Deprecated APIs and features

Angular strives to balance innovation and stability. Sometimes, APIs and features become obsolete and need to be removed or replaced so that Angular can stay current with new best practices, changing dependencies, or changes in the (web) platform itself.

To make these transitions as easy as possible, we deprecate APIs and features for a period of time before removing them. This gives you time to update your applications to the latest APIs and best practices.

This guide contains a summary of all Angular APIs and features that are currently deprecated.

Features and APIs that were deprecated in v6 or earlier are candidates for removal in version 9 or any later major version. For information about Angular's deprecation and removal practices, see Angular Release Practices.

For step-by-step instructions on how to update to the latest Angular release, use the interactive update guide at update.angular.io.

Index

To help you future-proof your projects, the following table lists all deprecated APIs and features, organized by the release in which they are candidates for removal. Each item is linked to the section later in this guide that describes the deprecation reason and replacement options.

Area	API or Feature	May be removed in
`@angular/common`	`ReflectiveInjector`	v11
`@angular/common`	`CurrencyPipe - DEFAULT_CURRENCY_CODE`	v11
`@angular/common`	`NgComponentOutlet.ngComponentOutletNgModuleFactory`	v17
`@angular/common/http`	`XhrFactory`	v15
`@angular/common/http/testing`	`TestRequest` accepting `ErrorEvent` for error simulation	v16
`@angular/core`	`DefaultIterableDiffer`	v11
`@angular/core`	`ReflectiveKey`	v11
`@angular/core`	`RenderComponentType`	v11
`@angular/core`	Factory-based signature of `ApplicationRef.bootstrap`	v15
`@angular/core`	`PlatformRef.bootstrapModuleFactory`	v15
`@angular/core`	`getModuleFactory`	v16
`@angular/core`	`ModuleWithComponentFactories`	v16
`@angular/core`	`Compiler`	v16
`@angular/core`	`CompilerFactory`	v16
`@angular/core`	`NgModuleFactory`	v16
`@angular/core`	`ComponentFactory`	v16

`@angular/core`	`ComponentFactoryResolver`	v16
`@angular/core`	`CompilerOptions.useJit and CompilerOptions.missingTranslation config options`	v16
`@angular/platform-browser-dynamic`	`JitCompilerFactory`	v16
`@angular/platform-browser-dynamic`	`RESOURCE_CACHE_PROVIDER`	v16
`@angular/forms`	`ngModel` with reactive forms	v11
`@angular/upgrade`	`@angular/upgrade`	v11
`@angular/upgrade`	`getAngularLib`	v11
`@angular/upgrade`	`setAngularLib`	v11
`@angular/upgrade`	Factory-based signature of `downgradeModule`	v15
template syntax	`<template>`	v11
polyfills	reflect-metadata	v11
`@angular/compiler-cli`	Input setter coercion	v15
`@angular/compiler-cli`	`fullTemplateTypeCheck`	v15
`@angular/core`	`defineInjectable`	v11
`@angular/core`	`entryComponents`	v11
`@angular/core`	`ANALYZE_FOR_ENTRY_COMPONENTS`	v11
`@angular/core`	Factory-based signature of `ViewContainerRef.createComponent`	v15
`@angular/core/testing`	`TestBed.get`	v12
`@angular/core/testing`	`async`	v12
`@angular/core/testing`	`aotSummaries` argument in `TestBed.initTestEnvironment`	v14
`@angular/core/testing`	`aotSummaries` field of the `TestModuleMetadata` type	v14
`@angular/forms`	`FormBuilder.group` legacy options parameter	v14
`@angular/platform-server`	`renderModuleFactory`	v15
`@angular/service-worker`	`SwUpdate#activated`	v16
`@angular/service-worker`	`SwUpdate#available`	v16
template syntax	`/deep/`, `>>>`, and `::ng-deep`	unspecified
template syntax	`bind-`, `on-`, `bindon-`, and `ref-`	v15
`@angular/router`	`relativeLinkResolution`	v16

For information about Angular CDK and Angular Material deprecations, see the changelog.

Deprecated APIs

This section contains a complete list all of the currently-deprecated APIs, with details to help you plan your migration to a replacement.

TIP: In the API reference section of this site, deprecated APIs are indicated by ~~strikethrough.~~ You can filter the API list by **Status: deprecated**.

{@a common}

@angular/common

API	Replacement	De an
`CurrencyPipe - DEFAULT_CURRENCY_CODE`	`{provide: DEFAULT_CURRENCY_CODE, useValue: 'USD'}`	v9
`NgComponentOutlet.ngComponentOutletNgModuleFactory`	`NgComponentOutlet.ngComponentOutletNgModule`	v1

{@a common-http}

@angular/common/http

API	Replacement	Deprecation announced	Notes
`XhrFactory`	`XhrFactory` in `@angular/common`	v12	The `XhrFactory` has moved from `@angular/common/http` to `@angular/common`.

{@a core}

@angular/core

API	Replacement	Deprecation announced	Notes
`DefaultIterableDiffer`	n/a	v4	Not part (
`ReflectiveInjector`	`{@link Injector#create Injector.create()}`	v5	See `Refle`
`ReflectiveKey`	none	v5	none
`defineInjectable`	`ɵɵdefineInjectable`	v8	Used only No sourc depend c

entryComponents	none	v9	See entry
ANALYZE_FOR_ENTRY_COMPONENTS	none	v9	See ANALYZE_
async	waitForAsync	v11	The async @angular/ been rena in order t with the r async syn function i be remov version.
getModuleFactory	getNgModuleById	v13	Ivy allows NgModul without r correspor
`ViewChildren.emitDistinctChangesOnly` / `ContentChildren.emitDistinctChangesOnly`	none (was part of issue #40091)		This is a t introduce issue #40 removed.
Factory-based signature of ApplicationRef.bootstrap	Type-based signature of ApplicationRef.bootstrap	v13	With Ivy, resolve C and Comp provided
PlatformRef.bootstrapModuleFactory	PlatformRef.bootstrapModule	v13	With Ivy, resolve N NgModul provided
ModuleWithComponentFactories	none	v13	Ivy JIT mc accessing API chang ViewEngi additiona
Compiler	none	v13	Ivy JIT mc accessing API chang ViewEngi additiona
CompilerFactory	none	v13	Ivy JIT mc accessing API chang ViewEngi additiona

NgModuleFactory	Use non-factory based framework APIs like PlatformRef.bootstrapModule and createNgModuleRef	v13	Ivy JIT mc accessing API chang ViewEngi additiona
Factory-based signature of ViewContainerRef.createComponent	Type-based signature of ViewContainerRef.createComponent	v13	Angular r compone dynamica compone signature createCor which allc Compone
ComponentFactory	Use non-factory based framework APIs.	v13	Since Ivy, are not re provides Compone used dire
ComponentFactoryResolver	Use non-factory based framework APIs.	v13	Since Ivy, are not re no need t
CompilerOptions.useJit and CompilerOptions.missingTranslation config options	none	v13	Since Ivy, are unuse no effect.

{@a testing}

@angular/core/testing

API	Replacement	Deprecation announced	Notes
TestBed.get	TestBed.inject	v9	Same behavior, but type safe.
async	waitForAsync	v10	Same behavior, but rename to avoid confusion.
aotSummaries argument in TestBed.initTestEnvironment	No replacement needed	v13	Summary files are unused in Ivy.
aotSummaries field of the TestModuleMetadata type	No replacement needed	v13	Summary files are unused in Ivy.

{@a platform-browser-dynamic}

@angular/platform-browser-dynamic

API	Replacement	Deprecation	Notes

		announced	
JitCompilerFactory	none	v13	This symbol is no longer necessary. See JIT API changes due to ViewEngine deprecation for additional context.
RESOURCE_CACHE_PROVIDER	none	v13	This was previously necessary in some cases to test AOT-compiled components with View Engine, but is no longer since Ivy.

{@a platform-server}

@angular/platform-server

API	Replacement	Deprecation announced	Notes
renderModuleFactory	renderModule	v13	This symbol is no longer necessary. See JIT API changes due to ViewEngine deprecation for additional context.

{@a forms}

@angular/forms

API	Replacement	Deprecation announced	Notes
ngModel with reactive forms	FormControlDirective	v6	none
FormBuilder.group legacy options parameter	AbstractControlOptions parameter value	v11	none

{@a service-worker}

@angular/service-worker

API	Replacement	Deprecation announced	Notes
SwUpdate#activated	SwUpdate#activateUpdate() return value	v13	The return value of SwUpdate#activateUpdate() indicates whether an update was successfully activated.
SwUpdate#available	SwUpdate#versionUpdates	v13	The behavior of SwUpdate#available can be rebuilt by filtering for VersionReadyEvent events on SwUpdate#versionUpdates

{@a upgrade}

@angular/upgrade

API	Replacement	Deprecation announced	Notes
All entry points	`@angular/upgrade/static`	v5	See Upgrading from AngularJS.

{@a upgrade-static}

@angular/upgrade/static

API	Replacement	Deprecation announced	Notes
`getAngularLib`	`getAngularJSGlobal`	v5	See Upgrading from AngularJS.
`setAngularLib`	`setAngularJSGlobal`	v5	See Upgrading from AngularJS.
Factory-based signature of `downgradeModule`	NgModule-based signature of `downgradeModule`	v13	The `downgradeModule` supports more ergonomic NgModule-based API (vs NgModule factory based API).

{@a deprecated-features}

Deprecated features

This section lists all of the currently-deprecated features, which includes template syntax, configuration options, and any other deprecations not listed in the Deprecated APIs section above. It also includes deprecated API usage scenarios or API combinations, to augment the information above.

{@a bazelbuilder}

Bazel builder and schematics

Bazel builder and schematics were introduced in Angular Labs to let users try out Bazel without having to manage Bazel version and BUILD files. This feature has been deprecated. For more information, please refer to the migration doc.

{@a wtf}

Web Tracing Framework integration

Angular previously supported an integration with the Web Tracing Framework (WTF) for performance testing of Angular applications. This integration has not been maintained and is now defunct. As a result, the integration was deprecated in Angular version 8, and due to no evidence of any existing usage, removed in version 9.

{@a deep-component-style-selector}

`/deep/` , `>>>` , and `::ng-deep` component style selectors

The shadow-dom-piercing descendant combinator is deprecated and support is being removed from major browsers and tools. As such, in v4 we deprecated support in Angular for all three of `/deep/` , `>>>` , and `::ng-deep` . Until removal, `::ng-deep` is preferred for broader compatibility with the tools.

For more information, see /deep/, >>>, and ::ng-deep in the Component Styles guide.

{@a bind-syntax}

`bind-` , `on-` , `bindon-` , and `ref-` prefixes

The template prefixes `bind-` , `on-` , `bindon-` , and `ref-` have been deprecated in v13. Templates should use the more widely documented syntaxes for binding and references:

- `[input]="value"` instead of `bind-input="value"`
- `[@trigger]="value"` instead of `bind-animate-trigger="value"`
- `(click)="onClick()"` instead of `on-click="onClick()"`
- `[(ngModel)]="value"` instead of `bindon-ngModel="value"`
- `#templateRef` instead of `ref-templateRef`

{@a template-tag}

`<template>` tag

The `<template>` tag was deprecated in v4 to avoid colliding with the DOM's element of the same name (such as when using web components). Use `<ng-template>` instead. For more information, see the Ahead-of-Time Compilation guide.

{@a ngmodel-reactive}

ngModel with reactive forms

Support for using the `ngModel` input property and `ngModelChange` event with reactive form directives has been deprecated in Angular v6 and will be removed in a future version of Angular.

Now deprecated:

This support was deprecated for several reasons. First, developers found this pattern confusing. It seems like the actual `ngModel` directive is being used, but in fact it's an input/output property named `ngModel` on the reactive form directive that approximates some, but not all, of the directive's behavior. It allows getting and setting a value and intercepting value events, but some of `ngModel` 's other features, such as delaying updates with `ngModelOptions` or exporting the directive, don't work.

In addition, this pattern mixes template-driven and reactive forms strategies, which prevents taking advantage of the full benefits of either strategy. Setting the value in the template violates the template-agnostic principles behind reactive forms, whereas adding a `FormControl` / `FormGroup` layer in the class removes the convenience of defining forms in the template.

To update your code before support is removed, you'll want to decide whether to stick with reactive form directives (and get/set values using reactive forms patterns) or switch to template-driven directives.

After (choice 1 - use reactive forms):

After (choice 2 - use template-driven forms):

By default, when you use this pattern, you will see a deprecation warning once in dev mode. You can choose to silence this warning by configuring `ReactiveFormsModule` at import time:

Alternatively, you can choose to surface a separate warning for each instance of this pattern with a configuration value of `"always"` . This may help to track down where in the code the pattern is being used as the code is being updated.

{@a reflectiveinjector}

ReflectiveInjector

In v5, Angular replaced the `ReflectiveInjector` with the `StaticInjector`. The injector no longer requires the Reflect polyfill, reducing application size for most developers.

Before:

After:

{@a relativeLinkResolution}

The `relativeLinkResolution` option is being removed. The default was changed to the corrected behavior in version 11. Once this option is removed, the corrected behavior will always be used without the option to opt-in to the old, broken behavior.

{@a loadChildren}

loadChildren string syntax

When Angular first introduced lazy routes, there wasn't browser support for dynamically loading additional JavaScript. Angular created our own scheme using the syntax `loadChildren: './lazy/lazy.module#LazyModule'` and built tooling to support it. Now that ECMAScript dynamic import is supported in many browsers, Angular is moving toward this new syntax.

In version 8, the string syntax for the `loadChildren` route specification was deprecated, in favor of new syntax that uses `import()` syntax.

Before:

After:

Version 8 update: When you update to version 8, the `ng update` command performs the transformation automatically. Prior to version 7, the `import()` syntax only works in JIT mode (with view engine).

Declaration syntax: It's important to follow the route declaration syntax `loadChildren: () => import('...').then(m => m.ModuleName)` to allow `ngc` to discover the lazy-loaded module and the associated `NgModule`. You can find the complete list of allowed syntax constructs here. These restrictions will be relaxed with the release of Ivy since it'll no longer use `NgFactories`.

{@a reflect-metadata}

Dependency on a reflect-metadata polyfill in JIT mode

Angular applications, and specifically applications that relied on the JIT compiler, used to require a polyfill for the reflect-metadata APIs.

The need for this polyfill was removed in Angular version 8.0 (see #14473), rendering the presence of the poylfill in most Angular applications unnecessary. Because the polyfill can be depended on by 3rd-party libraries, instead of removing it from all Angular projects, we are deprecating the requirement for this polyfill as of version 8.0. This should give library authors and application developers sufficient time to evaluate if they need the polyfill, and perform any refactoring necessary to remove the dependency on it.

In a typical Angular project, the polyfill is not used in production builds, so removing it should not impact production applications. The goal behind this removal is overall simplification of the build setup and decrease in the number of external dependencies.

{@a static-query-resolution}

`@ViewChild()` / `@ContentChild()` static resolution as the default

See the dedicated migration guide for static queries.

{@a contentchild-input-together}

`@ContentChild()` / `@Input()` used together

The following pattern is deprecated:

Rather than using this pattern, separate the two decorators into their own properties and add fallback logic as in the following example:

{@a cant-assign-template-vars}

Cannot assign to template variables

In the following example, the two-way binding means that `optionName` should be written when the `valueChange` event fires.

However, in practice, Angular ignores two-way bindings to template variables. Starting in version 8, attempting to write to template variables is deprecated. In a future version, we will throw to indicate that the write is not supported.

{@a binding-to-innertext}

Binding to `innerText` in `platform-server`

Domino, which is used in server-side rendering, doesn't support `innerText`, so in platform-server's "domino adapter", there was special code to fall back to `textContent` if you tried to bind to `innerText`.

These two properties have subtle differences, so switching to `textContent` under the hood can be surprising to users. For this reason, we are deprecating this behavior. Going forward, users should explicitly bind to `textContent` when using Domino.

{@a wtf-apis}

`wtfStartTimeRange` and all `wtf*` APIs

All of the `wtf*` APIs are deprecated and will be removed in a future version.

{@a entryComponents}

`entryComponents` and `ANALYZE_FOR_ENTRY_COMPONENTS` no longer required

Previously, the `entryComponents` array in the `NgModule` definition was used to tell the compiler which components would be created and inserted dynamically. With Ivy, this isn't a requirement anymore and the `entryComponents` array can be removed from existing module declarations. The same applies to the `ANALYZE_FOR_ENTRY_COMPONENTS` injection token.

NOTE: You may still need to keep these if building a library that will be consumed by a View Engine application.

{@a moduleWithProviders}

`ModuleWithProviders` type without a generic

Some Angular libraries, such as `@angular/router` and `@ngrx/store`, implement APIs that return a type called `ModuleWithProviders` (typically using a method named `forRoot()`). This type represents an `NgModule` along with additional providers. Angular version 9 deprecates use of `ModuleWithProviders` without an explicitly generic type, where the generic type refers to the type of the `NgModule`. In a future version of Angular, the generic will no longer be optional.

If you're using the CLI, `ng update` should migrate your code automatically. If you're not using the CLI, you can add any missing generic types to your application manually. For example:

Before

After:

{@a input-setter-coercion}

Input setter coercion

Since the `strictTemplates` flag has been introduced in Angular the compiler has been able to type-check input bindings to the declared input type of the corresponding directive. When a getter/setter pair is being used for the input it may be desirable to let the setter accept a broader set of types than what is returned by the getter, for example when the setter first converts the input value. However, until TypeScript 4.3 a getter/setter pair was required to have identical types so this pattern could not be accurately declared.

To mitigate this limitation, it was made possible to declare input setter coercion fields in directives that are used when type-checking input bindings. However, since TypeScript 4.3 the limitation has been removed; setters can now accept a wider type than what is returned by the getter. This means that input coercion fields are no longer needed, as their effects can be achieved by widening the type of the setter.

For example, the following directive:

can be refactored as follows:

{@a full-template-type-check}

`fullTemplateTypeCheck`

When compiling your application using the AOT compiler, your templates are type-checked according to a certain strictness level. Before Angular 9 there existed only two strictness levels of template type checking as determined by the `fullTemplateTypeCheck` compiler option. In version 9 the `strictTemplates` family of compiler options has been introduced as a more fine-grained approach to configuring how strict your templates are being type-checked.

The `fullTemplateTypeCheck` flag is being deprecated in favor of the new `strictTemplates` option and its related compiler options. Projects that currently have `fullTemplateTypeCheck: true` configured can migrate to the following set of compiler options to achieve the same level of type-checking:

{ "angularCompilerOptions": { ... "strictTemplates": true, "strictInputTypes": false, "strictNullInputTypes": false, "strictAttributeTypes": false, "strictOutputEventTypes": false, "strictDomEventTypes": false, "strictDomLocalRefTypes": false, "strictSafeNavigationTypes": false, "strictContextGenerics": false, ... } }

{@a jit-api-changes}

JIT API changes due to ViewEngine deprecation

In ViewEngine, JIT compilation required special providers (like `Compiler`, `CompilerFactory`, etc) to be injected in the app and corresponding methods to be invoked. With Ivy, JIT compilation takes place implicitly if the Component, NgModule, etc have not already been AOT compiled. Those special providers were made available in Ivy for backwards-compatibility with ViewEngine to make the transition to Ivy smoother. Since ViewEngine is deprecated and will soon be removed, those symbols are now deprecated as well.

Important note: this deprecation doesn't affect JIT mode in Ivy (JIT remains available with Ivy, however we are exploring a possibility of deprecating it in the future. See RFC: Exploration of use-cases for Angular JIT compilation mode).

{@a testrequest-errorevent}

`TestRequest` accepting `ErrorEvent`

Angular provides utilities for testing `HttpClient`. The `TestRequest` class from `@angular/common/http/testing` mocks HTTP request objects for use with `HttpTestingController`.

`TestRequest` provides an API for simulating an HTTP response with an error. In earlier versions of Angular, this API accepted objects of type `ErrorEvent`, which does not match the type of error event that browsers return natively. If you use `ErrorEvent` with `TestRequest`, you should switch to `ProgressEvent`.

Here is an example using a `ProgressEvent`:

```
const mockError = new ProgressEvent('error');
const mockRequest = httpTestingController.expectOne(..);

mockRequest.error(mockError);
```

{@a deprecated-cli-flags}

Deprecated CLI APIs and Options

This section contains a complete list all of the currently deprecated CLI flags.

@angular/cli

API/Option	May be removed in	Notes
`--prod`	v14	Use `--configuration production` instead.
`ng update --all`	v14	No longer has an effect.

@angular-devkit/build-angular

API/Option	May be removed in	Notes
`deployUrl`	v15	Use `baseHref` option, `APP_BASE_HREF` DI token or a combination of both instead. For more information, see the deploy url.
`showCircularDependencies`	v14	The recommended method to detect circular dependencies in project code is to use either a lint rule or other external tooling.

Protractor builder	v14	Deprecate as part of the Protractor deprecation.

@angular-devkit/build-optimizer

The entire NPM package is deprecated. It has always been experimental (never hit `1.0.0`) and has been an internal package for the Angular CLI. All the relevant functionality has been moved to `@angular-devkit/build-angular`.

{@a removed}

Removed APIs

The following APIs have been removed starting with version 11.0.0*:

Package	API	Replacement	Notes
`@angular/router`	`preserveQueryParams`	`queryParamsHandling`	

* To see APIs removed in version 10, check out this guide on the version 10 docs site.

{@a style-sanitization}

Style Sanitization for `[style]` and `[style.prop]` bindings

Angular used to sanitize `[style]` and `[style.prop]` bindings to prevent malicious code from being inserted through `javascript:` expressions in CSS `url()` entries. However, most modern browsers no longer support the usage of these expressions, so sanitization was only maintained for the sake of IE 6 and 7. Given that Angular does not support either IE 6 or 7 and sanitization has a performance cost, we will no longer sanitize style bindings as of version 10 of Angular.

`loadChildren` string syntax in `@angular/router`

It is no longer possible to use the `loadChildren` string syntax to configure lazy routes. The string syntax has been replaced with dynamic import statements. The `DeprecatedLoadChildren` type was removed from `@angular/router`. Find more information about the replacement in the `LoadChildrenCallback` documentation.

The supporting classes `NgModuleFactoryLoader`, `SystemJsNgModuleLoader` and `SystemJsNgModuleLoaderConfig` classes were removed from `@angular/core`, as well as `SpyNgModuleFactoryLoader` from `@angular/router`.

`WrappedValue`

The purpose of `WrappedValue` was to allow the same object instance to be treated as different for the purposes of change detection. It was commonly used with the `async` pipe in the case where the `Observable` produces the same instance of the value.

Given that this use case is relatively rare and special handling impacted application performance, the `WrappedValue` API has been removed in Angular 13.

If you rely on the behavior that the same object instance should cause change detection, you have two options:

- Clone the resulting value so that it has a new identity.

- Explicitly call `ChangeDetectorRef.detectChanges()` to force the update.

@reviewed 2021-09-15