AngularJS to Angular concepts: Quick reference

{@a top}

Angular is the name for the Angular of today and tomorrow. *AngularJS* is the name for all v1.x versions of Angular.

This guide helps you transition from AngularJS to Angular by mapping AngularJS syntax to the equivalent Angular syntax.

See the Angular syntax in this .

Template basics

Templates are the user-facing part of an Angular application and are written in HTML. The following table lists some of the key AngularJS template features with their equivalent Angular template syntax.

```
<th>
  AngularJS
</th>

<th>
  Angular
</th>
```

```
<td>


  ### Bindings/interpolation

  <code-example hideCopy>
    Your favorite hero is: {{vm.favoriteHero}}
  </code-example>


  In AngularJS, an expression in curly braces denotes one-way binding.
  This binds the value of the element to a property in the controller
  associated with this template.

  When using the `controller as` syntax,
  the binding is prefixed with the controller alias (`vm` or `$ctrl`) because you
  have to be specific about the source of the binding.
</td>

<td>


  ### Bindings/interpolation

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.html"
region="interpolation"></code-example>
```

```
  In Angular, a template expression in curly braces still denotes one-way binding.
  This binds the value of the element to a property of the component.
  The context of the binding is implied and is always the
  associated component, so it needs no reference variable.

  For more information, see the [Interpolation](guide/interpolation) guide.
</td>
```

```
<td>


  ### Filters

  <code-example hideCopy>
    &lt;td>{{movie.title | uppercase}}&lt;/td>
  </code-example>


  To filter output in AngularJS templates, use the pipe character (|) and one or more
filters.

  This example filters the `title` property to uppercase.
</td>

<td>


  ### Pipes

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="uppercase"></code-example>


  In Angular you use similar syntax with the pipe (|) character to filter output, but
now you call them **pipes**.
  Many (but not all) of the built-in filters from AngularJS are
  built-in pipes in Angular.

  For more information, see [Filters/pipes](guide/ajs-quick-reference#filters-pipes)
below.
</td>
```

```
<td>


  ### Local variables

  <code-example hideCopy format="">
    &lt;tr ng-repeat="movie in vm.movies">
      &lt;td>{{movie.title}}&lt;/td>
    &lt;/tr>
```

```
  </code-example>


  Here, `movie` is a user-defined local variable.
</td>

<td>


  ### Input variables

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="local"></code-example>


  Angular has true template input variables that are explicitly defined using the
`let` keyword.

  For more information, see the [Structural directive shorthand](guide/structural-
directives#shorthand) section of [Structural Directives](guide/structural-directives).
</td>
```

Template directives

AngularJS provides more than seventy built-in directives for templates. Many of them aren't needed in Angular because of its more capable and expressive binding system. The following are some of the key AngularJS built-in directives and their equivalents in Angular.

```
<th>
  AngularJS
</th>

<th>
  Angular
</th>
```

```
<td>


  ### ng-app

  <code-example hideCopy>
    &lt;body ng-app="movieHunter">
  </code-example>


  The application startup process is called **bootstrapping**.

  Although you can bootstrap an AngularJS application in code,
  many applications bootstrap declaratively with the `ng-app` directive,
```

```
  giving it the name of the application's module (`movieHunter`).
</td>

<td>


  ### Bootstrapping

  <code-example hideCopy path="ajs-quick-reference/src/main.ts" header="main.ts">
</code-example>
  <br>

  <code-example hideCopy path="ajs-quick-reference/src/app/app.module.1.ts"
header="app.module.ts"></code-example>


  Angular doesn't have a bootstrap directive.
  To launch the application in code, explicitly bootstrap the application's root
module (`AppModule`)
  in `main.ts`
  and the application's root component (`AppComponent`) in `app.module.ts`.
</td>
```

```
<td>


  ### ng-class

  <code-example hideCopy format="">
    &lt;div ng-class="{active: isActive}">
    &lt;div ng-class="{active: isActive,
                     shazam: isImportant}">
  </code-example>


  In AngularJS, the `ng-class` directive includes/excludes CSS classes
  based on an expression. That expression is often a key-value control object with
each
  key of the object defined as a CSS class name, and each value defined as a template
expression
  that evaluates to a Boolean value.

  In the first example, the `active` class is applied to the element if `isActive` is
true.

  You can specify multiple classes, as shown in the second example.
</td>

<td>


  ### ngClass
```

```
  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="ngClass"></code-example>


  In Angular, the `ngClass` directive works similarly.
  It includes/excludes CSS classes based on an expression.

  In the first example, the `active` class is applied to the element if `isActive` is
true.

  You can specify multiple classes, as shown in the second example.

  Angular also has **class binding**, which is a good way to add or remove a single
class,
  as shown in the third example.

  For more information see [Attribute, class, and style bindings](guide/attribute-
binding) page.

</td>
```

```
<td>


  ### ng-click

  <code-example hideCopy format="">
    &lt;button ng-click="vm.toggleImage()">
    &lt;button ng-click="vm.toggleImage($event)">
  </code-example>


  In AngularJS, the `ng-click` directive allows you to specify custom behavior when an
element is clicked.

  In the first example, when the user clicks the button, the `toggleImage()` method in
the controller referenced by the `vm` `controller as` alias is executed.

  The second example demonstrates passing in the `$event` object, which provides
details about the event
  to the controller.
</td>

<td>


  ### Bind to the `click` event

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="event-binding"></code-example>
```

```
  AngularJS event-based directives do not exist in Angular.
  Rather, define one-way binding from the template view to the component using **event
binding**.

  For event binding, define the name of the target event within parenthesis and
  specify a template statement, in quotes, to the right of the equals. Angular then
  sets up an event handler for the target event. When the event is raised, the handler
  executes the template statement.

  In the first example, when a user clicks the button, the `toggleImage()` method in
the associated component is executed.

  The second example demonstrates passing in the `$event` object, which provides
details about the event
  to the component.

  For a list of DOM events, see: https://developer.mozilla.org/en-US/docs/Web/Events.

  For more information, see the [Event binding](guide/event-binding) page.

</td>
```

```
<td>


  ### ng-controller

  <code-example hideCopy format="">
    &lt;div ng-controller="MovieListCtrl as vm">
  </code-example>


  In AngularJS, the `ng-controller` directive attaches a controller to the view.
  Using the `ng-controller` (or defining the controller as part of the routing) ties
the
  view to the controller code associated with that view.
</td>

<td>


  ### Component decorator

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.ts"
region="component"></code-example>


  In Angular, the template no longer specifies its associated controller.
  Rather, the component specifies its associated template as part of the component
class decorator.
```

```
  For more information, see [Architecture Overview](guide/architecture#components).

</td>
```

```
<td>


  ### ng-hide
  In AngularJS, the `ng-hide` directive shows or hides the associated HTML element
based on
  an expression. For more information, see [ng-show](guide/ajs-quick-reference#ng-
show).
</td>

<td>


  ### Bind to the `hidden` property
  In Angular, you use property binding; there is no built-in *hide* directive.
  For more information, see [ng-show](guide/ajs-quick-reference#ng-show).
</td>
```

```
<td>


  ### ng-href

  <code-example hideCopy format="">
    &lt;a ng-href="{{ angularDocsUrl }}">Angular Docs&lt;/a>
  </code-example>


  The `ng-href` directive allows AngularJS to preprocess the `href` property so that
it
  can replace the binding expression with the appropriate URL before the browser
  fetches from that URL.

  In AngularJS, the `ng-href` is often used to activate a route as part of navigation.

  <code-example hideCopy format="">
    &lt;a ng-href="#{{ moviesHash }}">Movies&lt;/a>
  </code-example>


  Routing is handled differently in Angular.
</td>

<td>


  ### Bind to the `href` property
```

```
  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="href"></code-example>


  Angular uses property binding; there is no built-in *href* directive.
  Place the element's `href` property in square brackets and set it to a quoted
template expression.

  For more information see the [Property binding](guide/property-binding) page.

  In Angular, `href` is no longer used for routing. Routing uses `routerLink`, as
shown in the following example.

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="router-link"></code-example>


  For more information on routing, see [Defining a basic route](guide/router#basic-
route)
  in the [Routing & Navigation](guide/router) page.

</td>
```

```
<td>


  ### ng-if

  <code-example hideCopy format="">
    &lt;table ng-if="movies.length">
  </code-example>


  In AngularJS, the `ng-if` directive removes or recreates a portion of the DOM,
  based on an expression. If the expression is false, the element is removed from the
DOM.

  In this example, the `<table>` element is removed from the DOM unless the `movies`
array has a length greater than zero.
</td>

<td>


  ### *ngIf

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.html"
region="ngIf"></code-example>


  The `*ngIf` directive in Angular works the same as the `ng-if` directive in
```

```
AngularJS. It removes
  or recreates a portion of the DOM based on an expression.

  In this example, the `<table>` element is removed from the DOM unless the `movies`
array has a length.

  The (*) before `ngIf` is required in this example.
  For more information, see [Structural Directives](guide/structural-directives).
</td>
```

```
<td>


  ### ng-model

  <code-example hideCopy format="">
    &lt;input ng-model="vm.favoriteHero"/>
  </code-example>


  In AngularJS, the `ng-model` directive binds a form control to a property in the
controller associated with the template.
  This provides **two-way binding**, whereby any change made to the value in the view
is synchronized with the model, and any change to the model is synchronized with the
value in the view.
</td>

<td>


  ### ngModel

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.html"
region="ngModel"></code-example>


  In Angular, **two-way binding** is denoted by `[()]`, descriptively referred to as a
"banana in a box". This syntax is a shortcut for defining both property binding (from
the component to the view)
  and event binding (from the view to the component), thereby providing two-way
binding.

  For more information on two-way binding with `ngModel`, see the [NgModel&amp;mdash;Two-
way binding to
  form elements with `[(ngModel)]`](../guide/built-in-directives#ngModel)
  section of the [Built-in directives](guide/built-in-directives) page.
</td>
```

```
<td>
```

```
  ### ng-repeat

  <code-example hideCopy format="">
    &lt;tr ng-repeat="movie in vm.movies">
  </code-example>


  In AngularJS, the `ng-repeat` directive repeats the associated DOM element
  for each item in the specified collection.

  In this example, the table row (`<tr>`) element repeats for each movie object in the
collection of movies.
</td>

<td>


  ### *ngFor

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.html"
region="ngFor"></code-example>


  The `*ngFor` directive in Angular is similar to the `ng-repeat` directive in
AngularJS. It repeats
  the associated DOM element for each item in the specified collection.
  More accurately, it turns the defined element (`<tr>` in this example) and its
contents into a template and
  uses that template to instantiate a view for each item in the list.

  Notice the other syntax differences:
  The (*) before `ngFor` is required;
  the `let` keyword identifies `movie` as an input variable;
  the list preposition is `of`, not `in`.

  For more information, see [Structural Directives](guide/structural-directives).
</td>
```

```
<td>


  ### ng-show

  <code-example hideCopy format="">
    &lt;h3 ng-show="vm.favoriteHero">
      Your favorite hero is: {{vm.favoriteHero}}
    &lt;/h3>
  </code-example>


  In AngularJS, the `ng-show` directive shows or hides the associated DOM element,
based on
```

```
  an expression.

  In this example, the `<div>` element is shown if the `favoriteHero` variable is
truthy.
</td>

<td>


  ### Bind to the `hidden` property

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.html"
region="hidden"></code-example>


  Angular uses property binding; there is no built-in *show* directive.
  For hiding and showing elements, bind to the HTML `hidden` property.

  To conditionally display an element, place the element's `hidden` property in square
brackets and
  set it to a quoted template expression that evaluates to the *opposite* of *show*.

  In this example, the `<div>` element is hidden if the `favoriteHero` variable is not
truthy.

  For more information on property binding, see the [Property binding](guide/property-
binding) page.
</td>
```

```
<td>


  ### ng-src

  <code-example hideCopy format="">
    &lt;img ng-src="{{movie.imageurl}}">
  </code-example>


  The `ng-src` directive allows AngularJS to preprocess the `src` property so that it
  can replace the binding expression with the appropriate URL before the browser
  fetches from that URL.
</td>

<td>


  ### Bind to the `src` property

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="src"></code-example>
```

```
  Angular uses property binding; there is no built-in *src* directive.
  Place the `src` property in square brackets and set it to a quoted template
expression.

  For more information on property binding, see the [Property binding](guide/property-
binding) page.
</td>
```

```
<td>


  ### ng-style

  <code-example hideCopy format="">
    &lt;div ng-style="{color: colorPreference}">
  </code-example>


  In AngularJS, the `ng-style` directive sets a CSS style on an HTML element
  based on an expression. That expression is often a key-value control object with
each
  key of the object defined as a CSS property, and each value defined as an expression
  that evaluates to a value appropriate for the style.

  In the example, the `color` style is set to the current value of the
`colorPreference` variable.
</td>

<td>


  ### ngStyle

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="ngStyle"></code-example>


  In Angular, the `ngStyle` directive works similarly. It sets a CSS style on an HTML
element based on an expression.

  In the first example, the `color` style is set to the current value of the
`colorPreference` variable.

  Angular also has **style binding**, which is good way to set a single style. This is
shown in the second example.

  For more information on style binding, see the [Style binding](guide/attribute-
binding#style-binding) section of the
  [Attribute binding](guide/attribute-binding) page.

  For more information on the `ngStyle` directive, see the [NgStyle](guide/built-in-
```

```
directives#ngstyle)
  section of the [Built-in directives](guide/built-in-directives) page.
</td>
```

```
<td>


  ### ng-switch

  <code-example hideCopy format="">
    &lt;div ng-switch="vm.favoriteHero &&
                    vm.checkMovieHero(vm.favoriteHero)">
        &lt;div ng-switch-when="true">
          Excellent choice!
        &lt;/div>
        &lt;div ng-switch-when="false">
          No movie, sorry!
        &lt;/div>
        &lt;div ng-switch-default>
          Please enter your favorite hero.
        &lt;/div>
    &lt;/div>
  </code-example>


  In AngularJS, the `ng-switch` directive swaps the contents of
  an element by selecting one of the templates based on the current value of an
expression.

  In this example, if `favoriteHero` is not set, the template displays "Please enter
...".
  If `favoriteHero` is set, it checks the movie hero by calling a controller method.
  If that method returns `true`, the template displays "Excellent choice!".
  If that methods returns `false`, the template displays "No movie, sorry!".
</td>

<td>


  ### ngSwitch

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.html"
region="ngSwitch"></code-example>


  In Angular, the `ngSwitch` directive works similarly.
  It displays an element whose `*ngSwitchCase` matches the current `ngSwitch`
expression value.

  In this example, if `favoriteHero` is not set, the `ngSwitch` value is `null`
  and `*ngSwitchDefault` displays, "Please enter ...".
  If `favoriteHero` is set, the application checks the movie hero by calling a
```

```
component method.
  If that method returns `true`, the application selects `*ngSwitchCase="true"` and
displays: "Excellent choice!"
  If that methods returns `false`, the application selects `*ngSwitchCase="false"` and
displays: "No movie, sorry!"

  The (*) before `ngSwitchCase` and `ngSwitchDefault` is required in this example.

  For more information, see [The NgSwitch directives](guide/built-in-
directives#ngSwitch)
  section of the [Built-in directives](guide/built-in-directives) page.
</td>
```

{@a filters-pipes}

Filters/pipes

Angular **pipes** provide formatting and transformation for data in the template, similar to AngularJS **filters**. Many of the built-in filters in AngularJS have corresponding pipes in Angular. For more information on pipes, see Pipes.

```
<th>
  AngularJS
</th>

<th>
  Angular
</th>
```

```
<td>


  ### currency

  <code-example hideCopy>
    &lt;td>{{movie.price | currency}}&lt;/td>
  </code-example>


  Formats a number as currency.
</td>

<td>


  ### currency

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="currency"></code-example>
```

```
  The Angular `currency` pipe is similar although some of the parameters have changed.
</td>
```

```
<td>


  ### date

  <code-example hideCopy>
    &lt;td>{{movie.releaseDate | date}}&lt;/td>
  </code-example>


  Formats a date to a string based on the requested format.
</td>

<td>


  ### date

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="date"></code-example>


  The Angular `date` pipe is similar.

</td>
```

```
<td>


  ### filter

  <code-example hideCopy>
    &lt;tr ng-repeat="movie in movieList | filter: {title:listFilter}">
  </code-example>


  Selects a subset of items from the defined collection, based on the filter criteria.
</td>

<td>


  ### none
  For performance reasons, no comparable pipe exists in Angular. Do all your filtering
in the component. If you need the same filtering code in several templates, consider
building a custom pipe.

</td>
```

```
<td>


  ### json

  <code-example hideCopy>
    &lt;pre>{{movie | json}}&lt;/pre>
  </code-example>


  Converts a JavaScript object into a JSON string. This is useful for debugging.
</td>

<td>


  ### json

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="json"></code-example>


  The Angular [`json`](api/common/JsonPipe) pipe does the same thing.
</td>
```

```
<td>


  ### limitTo

  <code-example hideCopy>
    &lt;tr ng-repeat="movie in movieList | limitTo:2:0">
  </code-example>


  Selects up to the first parameter (2) number of items from the collection
  starting (optionally) at the beginning index (0).
</td>

<td>


  ### slice

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="slice"></code-example>


  The `SlicePipe` does the same thing but the *order of the parameters is reversed*,
in keeping
  with the JavaScript `Slice` method.
```

```
  The first parameter is the starting index; the second is the limit.
  As in AngularJS, coding this operation within the component instead could improve
performance.
</td>
```

```
<td>


  ### lowercase

  <code-example hideCopy>
    &lt;td>{{movie.title | lowercase}}&lt;/td>
  </code-example>


  Converts the string to lowercase.
</td>

<td>


  ### lowercase

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="lowercase"></code-example>


  The Angular `lowercase` pipe does the same thing.
</td>
```

```
<td>


  ### number

  <code-example hideCopy>
    &lt;td>{{movie.starRating | number}}&lt;/td>
  </code-example>


  Formats a number as text.
</td>

<td>


  ### number

  <code-example hideCopy path="ajs-quick-reference/src/app/app.component.html"
region="number"></code-example>
```

```
  The Angular [`number`](api/common/DecimalPipe) pipe is similar.
  It provides more functionality when defining
  the decimal places, as shown in the second example above.

  Angular also has a `percent` pipe, which formats a number as a local percentage
  as shown in the third example.
</td>
```

```
<td>


  ### orderBy

  <code-example hideCopy>
    &lt;tr ng-repeat="movie in movieList | orderBy : 'title'">
  </code-example>


  Displays the collection in the order specified by the expression.
  In this example, the movie title orders the `movieList`.
</td>

<td>


  ### none
  For performance reasons, no comparable pipe exists in Angular.
  Instead, use component code to order or sort results. If you need the same ordering
or sorting code in several templates, consider building a custom pipe.

</td>
```

{@a controllers-components}

Modules/controllers/components

In both AngularJS and Angular, modules help you organize your application into cohesive blocks of functionality.

In AngularJS, you write the code that provides the model and the methods for the view in a **controller**. In Angular, you build a **component**.

Because much AngularJS code is in JavaScript, JavaScript code is shown in the AngularJS column. The Angular code is shown using TypeScript.

```
<th>
  AngularJS
</th>

<th>
```

```
  Angular
</th>
```

```
<td>


  ### IIFE

  <code-example hideCopy>
    (function () {
      ...
    }());
  </code-example>


  In AngularJS, an immediately invoked function expression (or IIFE) around controller
code
  keeps it out of the global namespace.

</td>

<td>


  ### none
  This is a nonissue in Angular because ES 2015 modules
  handle the namespacing for you.

  For more information on modules, see the [Modules](guide/architecture#modules)
section of the
  [Architecture Overview](guide/architecture).
</td>
```

```
<td>


  ### Angular modules

  <code-example hideCopy>
    angular.module("movieHunter", ["ngRoute"]);
  </code-example>


  In AngularJS, an Angular module keeps track of controllers, services, and other
code.
  The second argument defines the list of other modules that this module depends upon.
</td>

<td>
```

```
  ### NgModules

  <code-example hideCopy path="ajs-quick-reference/src/app/app.module.1.ts"></code-
example>


  NgModules, defined with the `NgModule` decorator, serve the same purpose:

  * `imports`: specifies the list of other modules that this module depends upon
  * `declaration`: keeps track of your components, pipes, and directives.

  For more information on modules, see [NgModules](guide/ngmodules).
</td>
```

```
<td>


  ### Controller registration

  <code-example hideCopy>
    angular
      .module("movieHunter")
      .controller("MovieListCtrl",
                  ["movieService",
                   MovieListCtrl]);
  </code-example>


  AngularJS has code in each controller that looks up an appropriate Angular module
  and registers the controller with that module.

  The first argument is the controller name. The second argument defines the string
names of
  all dependencies injected into this controller, and a reference to the controller
function.
</td>

<td>


  ### Component decorator

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.ts"
region="component"></code-example>


  Angular adds a decorator to the component class to provide any required metadata.
  The `@Component` decorator declares that the class is a component and provides
metadata about
  that component such as its selector (or tag) and its template.

  This is how you associate a template with logic, which is defined in the component
```

```
class.

  For more information, see the [Components](guide/architecture#components)
  section of the [Architecture Overview](guide/architecture) page.
</td>
```

```
<td>


  ### Controller function

  <code-example hideCopy>
    function MovieListCtrl(movieService) {
    }
  </code-example>


  In AngularJS, you write the code for the model and methods in a controller function.
</td>

<td>


  ### Component class

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.ts"
region="class"></code-example>


  In Angular, you create a component class to contain the data model and control
methods. Use the TypeScript <code>export</code> keyword to export the class so that
the functionality can be imported into NgModules.

  For more information, see the [Components](guide/architecture#components)
  section of the [Architecture Overview](guide/architecture) page.
</td>
```

```
<td>


  ### Dependency injection

  <code-example hideCopy>
    MovieListCtrl.$inject = ['MovieService'];
    function MovieListCtrl(movieService) {
    }
  </code-example>


  In AngularJS, you pass in any dependencies as controller function arguments.
  This example injects a `MovieService`.
```

```
  To guard against minification problems, tell Angular explicitly
  that it should inject an instance of the `MovieService` in the first parameter.
</td>

<td>


  ### Dependency injection

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.ts"
region="di"></code-example>


  In Angular, you pass in dependencies as arguments to the component class
constructor.
  This example injects a `MovieService`.
  The first parameter's TypeScript type tells Angular what to inject, even after
minification.

  For more information, see the [Dependency injection](guide/architecture#dependency-
injection)
  section of the [Architecture Overview](guide/architecture).
</td>
```

{@a style-sheets}

Style sheets

Style sheets give your application a nice look. In AngularJS, you specify the style sheets for your entire application. As the application grows over time, the styles for the many parts of the application merge, which can cause unexpected results. In Angular, you can still define style sheets for your entire application. But now you can also encapsulate a style sheet within a specific component.

```
<th>
  AngularJS
</th>

<th>
  Angular
</th>
```

```
<td>


  ### Link tag

  <code-example hideCopy>
    &lt;link href="styles.css" rel="stylesheet" />
  </code-example>
```

```
  AngularJS, uses a `link` tag in the head section of the `index.html` file
  to define the styles for the application.
</td>

<td>



  ### Styles configuration
  <code-example hideCopy path="ajs-quick-reference/.angular-cli.1.json"
region="styles"></code-example>

  With the Angular CLI, you can configure your global styles in the `angular.json`
file.
  You can rename the extension to `.scss` to use sass.

  ### StyleUrls
  In Angular, you can use the `styles` or `styleUrls` property of the `@Component`
metadata to define
  a style sheet for a particular component.

  <code-example hideCopy path="ajs-quick-reference/src/app/movie-list.component.ts"
region="style-url"></code-example>



  This allows you to set appropriate styles for individual components that won’t leak
into
  other parts of the application.
</td>
```