a.b : null

# Elvis
a?b.
```

- So named because it looks like an Elvis emoticon

Useful items that may not be well known

- events like keyup support syntactic sugar like this: <div (keyup.enter)="callback()">

```
# add common keys to events to only fire the callback for that key or combo
<div (keyup.enter)="callback()">
<div (keydown.shift.enter)="callback()">
```

- in a template, to bind to an attribute you must prepend “attr.” to the attribute name
    - see this and this for far more detail

```
# bind to a property
<my-element [title]="myTitle"...

# bind to an attribute
<div [attr.contenteditable]="myBoolean"...
```

- Events named using camelCase in dart code is accessible by words separated by dashes in template HTML - “camel-case”

```
Dart Source:

@Component(
    selector: ‘test-component’
    outputs: const [‘onChange’])                          ←------ Event named “onChange” in code
@View(
…)
class TestComponent {
…
}


Template using TestComponent:
...
<test-component (on-change)=”closeHandler()”>    ←--------- Event used as “on-change” in template
...
```

- Two-way binding in Angular2 (TBD)

- Note that selector in a @Component or @Directive is actually a CSS selector. So be careful when you convert a Component to a Directive or vice-versa. The directive has extra [and] around the name.

Directives

- Directive: Directive annotation + controller class
    - attaches behavior to 1 or more DOM elements
        - selector matches DOM
        - ElementInjector resolves the constructor args and injects
            - other Directives
            - element-specific special objects
            - delegates to the parent injector
        - Directives are instantiated html-depth-first

- Injection
    - classes marked @Injectable set tooling to show as available for dependency injection
    - in class constructor, if “@Optional() myInjectable” then a null myInjectable object won’t throw an error
    - parent Directives are instantiated before child Directives
        - so a Directive can't inject the list of its child Directives
        - instead, inject a QueryList, which updates its contents as children are added, removed, or moved

```
class MyDirective {MyDirective(@QueryChildren(Marker) QueryList<Marker> kids)}
class MyDirective {MyDirective(@QueryDescendents(Marker) QueryList<Marker> kidsKids)}
```

    - can inject instances from closest components or their parents
        - e.g. below, “my-directive” can inject Injectables declared in “SomeClass”

```
<div some-class="3" my-directive>
---
@Directive(selector: '[my-directive]')
// SomeService is declared as @Injectable in SomeClass
class MyDirective {MyDirective(SomeService someService) {...}}
class MyDirective {MyDirective(SomeClass someClass) {...}}
// get the parent of SomeClass, even if SomeClass would satisfy...
class MyDirective {MyDirective(@Parent() SomeClass someClassParent) {...}}
// get an ancestor of SomeClass, even if SomeClass would satisfy...
class MyDirective {MyDirective(@Ancestor() SomeClass someClassAncestor) {...}}
```

- the <template> causes a ViewContainerRef to be created which can be injected into Foo
    - the Directive uses it to instantiate, move, add and delete views
    - new views created in Foo will be siblings of <template> like the last line below:

```
...
Foo(this.viewContainer = ViewContainerRef, this.protoViewRef = ProtoViewRef);
...
this.viewContainer.create(this.protoViewRef);
...
<template [foo]="bar">
  <li title="text"></li>
</template>
<li title="text"></li>
```

- constructor parameters

    - selector:string, // element(s) to match and bind to, with these options:

```
element-name: select by element name
.class: select by class name
[attribute]: select by attribute name
[attribute=value]: select by attribute name and value
:not(sub_selector): select only if the element does not match the sub_selector
selector1, selector2: select if either selector1 or selector2 matches
```

- inputs:any, // Directive property to set from the value of a DOM property

```
inputs: [
  'aProperty',
  'renamedProperty': 'renamed-property',
  'transformedProperty': 'alsorenamed-property | aSecondTranformation | ...'
  // piped functions will be applied in order: aFirstTranformation, aSecondTranformation
}
---
<div [some-property]="someExpression">...</div>
<div some-property="Some Text">...</div>
<div [another-property]="anotherExpression | aFirstTranformation">
```

- outputs:List, // events this component emits via EventEmitter
- hostListeners: any, // listen for host element events, take the specified action
  - To listen to global events, a target must be added to the event name. The target can be: window, document or body

```
hostListeners: {
  'event1': 'onMethod1(arguments)', // if exp returns false, event1.preventDefault()
  'window:event2': 'onMethod2(arguments, $event, $target)'
}
```

  - in the directive event binding, these variables can be used
    - $event: Current event object which triggered the event.
    - $target: The source of the event. This will be either a DOM element or an Angular directive. (will be implemented in later release)
  - hostInputs: any, // map of DOM element properties to update when the matching class var changes

```
hostInputs: {
  'some-property': 'myVariable' // is this correct or reversed?
}
```

- lifecycle:List, // hostListeners in which this Directive participates, e.g. onChange
  - onDestroy, onChange, onAllChangesDone
- compileChildren:boolean // if false, don't compile this Directive's children, default true

Directive functions

@View(directives: const [NgFor, NgIf, NgNonBindable, NgSwitch, NgSwitchDefault, NgSwitchWhen])

For: iterator

```
<li *for="#error of errors; #i = index">
Error {{i}} of {{errors.length}}: {{error.message}}
</li>
```

If: remove from DOM if condition is false

```
<div *if="errorCount > 0" class="error">
```

Non-bindable: ignore Angular symbols

```
<div non-bindable>Ignored: {{1 + 2}}</div> // output: Ignored: {{1 + 2}}
```

Switch: set the matching nested element **visible** when expression == whenExpression

```
<ANY [switch]="expression">
  <template [switch-when]="whenExpression1">...</template>
  <template [switch-when]="whenExpression2">...</template>
  <template [switch-default]>...</template>
</ANY>
```

*foo: make this element the template and pass foo's value to it

```
<li *foo="bar" title="text"></li>

// is expanded to
<template [foo]="bar">
  <li title="text"></li>
</template>
```

Components

- Component == Directive + an embedded View
    - Each @Component has
        - its own ElementInjector
        - a Shadow DOM
            - each element has its own ElementInjector
- Annotations
    - @Component: when to instantiate the Component (can have only one)
    - @View annotation: specifies the template with its directives (can have one or more)
        - or specifies a custom 'renderer' to use instead
        - see a template literal below
        - templateUrl specifies a path to an html file, relative to the Component file

```
library gnome_pics;
import 'package:angular2/angular2.dart';

@Component(
    selector: 'gnome-pics', // selects the host element, replaced by this
    inputs: const ['isSleeping'], // set isSleeping from attribute
    outputs: ['rate', 'myEmmitterName: myevent'], // emits 'rate' and 'myevent'
    lifecycle: [onChange] // onChange(event), called  when props change
    directives: [MyButton] // a MyButton object will be injected
    )
@View(
    template: '''
      <img *if="isSleeping" id="awake" src="images/happy_gnome.jpeg"/>
      <img *if="!isSleeping" id="sleepy" src="images/sleepy_gnome.jpeg"/>
    '''
    )
class GnomePics {
  bool isSleeping = false;
  GnomeHome gnomeHome;
  String gnomeName;
  EventEmitter rate;

  // inject gnomeHome from the injectables, set gnomeName from an attribute's value
  GnomePics(this.gnomeHome, @Attribute('gnome-name') gnomeName) {
    this.rate = new EventEmitter();;
  }

  void onChange(changes) {
     // will be called when the component's properties change
     rate.add(3.5);
  }
}
---
<gnome-pics
  [gnome-name]="22 + 2 + 'Sleepy'"
  (rate)="onRate($event)"
  isSleeping="true">
</gnome-pics>
```

- Dynamic loading: load a component at run-time

```
@Component(
  selector: 'my-dynamic-gnome-pics'
)
class MyDynamicGnomePics {
  var gnomePics:GnomePics;
  MyDynamicGnomePics(DynamicComponentLoader loader, ElementRef location) {
    loader.load(GnomePics, location).then((gnomePics) {
      this.gnomePics = gnomePics;
    );
  }
}
```

- Specially handle exceptions using ExceptionHandler

```
@Component({
  selector: 'my-app',
  injectables: [
    bind(ExceptionHandler).toClass(MyExceptionHandler)
  ]
```

```
})
@View(...)
class MyApp { ... }
class MyExceptionHandler implements ExceptionHandler {
  call(error, stackTrace = null, reason = null) {
    // do something with the exception
  }
}
```

- annotations beyond Directive's

-
    - injectables: classes to inject (e.g., see GnomeHome above)
    - @Attribute('theAttributeName')
- on Component instantiation, Angular
    - creates a shadow DOM and loads the specified template into it
    - creates a child Injector, configured with the specified Injectables
    - evaluates the template against the Component

Application

- an application is essentially made up of nested Components
    - application html, usually "<my-app>" in /web/index.html
    - component template html syntax:

```
...
<!-show talk.title's value->
{{talk.title}}

<!- pass talk.rating value to this component's "rating" parameter->
<formatted-rating [rating]="talk.rating"></formatted-rating>

<!- when this component's "rate" event arrives, pass it to "onRate"->
<rate-button (rate)="onRate()"></rate-button>
       ...
```

- Property and event bindings are the public API of a component

Core

Notes

- Angular 2 does not compile/process bindings in index.html

Bootstrap Angular

- bootstrap: specify the application's root Component

```
main() {
  return bootstrap(GnomePics);
}
```

- options
  - Class appComponentType: the root component's class
  - List<Binding> componentInjectableBindings: override the "injectables" configuration
    - defaults to null
  - Function errorReporter = null: error reporter method for unhandled exceptions
    - `Future errorReporter(exception:any, stackTrace:string)`

Router

- route from urls and links to specific application states
- show current state in the browser's url bar

```
<a router-link="user">link to the user component from a template</a>
---
router = new Router(RouteRegistry registry, Pipeline pipeline, Location location, Router parent,
hostComponent);

router.config([
  { 'path': '/', 'component': IndexComp },
  { 'path': '/user/:id', 'component': UserComp, alias 'user' },
]);

// utility functions
String url = generate(String name, var params);
Future<String> cannonicalUrl = navigate(String url); // navigate to url
Instruction componentGraph = recognize(String url);
renavigate(); // go to the last url
subscribe(onNext); // subscribe to url updates

// properties
Component hostComponent
String lastNavigationAttempt
bool navigating;
Component parent;
String previousUrl;
```

No docs exist for these, TBD

- activateOutlets(instruction:Instruction)
- traverseOutlets(fn)
- docs say these are probably only needed if writing a reusable component
  - childRouter(outletName = 'default')
  - registerOutlet(RouterOutlet outlet, name = 'default')

Test

TestComponentBuilder

- see this pull request to resolve this issue. TestComponentBuilder currently only works with Guinness tests. A fix is planned to separate Guinness from TestLib, which will solve this problem.

TestBed (will be deprecated)

- create a view with a component in it, to test the component by itself
    - either a component or a context must be specified, both can be specified

```
// create a test bed for the component
TestBed testBed = new TestBed(Injector: injector);

// create the component in a test View
AppView appView = testBed.createView(
Type component,
{context = null, html = null}: {context:any, html: string} = {},
[object Object],
[object Object],
[object Object]);

// override a directive from the component's View.
testBed.overrideDirective(Type component, Type from, Type to, [Object object], [Object object],
[Object object]);

// only override a component's html
testBed.setInlineTemplate(Type component, String html, [Object object], [Object object]);
```

Inject

- Inject dependencies into a test function
- The syntax will change, will become an annotation like:
    - `it('...', @Inject (object: AClass, async: AsyncTestCompleter) => { ... });`

```
beforeEach(inject([Dependency, AClass], (dep, object) => {
  // some code that uses `dep` and `object`
  // ...
}));
it('...', inject([AClass, AsyncTestCompleter], (object, async) => {
  object.doSomething().then(() => {
    expect(...);
    // mark the async test complete, like "done" in Jasmine
    async.done();
  });
})
```

Errors

- If you see an error like this involving the change detector:

Caught Expression 'selected in hostInputs of <div debug-tab title="tab1">' has changed after it was checked. Previous value: 'false'. Current value: 'true' in [selected in hostInputs of <div debug-tab title="tab1">] package:angular2/src/change_detection/abstract_change_detector.dart 286:5 AbstractChangeDetector._throwError

package:angular2/src/change_detection/abstract_change_detector.dart 123:12
AbstractChangeDetector.detectChangesInRecords
package:angular2/src/change_detection/abstract_change_detector.dart 98:10
AbstractChangeDetector.runDetectChanges
package:angular2/src/change_detection/abstract_change_detector.dart 186:12
AbstractChangeDetector._detectChangesInShadowDomChildren
package:angular2/src/change_detection/abstract_change_detector.dart 101:10
AbstractChangeDetector.runDetectChanges
package:angular2/src/change_detection/dynamic_change_detector.dart 106:10
DynamicChangeDetector.checkNoChanges
package:angular2/src/core/life_cycle/life_cycle.dart 80:30 LifeCycle.tick

This sort of exception happens only in test since change detection exceptions are raised only in test mode. To prevent this the selected property had to be changed from a scheduleMicroTask() callback so that it happens in the next cycle.

PageLoader

TODO

Forms module

- it's not part of Angular2 module but must be explicitly imported
- ControlGroups of Controls, groups aggregate form values and errors
- TypeScript example from angular2 source

```
// bind a control group to the form, the login and password controls to their matching elements
@Component({selector: "login-comp"})
@View({
     directives: [formDirectives], // shorthand for getting all form directives, e.g. ControlGroup
     inline: "<ng-form [ng-control-group]='loginForm'>" +
             "Login <input type='text' ng-control='login'>" +
             "Password <input type='password' ng-control='password'>" +
             "<button (click)="onLogin()">Login</button>" +
             "</ng-form>"
     })
class LoginComp {
 ControlGroup loginForm;

 LoginComp () {
   this.loginForm = new ControlGroup({
     login: new Control(""),
     password: new Control("")
   });
 }
 onLogin() {
   // this.loginForm.updateValue
 }
}

// a control that can't be divided into other controls
Control myControl = new Control(var value, Function validator = Validators.nullValidator);
myControl.updateValue(var value);

// a fixed-length form section that contains other controls, aggregates their values and errors
ControlGroup myControlGroup = new ControlGroup(
```

```
  Map<String> controls,
  Map<String> optionals = null,
  Function validator = Validators.group);

// validator values
Forms.VALID
Forms.INVALID

// check whether a control is within a ControlGroup
bool inThere = myControlGroup.contains(String controlName);

// don't know what these are for, there are no docs
exclude(String controlName)
include(String controlName)

// a variable-length form section that contains other controls, aggregates their values and errors
ControlArray myControlArray = new ControlArray(
  List<AbstractControl> controls
  Function validator = Validators.array);

// properties
Array controls = myControlArray.controls;
Control myControl = myControlArray.at(int index);
myControlArray.push(AbstractControl control);
myControlArray.insert(index:number, AbstractControl control);
myControlArray.remove(index:number, AbstractControl control);

int howMany = myControlArray.length;

// create a Form object from a configuration
var loginForm = builder.group({
   login: ["", Validators.required],
   passwordRetry: builder.group({
     password: ["", Validators.required],
     passwordConfirmation: ["", Validators.required]
   })
 });

// add controls to a form
loginForm.array(List controlsConfig, Function validator = null);

// add a control to a form
loginForm.control(value, Function validator = null);

// add a group to a form, like the above
loginForm.group(controlsConfig, extra = null)

// shorthand-specify all @View form directives, e.g. ControlDirective, ControlGroupDirective, etc.
@View({
   directives: [FormDirectives],
   ...
})
```

Transformers

Angular2

From pubspec.yaml:

```
transformers:
  - angular2:
      entry_point: web/index.dart
      #reflection_entry_point: web/index.dart #not needed if this is the "entry_point" file
```

```
    custom_annotations:
       - name: MyComponent # The name of the class.
       import: 'package:my_package/my_component.dart' # The import that defines the class.
       superClass: Component # The class that this class extends.
```

- entry_point: where you call bootstrap(myModule)
    - see the Transformer docs
- reflection_entry_point: the file where ReflectionCapabilities is specified
    - not needed if this is the same as the "entry_point" file
- custom_annotations: your custom versions of the built-in anotations like @Component and @Injectable

Dependency injection (Ignore for now, WIP, not tested)

Goal

Instantiate object independently of the methods that use them, decoupling which allows easier testing, code reuse and the ability to work in different environments (see Vojta Jina's ng-conf talk for more)

Annotations

```
class MyService {...} // make a service that others will inject

//---main.dart---
main() {
    (new Injector());
}

//---someclass.dart---
@Inject(MyService)
class SomeClass {
    var myService;

    SomeClass(MyService this.myService);

    getSomething() => myService.getIt();
}
```

In testing

```
@Provide(MyService)
class MockMyService {...}


---someclass_test.dart---
...
main() {
    Injector injector;
    setUp(() {

      injector = new Injector([MockMyService]);
    });

    test('foo should use MyService output to return bar, () {
      var myService = injector.get(MyService); // will return MockMyService
      expect(foo(myService.getIt()), equals(bar));
```

```
    });
}
```

Is it flexible? Yes.

```
InjectPromise, InjectLazy, use factory functions, etc.
Inline injecttion with SomeClass(Inject(
```

Error messages and their meanings

- “Cannot find reflection information on <@Injectable Type>”
    - Your @Injectable dependencies failed to run the Angular2 Transformer
- “Class 'Object' has no instance getter 'value'.”
    - Could be that a dynamic component failed to load but the template was processed
        - issue #2392
- Failed to load "test/my_element_test.dart": Failed to load script at http://localhost:8081/my_element_test.html.polymer.bootstrap.dart.browser_test.dart.
    - The file polymer/bootstrap.dart wasn’t found at load time, e.g. because a Future wasn’t waited for like in this case

Advanced cases (read the docs)

- The latest API docs (none for Dart yet)
- LifeCycle.tick():explicitly process change detection and its side-effects
- ViewContainerRef: manually create, manipulate or destroy views
- Dependency Injection: ways to transform or alias injectables
    - DI errors
- Inject class, future, lazy: more ways to inject objects besides @Optional and @Injectable
- Events
- Pipe: extended change detection options
- About the Angular2 Dart Transformer

Useful Angular2 links

- Nice links summary by Tim Jacobi
- Angular2 Template Syntax

Useful Dart links

- DartPad
- Using JSON
- JsonObject: access JSON as a map of maps
- Standard Pub package layout