Static query migration guide

Important note for library authors: This migration is especially crucial for library authors to facilitate their users upgrading to version 9 when it becomes available.

In version 9, the default setting for `@ViewChild` and `@ContentChild` queries is changing in order to fix buggy and surprising behavior in queries (read more about that here).

In preparation for this change, in version 8, we are migrating all applications and libraries to explicitly specify the resolution strategy for `@ViewChild` and `@ContentChild` queries.

Specifically, this migration adds an explicit "static" flag that dictates when that query's results should be assigned. Adding this flag will ensure your code works the same way when upgrading to version 9.

Before:

```
// query results sometimes available in `ngOnInit`, sometimes in `ngAfterViewInit` (based or
@ViewChild('foo') foo: ElementRef;
```

After:

```
// query results available in ngOnInit
@ViewChild('foo', {static: true}) foo: ElementRef;

OR

// query results available in ngAfterViewInit
@ViewChild('foo', {static: false}) foo: ElementRef;
```

Starting with version 9, the `static` flag will default to false. At that time, any `{static: false}` flags can be safely removed, and we will have a schematic that will update your code for you.

Note: this flag only applies to `@ViewChild` and `@ContentChild` queries specifically, as `@ViewChildren` and `@ContentChildren` queries do not have a concept of static and dynamic (they are always resolved as if they are "dynamic").

FAQ

{@a what-to-do-with-todo} ### What should I do if I see a `/* TODO: add static flag */` comment printed by the schematic?

If you see this comment, it means that the schematic couldn't statically figure out the correct flag. In this case, you'll have to add the correct flag based on your application's behavior. For more information on how to choose, see the next question.

{@a how-do-i-choose} ### How do I choose which `static` flag value to use: `true` or `false`?

In the official API docs, we have always recommended retrieving query results in `ngAfterViewInit` for view queries and `ngAfterContentInit` for content queries. This is because by the time those lifecycle hooks run, change detection has completed for the relevant nodes and we can guarantee that we have collected all the possible query results.

Most applications will want to use `{static: false}` for the same reason. This setting will ensure query matches that are dependent on binding resolution (e.g. results inside `*ngIf`s or `*ngFor`s) will be found by the query.

There are rarer cases where `{static: true}` flag might be necessary (see answer here).

{@a should-i-use-static-true} ### Is there a case where I should use `{static: true}`?

This option was introduced to support creating embedded views on the fly. If you need access to a `TemplateRef` in a query to create a view dynamically, you won't be able to do so in `ngAfterViewInit`. Change detection has already run on that view, so creating a new view with the template will cause an `ExpressionHasChangedAfterChecked` error to be thrown. In this case, you will want to set the `static` flag to `true` and create your view in `ngOnInit`. In most other cases, the best practice is to use `{static: false}`.

However, to facilitate the migration to version 8, you may also want to set the `static` flag to `true` if your component code already depends on the query results being available some time **before** `ngAfterViewInit` (for view queries) or `ngAfterContentInit` (for content queries). For example, if your component relies on the query results being populated in the `ngOnInit` hook or in `@Input` setters, you will need to either set the flag to `true` or re-work your component to adjust to later timing.

Note: Selecting the static option means that query results nested in `*ngIf` or `*ngFor` will not be found by the query. These results are only retrievable after change detection runs.

{@a what-does-this-flag-mean} ### What does this flag mean and why is it necessary?

The default behavior for queries has historically been undocumented and confusing, and has also commonly led to issues that are difficult to debug. In version 9, we would like to make query behavior more consistent and simple to understand.

To explain why, first it's important to understand how queries have worked up until now.

Without the `static` flag, the compiler decided when each query would be resolved on a case-by-case basis. All `@ViewChild`/`@ContentChild` queries were

categorized into one of two buckets at compile time: “static” or “dynamic”. This classification determined when query results would become available to users.

- **Static queries** were queries where the result could be determined statically because the result didn’t depend on runtime values like bindings. Results from queries classified as static were available before change detection ran for that view (accessible in `ngOnInit`).
- **Dynamic queries** were queries where the result could NOT be determined statically because the result depended on runtime values (aka bindings). Results from queries classified as dynamic were not available until after change detection ran for that view (accessible in `ngAfterContentInit` for content queries or `ngAfterViewInit` for view queries).

For example, let’s say we have a component, `Comp`. Inside it, we have this query:

```
@ViewChild(Foo) foo: Foo;
```

and this template:

```
<div foo></div>
```

This `Foo` query would be categorized as static because at compile-time it’s known that the `Foo` instance on the `<div>` is the correct result for the query. Because the query result is not dependent on runtime values, we don’t have to wait for change detection to run on the template before resolving the query. Consequently, results can be made available in `ngOnInit`.

Let’s say the query is the same, but the component template looks like this:

```
<div foo *ngIf="showing"></div>
```

With that template, the query would be categorized as a dynamic query. We would need to know the runtime value of `showing` before determining what the correct results are for the query. As a result, change detection must run first, and results can only be made available in `ngAfterViewInit` or a setter for the query property.

The effect of this implementation is that adding an `*ngIf` or `*ngFor` anywhere above a query match can change when that query’s results become available.

Keep in mind that these categories only applied to `@ViewChild` and `@ContentChild` queries specifically. `@ViewChildren` and `@ContentChildren` queries did not have a concept of static and dynamic, so they were always resolved as if they were “dynamic”.

This strategy of resolving queries at different times based on the location of potential query matches has caused a lot of confusion. Namely:

- Sometimes query results are available in `ngOnInit`, but sometimes they aren’t and it’s not clear why (see 21800 or 19872).

- `@ViewChild` queries are resolved at a different time from `@ViewChildren` queries, and `@ContentChild` queries are resolved at a different time from `@ContentChildren` queries. If a user turns a `@ViewChild` query into a `@ViewChildren` query, their code can break suddenly because the timing has shifted.
- Code depending on a query result can suddenly stop working as soon as an `*ngIf` or an `*ngFor` is added to a template.
- A `@ContentChild` query for the same component will resolve at different times in the lifecycle for each usage of the component. This leads to buggy behavior where using a component with `*ngIf` is broken in subtle ways that aren't obvious to the component author.

In version 9, we plan to simplify the behavior so all queries resolve after change detection runs by default. The location of query matches in the template cannot affect when the query result will become available and suddenly break your code, and the default behavior is always the same. This makes the logic more consistent and predictable for users.

That said, if an application does need query results earlier (for example, the query result is needed to create an embedded view), it's possible to add the `{static: true}` flag to explicitly ask for static resolution. With this flag, users can indicate that they only care about results that are statically available and the query results will be populated before `ngOnInit`.

{@a view-children-and-content-children} ### Does this change affect `@ViewChildren` or `@ContentChildren` queries?

No, this change only affects `@ViewChild` and `@ContentChild` queries specifically. `@ViewChildren` and `@ContentChildren` queries are already "dynamic" by default and don't support static resolution.

{@a why-specify-static-false} ### Why do I have to specify `{static: false}`? Isn't that the default?

The goal of this migration is to transition apps that aren't yet on version 9 to a query pattern that is compatible with version 9. However, most applications use libraries, and it's likely that some of these libraries may not be upgraded to version 8 yet (and thus might not have the proper flags). Since the application's version of Angular will be used for compilation, if we change the default, the behavior of queries in the library's components will change to the version 8 default and possibly break. This way, an application's dependencies will behave the same way during the transition as they did in the previous version.

In Angular version 9 and later, it will be safe to remove any `{static: false}` flags and we will do this cleanup for you in a schematic.

{@a libraries} ### Can I keep on using Angular libraries that haven't yet updated to version 8 yet?

Yes, absolutely! Because we have not changed the default query behavior in version 8 (i.e. the compiler still chooses a timing if no flag is set), when your application runs with a library that has not updated to version 8, the library will run the same way it did in version 7. This guarantees your app will work in version 8 even if libraries take longer to update their code.

{@a update-library-to-use-static-flag} ### Can I update my library to version 8 by adding the `static` flag to view queries, while still being compatible with Angular version 7 apps?

Yes, the Angular team's recommendation for libraries is to update to version 8 and add the `static` flag. Angular version 7 apps will continue to work with libraries that have this flag.

However, if you update your library to Angular version 8 and want to take advantage of the new version 8 APIs, or you want more recent dependencies (such as Typescript or RxJS) your library will become incompatible with Angular version 7 apps. If your goal is to make your library compatible with Angular versions 7 and 8, you should not update your lib at all—except for `peerDependencies` in `package.json`.

In general, the most efficient plan is for libraries to adopt a 6 month major version schedule and bump the major version after each Angular update. That way, libraries stay in the same release cadence as Angular.