Angular 2.0 Packaging and Releasing

*Warning: This document is public.*

# Modules

A bit about terminology. We will call a *module* a Node or Dart package, and a *library* individual files that can be imported.

Angular 2 will consist of a number of modules and libraries that are required to build Angular. The goal is for the modules to be released separately, so they can be used in isolation by other projects.

Current list of Angular 2 modules and libraries:

( these modules will be published as they make up angular )

* angular2 - entry point into angular, this will be module everyone depends on. It will consist of the following libraries:
  + core - parser, compiler, element injectors etc
  + di - service injectors
  + change detection
  + facade
  + directives
  + reflection
  + test\_lib
    - merges previous test\_lib and e2e\_test\_lib
* benchpress - optional package used to write your end-to-end tests
* rtts\_asserts

( these modules are not technically part of angular 2.0, but are good for publishing, although not via npm / pub. The angular2 module should \*not\* depend on them.)

* benchmarks
* benchmarks\_external
* examples

External dependencies

* zone.js (right now it has to be manually included on the page by the user of the framework).

To simplify project management and distribution in the near term we will:

* Put all modules in one github repo at <https://github.com/angular/angular>
* Put essential runtime libraries (i.e. libraries shipping with the application) into the angular2 module. This way we have only one package to release and version, and users only need to depend on one package to quickly start building apps. Over time we will move some libraries into their own packages (e.g. di).

# Packaging and versioning

## Packaging

Angular 2 will be a new package and not a continuation of AngularJS 1.x. The reasoning is:

* Many teams will likely migrate from 1.x to 2 gradually. One developer will likely have to work with applications written against 1.x and applications written against 2, and perhaps applications that use both simultaneously. It all depends on the migration strategy a given project chooses to implement. It is easier to manage two packages than two versions of the same package.
* Angular2 and previous versions of Angular are developed in separate GitHub repositories and maintained by different (although overlapping) teams.
* It is expected that new versions of the same package evolve from the same code base incrementally. However, Angular 2 is a rewrite from scratch, and in fact, in a different source language (AtScript).
* Generally anything after 1.0 the package is considered stable. However, Angular 2 will be very unstable for a while.
* While API breakages are expected in major revisions, Angular 2 will be a significant api departure from AngularJS.

## Versioning

Angular 2’s version numbers will start with 2.0.0-alpha.#, where # will start with 1 and increment with each new milestone release, e.g. 2.0.0-alpha.5. “beta” will be used for pre-releases. Stable releases will have no labels.

# File structure

In the repository modules and libraries are organized like this:

modules/

module1/ - its contents become a single versioned package

src/ - implementation code not to be imported directly

library1/ - code pertaining to library1

library2/

test/ - unit-tests

library1/

library2/

docs/ - documentation

library1/

library2/

library1.js - library1 file to be imported

library2.js

package.json

pubspec.yaml

module2/

...

According to the above, Angular 2 file structure for published modules is (implementation files are omitted and marked using ellipsis):

modules/

angular2/

src/

core/...

di/...

change\_detection/...

facade/...

directives/...

reflection/…

test/...

docs/...

angular2.js

core.js

di.js

change\_detection.js

facade.js

directives.js

reflection.js

package.json

pubspec.yaml

benchpress/

src/...

benchpress.js

package.json

pubspec.yaml

## Translation to Dart

During translation to Dart the above directory structure is preserved. The \*.js files are transpiled to \*.dart files. package.json is omitted. The resulting file structure is published to pub as a stand-alone Dart package.

If and when we decide to move a library into its own package the top-level file used to import the library would simply re-export the new package. This will avoid having to force people to update their code. We had success in the past using this approach.

## Translation to ES5

TODO(Tobias)

# Discussion

## Packaging

**Q:** Would angular be published as a version of already existing angular packages (<https://pub.dartlang.org/packages/angular> and <https://www.npmjs.com/package/angular>) or published under a new brand name - “angular2” that starts with version 2.0?

existing package:

* shows continuation

new name:

* less surprise/pain for projects with dependency: “any”

**Decision:** We would push under a new name ‘angular2’.

**Q:** Should we publish all modules separately or publish in bulk at first and later split?

split:

* need to write a script to push all 7 modules
* complicating the versioning/releasing story?

in bulk:

* if we don’t publish the packages separately it would be harder for people to depend on a module, thus reducing the value of the module separation.

**Decision:** We would publish things in bulk first. <https://github.com/angular/angular/issues/520>

## Versioning

**Q:** Would angular 2 be semver’ed? Should we allow for breaking API changes until we hit angular2 3.x?

MISKO: v2.0.0-alpha.0

**Q:** Would individual packages be versioned independently of angular? If yes, how tight would angular’s dependencies on them be?

MISKO: Not until we reach release candidate for v2

# Releases

(**JS**) We would need to release the following permutations:

* es5 compiled single file - angular2.min.js that exposes all angular APIs on a global angular2 object. Should this include zones? traceur? system.js?
* es5 compiled multiple files using System.js or AMD to load dependencies.
* uncompiled es6 sources (with .es6 files renamed to .js), for people who want to transpile Angular along with their application into one bundle.

It is not clear how would NPM handle ES6 / AtScript packages (any info?)

(**Dart**) after build, a script will call `pub publish` in each module in dist/dart/<module>. Relative paths need to be removed from import statements.

Share with:

* Pavel, Caitlin, Rado, Igor

# JS packaging variations

Goals:

* have as little different packaging variations as possible
* support the major use cases that people have
* prevent path mappings as much as possible

With vs without runtime type assertions

* right now we publish builds with and without rtts-assertions in the same repository `angular2` under different versions: \*.dev and \*.prod
  + this marks the versions as prerelease versions, which they are right now but eventually should not be
  + this does not allow a user to depend on angular2 with wildcards and specify whether he would like to get the dev/prod version
* version without type assertions is only really needed when we minify
* BUT: Leaving the type assertions in, even when they point to noop functions still makes the tree benchmark about 8% slower!
* Conclusion:
  + short term:
    - es6: publish dev/prod versions
    - cjs: only publish dev version for now, maybe provide a noop assertion module when requested
  + long term:
    - only publish sources with `assert` statements, and have a tool that is able to remove them (e.g. via Closure Compiler)

Module formats

* Users who author in es5 need an already transpiled Angular2 version
  + Only CommonJS format really important
  + Needs to be in root folder of the package as CommonJS does not support path mappings!
  + Needed for Angular2 serverside tools, e.g. benchpress
  + For client side, es5 people can use browserify
  + We could create UMD modules to also support AMD community, but not sure how important this is, given that browserify exists...
* Users that author in es6 know how to transpile and should transpile Angular2 on their own
  + e.g. to prevent conflicting traceur runtime versions
  + e.g. to package only the needed parts

Summary / folder structure

|  |
| --- |
| /: CommonJS version  package.json: should mention Traceur version / depend on it  core/...  di/...  es6/{dev|prod}: Es6 sources with rtts  assertions enabled/disabled  atscro[t/\*: AtScript sources  es6/{dev|prod}/es5build.sh: script to transpile Es6 sources down to Es5 |