Testing Utility APIs

This page describes the most useful Angular testing features.

The Angular testing utilities include the `TestBed`, the `ComponentFixture`, and a handful of functions that control the test environment. The *TestBed* and *ComponentFixture* classes are covered separately.

Here's a summary of the stand-alone functions, in order of likely utility:

Function

Description

waitForAsync

```
  Runs the body of a test (`it`) or setup (`beforeEach`) function within a special _async te
  See [waitForAsync](guide/testing-components-scenarios#waitForAsync).

</td>
```

fakeAsync

```
  Runs the body of a test (`it`) within a special _fakeAsync test zone_, enabling
  a linear control flow coding style. See [fakeAsync](guide/testing-components-scenarios#fak

</td>
```

tick

```
  Simulates the passage of time and the completion of pending asynchronous activities
  by flushing both _timer_ and _micro-task_ queues within the _fakeAsync test zone_.

  <div class="alert is-helpful">

  The curious, dedicated reader might enjoy this lengthy blog post,
  ["_Tasks, microtasks, queues and schedules_"](https://jakearchibald.com/2015/tasks-microta

  </div>

  Accepts an optional argument that moves the virtual clock forward
  by the specified number of milliseconds,
  clearing asynchronous activities scheduled within that timeframe.
  See [tick](guide/testing-components-scenarios#tick).

</td>
```

inject

```
  Injects one or more services from the current `TestBed` injector into a test function.
  It cannot inject a service provided by the component itself.
  See discussion of the [debugElement.injector](guide/testing-components-scenarios#get-injec
```

</td>

discardPeriodicTasks

```
When a `fakeAsync()` test ends with pending timer event _tasks_ (queued `setTimeOut` and `
the test fails with a clear error message.

In general, a test should end with no queued tasks.
When pending timer tasks are expected, call `discardPeriodicTasks` to flush the _task_ que
and avoid the error.
```

</td>

flushMicrotasks

```
When a `fakeAsync()` test ends with pending _micro-tasks_ such as unresolved promises,
the test fails with a clear error message.

In general, a test should wait for micro-tasks to finish.
When pending microtasks are expected, call `flushMicrotasks` to flush the  _micro-task_ qu
and avoid the error.
```

</td>

ComponentFixtureAutoDetect

```
A provider token for a service that turns on [automatic change detection](guide/testing-co
```

</td>

getTestBed

```
Gets the current instance of the `TestBed`.
Usually unnecessary because the static class methods of the `TestBed` class are typically
The `TestBed` instance exposes a few rarely used members that are not available as
static methods.
```

</td>

{@a testbed-class-summary} ## *TestBed* class summary

The `TestBed` class is one of the principal Angular testing utilities. Its API is quite large and can be overwhelming until you've explored it, a little at a time. Read the early part of this guide first to get the basics before trying to absorb the full API.

The module definition passed to `configureTestingModule` is a subset of the `@NgModule` metadata properties.

type TestModuleMetadata = { providers?: any[]; declarations?: any[]; imports?: any[]; schemas?: Array<SchemaMetadata | any[]>; };

{@a metadata-override-object}

Each override method takes a `MetadataOverride<T>` where `T` is the kind of metadata appropriate to the method, that is, the parameter of an `@NgModule`, `@Component`, `@Directive`, or `@Pipe`.

type MetadataOverride<T> = { add?: Partial<T>; remove?: Partial<T>; set?: Partial<T>; };

{@a testbed-methods} {@a testbed-api-summary}

The `TestBed` API consists of static class methods that either update or reference a *global* instance of the `TestBed`.

Internally, all static methods cover methods of the current runtime `TestBed` instance, which is also returned by the `getTestBed()` function.

Call `TestBed` methods *within* a `beforeEach()` to ensure a fresh start before each individual test.

Here are the most important static methods, in order of likely utility.

Methods

Description

configureTestingModule

```
  The testing shims (`karma-test-shim`, `browser-test-shim`)
  establish the [initial test environment](guide/testing) and a default testing module.
  The default testing module is configured with basic declaratives and some Angular service

  Call `configureTestingModule` to refine the testing module configuration for a particular
  by adding and removing imports, declarations (of components, directives, and pipes), and p

</td>
```

compileComponents

```
  Compile the testing module asynchronously after you've finished configuring it.
  You **must** call this method if _any_ of the testing module components have a `templateUr
  or `styleUrls` because fetching component template and style files is necessarily asynchro
  See [compileComponents](guide/testing-components-scenarios#compile-components).

  After calling `compileComponents`, the `TestBed` configuration is frozen for the duration

</td>
```

createComponent

```
  Create an instance of a component of type `T` based on the current `TestBed` configuratio
  After calling `compileComponent`, the `TestBed` configuration is frozen for the duration o
```

</td>

overrideModule

```
Replace metadata for the given `NgModule`. Recall that modules can import other modules.
The `overrideModule` method can reach deeply into the current testing module to
modify one of these inner modules.
```

</td>

overrideComponent

```
Replace metadata for the given component class, which could be nested deeply
within an inner module.
```

</td>

overrideDirective

```
Replace metadata for the given directive class, which could be nested deeply
within an inner module.
```

</td>

overridePipe

```
Replace metadata for the given pipe class, which could be nested deeply
within an inner module.
```

</td>

{@a testbed-inject} inject

```
Retrieve a service from the current `TestBed` injector.

The `inject` function is often adequate for this purpose.
But `inject` throws an error if it can't provide the service.

What if the service is optional?

The `TestBed.inject()` method takes an optional second parameter,
the object to return if Angular can't find the provider
(`null` in this example):

<code-example path="testing/src/app/demo/demo.testbed.spec.ts" region="testbed-get-w-null'

After calling `TestBed.inject`, the `TestBed` configuration is frozen for the duration of
```

</td>

{@a testbed-initTestEnvironment} initTestEnvironment

```
  Initialize the testing environment for the entire test run.

  The testing shims (`karma-test-shim`, `browser-test-shim`) call it for you
  so there is rarely a reason for you to call it yourself.

  Call this method _exactly once_. To change
  this default in the middle of a test run, call `resetTestEnvironment` first.

  Specify the Angular compiler factory, a `PlatformRef`, and a default Angular testing modul
  Alternatives for non-browser platforms are available in the general form
  `@angular/platform-<platform_name>/testing/<platform_name>`.
```

`</td>`

resetTestEnvironment

```
  Reset the initial test environment, including the default testing module.
```

`</td>`

A few of the `TestBed` instance methods are not covered by static `TestBed` *class* methods. These are rarely needed.

{@a component-fixture-api-summary}

The *ComponentFixture*

The `TestBed.createComponent<T>` creates an instance of the component `T` and returns a strongly typed `ComponentFixture` for that component.

The `ComponentFixture` properties and methods provide access to the component, its DOM representation, and aspects of its Angular environment.

{@a component-fixture-properties}

ComponentFixture properties

Here are the most important properties for testers, in order of likely utility.

Properties

Description

componentInstance

```
  The instance of the component class created by `TestBed.createComponent`.
```

`</td>`

debugElement

```
  The `DebugElement` associated with the root element of the component.

  The `debugElement` provides insight into the component and its DOM element during test anc
  It's a critical property for testers. The most interesting members are covered [below](#de
```

</td>

nativeElement

```
  The native DOM element at the root of the component.
```

</td>

changeDetectorRef

```
  The `ChangeDetectorRef` for the component.

  The `ChangeDetectorRef` is most valuable when testing a
  component that has the `ChangeDetectionStrategy.OnPush` method
  or the component's change detection is under your programmatic control.
```

</td>

{@a component-fixture-methods}

ComponentFixture **methods**

The *fixture* methods cause Angular to perform certain tasks on the component tree. Call these method to trigger Angular behavior in response to simulated user action.

Here are the most useful methods for testers.

Methods

Description

detectChanges

```
  Trigger a change detection cycle for the component.

  Call it to initialize the component (it calls `ngOnInit`) and after your
  test code, change the component's data bound property values.
  Angular can't see that you've changed `personComponent.name` and won't update the `name`
  binding until you call `detectChanges`.

  Runs `checkNoChanges` afterwards to confirm that there are no circular updates unless
  called as `detectChanges(false)`;
```

</td>

autoDetectChanges

```
Set this to `true` when you want the fixture to detect changes automatically.

When autodetect is `true`, the test fixture calls `detectChanges` immediately
after creating the component. Then it listens for pertinent zone events
and calls `detectChanges` accordingly.
When your test code modifies component property values directly,
you probably still have to call `fixture.detectChanges` to trigger data binding updates.

The default is `false`. Testers who prefer fine control over test behavior
tend to keep it `false`.
```

</td>

checkNoChanges

```
Do a change detection run to make sure there are no pending changes.
Throws an exceptions if there are.
```
</td>

isStable

```
If the fixture is currently _stable_, returns `true`.
If there are async tasks that have not completed, returns `false`.
```

</td>

whenStable

```
Returns a promise that resolves when the fixture is stable.

To resume testing after completion of asynchronous activity or
asynchronous change detection, hook that promise.
See [whenStable](guide/testing-components-scenarios#when-stable).
```

</td>

destroy

```
Trigger component destruction.
```

</td>

{@a debug-element-details}

DebugElement The `DebugElement` provides crucial insights into the component's DOM representation.

From the test root component's `DebugElement` returned by `fixture.debugElement`, you can walk (and query) the fixture's entire element and component subtrees.

Here are the most useful `DebugElement` members for testers, in approximate order of utility:

Member

Description

nativeElement

```
  The corresponding DOM element in the browser (null for WebWorkers).

</td>
```

query

```
  Calling `query(predicate: Predicate<DebugElement>)` returns the first `DebugElement`
  that matches the [predicate](#query-predicate) at any depth in the subtree.

</td>
```

queryAll

```
  Calling `queryAll(predicate: Predicate<DebugElement>)` returns all `DebugElements`
  that matches the [predicate](#query-predicate) at any depth in subtree.

</td>
```

injector

```
  The host dependency injector.
  For example, the root element's component instance injector.

</td>
```

componentInstance

```
  The element's own component instance, if it has one.

</td>
```

context

```
  An object that provides parent context for this element.
  Often an ancestor component instance that governs this element.

  When an element is repeated within `*ngFor`, the context is an `NgForOf` whose `$implicit
  property is the value of the row instance value.
  For example, the `hero` in `*ngFor="let hero of heroes"`.

</td>
```

children

```
  The immediate `DebugElement` children. Walk the tree by descending through `children`.

  <div class="alert is-helpful">

  `DebugElement` also has `childNodes`, a list of `DebugNode` objects.
  `DebugElement` derives from `DebugNode` objects and there are often
  more nodes than elements. Testers can usually ignore plain nodes.

  </div>
</td>
```

parent

```
  The `DebugElement` parent. Null if this is the root element.

</td>
```

name

```
  The element tag name, if it is an element.

</td>
```

triggerEventHandler

```
  Triggers the event by its name if there is a corresponding listener
  in the element's `listeners` collection.
  The second parameter is the _event object_ expected by the handler.
  See [triggerEventHandler](guide/testing-components-scenarios#trigger-event-handler).

  If the event lacks a listener or there's some other problem,
  consider calling `nativeElement.dispatchEvent(eventObject)`.

</td>
```

listeners

```
  The callbacks attached to the component's `@Output` properties and/or the element's event

</td>
```

providerTokens

```
  This component's injector lookup tokens.
  Includes the component itself plus the tokens that the component lists in its `providers`

</td>
```

source

```
  Where to find this element in the source component template.
```

</td>

references

```
Dictionary of objects associated with template local variables (e.g. `#foo`),
keyed by the local variable name.
```

</td>

{@a query-predicate}

The `DebugElement.query(predicate)` and `DebugElement.queryAll(predicate)` methods take a predicate that filters the source element's subtree for matching `DebugElement`.

The predicate is any method that takes a `DebugElement` and returns a *truthy* value. The following example finds all `DebugElements` with a reference to a template local variable named "content":

The Angular `By` class has three static methods for common predicates:

- `By.all` - return all elements.
- `By.css(selector)` - return elements with matching CSS selectors.
- `By.directive(directive)` - return elements that Angular matched to an instance of the directive class.