ADescribe the mental model, philosophy, and vocabulary of Angular 2 applications. This document is intended for developers and documenters of Angular, not necessarily for users of Angular.

[Objective](#h.fxpk50cps4zs)

[Application](#h.z36qlmmy2kpe)

[Component](#h.fn502jippxx2)

[View](#h.b6j5s8qqa16r)

[Binding: Properties](#h.deoxhoagpy4y)

[Binding: Events](#h.oe3xvy6nekmy)

[Binding: Commands](#h.dixiv13lagx2)

[Component Data Flow in Shadow DOM](#h.ltl9c4uur06v)

[Shadow DOM vs. Light DOM](#h.ehavwx3i370)

[ViewRenderer](#h.krh6beyqrpkj)

[Application environment](#h.f3sbz3qbu67r)

[Component Injector](#h.4gg77k78769j)

[Element Injector](#h.dou19g8qni33)

[Templates](#h.nhwl065xz2ca)

[Zone](#h.xm9rbf3bjjih)

# Application

An Angular application has a root component, which can contain other components. (Note: When this document talks about components, it means *Angular components*.)



* An Angular application is a tree of components
* Data binding allows for automatic movement of data from parent to child component.
* Commands allow a parent event to request a side effect from a child component.
* Events allow a child component to notify the parent of state change.

The following code shows what components, data binding, commands, and events mean to Angular. Angular users don’t write code like this; Angular takes care of it for them.

|  |
| --- |
| // The code people don’t need to write, thanks to Angular:  function ParentComponent() {  var input = new HTMLInputElement( ); // Child component  input.value = 'world'; // Property assignment (data binding)  input.focus(); // command  input.addEventListener('change', function (){  // event notification  });  } |

# Component

An Angular component is a building block that consists of a **controller**, a **view**, and a **detector**.

|  |
| --- |
| // Component annotation  // <prompt question="What is your name?"  // (answer)="greet($event)"></prompt>  @Component({  // PUBLIC BINDINGS  selector: 'prompt',  properties: ['question'],  events: ['answer'],  // VIEW PRIVATE  commands: ['focus'],  listeners: { 'click': 'ask()' }  })  // View annotation  @View({  template: `  <form (submit)="onAnswer(input.value)" [hidden]="!show">  {{question}}  <input #input @focus="input.focus()">  </form>  `,  stylesheets: [],  directives: []  })  // Component controller  class Prompt {  // PUBLIC BINDINGS  question:string;  answer:Emitter;  // VIEW PRIVATE  show:bool;  focus:Emitter;  constructor() {  this.focus = new Emitter();  this.answer = new Emitter();  }  ask() {  this.show = true;  this.focus.call();  }    onAnswer(value) {  this.answer.call(value);  this.show = false;  }  } |

* The **controller** is the class that’s annotated with @Component and @View.
* The **view** is created from the @View annotation.
* The **detector** is created dynamically from the expressions in the @View template.
* The controller instance is the evaluation context for all expressions/statements in the @View.



# View

* A view is a visual representation of the UI.
* Views can be made up of other components or primitives (HTML).
* Input to views:
  + Setting a property on the view updates a child component or HTML element property.
  + Invoking an action on the view invokes a method on a child component or HTML element.
* Output from views:
  + A view can listen to child components or HTML elements and invoke methods on the controller.



## Binding: Properties

View properties are placeholders in the view that can accept values.

View properties are declared using {{exp}} or [prop]='exp' syntax. The view retains the placeholder location for the value. The detector handles watching the exp. When the detector detects a change, it sets the view property to the new value of the expression.



## Binding: Events

Events are used to execute code with side effects, usually due to user UI interactions.

Events are declared using(event-name)="statement" syntax and set up event listeners on the DOM element. When the underlying component fires the event, Angular executes statement on the controller.



## Binding: Actions

Actions allow the controller to invoke methods on the elements in the. (focusing, transitions, ...)

Actions are declared using @name="statement" syntax. When the controller fires an action and the view then invokes methods on the child elements or components.



# Component Data Flow in Shadow DOM

A component is built up from smaller components that are specified in the @View.



## Shadow DOM vs. Light DOM

Shadow DOM is the private implementation of a component.

Light DOM is how existing components are composed together. (A Light DOM is often part of larger Shadow DOM.)

* Source code:
  + Both <my-app> (red) and <zippy> (green) have a template. The template is made up of other HTML elements.
* Runtime DOM representation:
  + At runtime each component gets its own Shadow DOM, into which the component’s view is loaded.
  + <img> is part of <my-app> Shadow DOM.
  + <img> is also part of <zippy>'s Light DOM (it is inside of the <zippy> element).
  + NOTE: Element can be considered both Light and Shadow DOM depending on the point of view.
    - When discussing Zippy behavior we refer to <img> as being in <zippy>'s light DOM.
    - When discussing <my-app> we refer to <img> as part of <my-app>'s shadow DOM.
* Visual representation:
  + Projection. The <img> element is projected from <zippy>'s <content> element. (It behaves as if <img> was inside the <content>.)
  + The <content> element behaves as if the component's light DOM was inside the <content> element.

# ViewRenderer

# Application environment

In Angular an application’s environment consists of:

* Platform Injector
* AngularZone
  + AngularZone sets up Change Detection Domain
* Application Injector (filled with Angular-specific things)



# Component Injector

# Element Injector

# Templates

# Zone