Angular2: observable support

*Status: (Draft, Final)*

*Authors: username@angularjs.org*

*This document is published to the web as part of the public* [*Angular Design Docs*](https://drive.google.com/#folders/0BxgtL8yFJbacUnUxc3l5aTZrbVk) *folder*

# Objective

Allow angular to bind to objects which support observables in constant dirty checking time.

NOTE: In this document the observables refers to a objects which can fire events when their properties changes, (not the reactive observables.)

# Background

NOTE: we are using generic syntax here, but it should work for any one specific implementation of observables.

|  |
| --- |
| interface Observable {  listen(listenerFn: (property:string, newValue:any) => void);  } |

## Example

Let's assume that we have an observable component such as this.

|  |
| --- |
| @Component(...)  @View(...)  class MyComponent implements Observable {  address:Address;    emails: ObservableList<string>;  } |

Notice that there are two kinds of observables which can be done. Observing properties, or observing items in the collection. Let's look at each in turn.

### Collections

Angular can be easily extended to support observation of collections. Notice that the [iterableDiff](https://github.com/angular/angular/blob/c32dbad7478b2945e7fec98d5d440aaa9a0ac2a0/modules/angular2/src/change_detection/change_detection.ts#L29) consist of IterableChangesFactory and NullPipeFactory, we can extended with ObservableChangesFactory, which will check to see if the iterable is ObservableList and register with it.

Because ng-repeat uses iterableDiff internally to process changes, it will automatically know how to process the observable collections.

The result is that the we can use an observable in the template as such.

|  |
| --- |
| <ul>  <li \*ng-for="#email of emails">{{email}}</li>  </ul> |

### Properties

Watching for properties using dirty checking syntax would be done as:

|  |
| --- |
| <span>{{address.street}}</span> |

If we would like to use observables we could implement an obs pipe and use it as such:

|  |
| --- |
| <span>{{this | obs:'address' | obs: 'street'}}</span> |

The result is that the obs pipe could manage registration and release of observables and save angular from doing dirty checking.

The above syntax is verbose, for this reason there is a proposal for AST transformation. This would allow the developer to write a more succinct syntax as follows:

|  |
| --- |
| <span>{{obs: address.street }}</span> |

The obs AST transform would then rewrite the binding in the previous long form:

|  |
| --- |
| <span>{{this | obs:'address' | obs: 'street'}}</span> |

# Processing Changes

Angular makes guarantees about the order in which the bindings are processed. Observables make no such guarantees. For this reason observables can not be processed synchronously. What the observables can do is mark a particular ChangeDetector as detached and then schedule the processing of the changes only when changes have been detected. In this way the order of processing changes can be guaranteed.

In order for the ChangeDetector to be detached, all of the bindings must be observable. Angular could produce a warning when hybrid mode is detected.