API Report File for "@angular/core"

> *Do not edit this file. It is a report generated by [API Extractor](#).*

```
import { Observable } from 'rxjs';
import { Subject } from 'rxjs';
import { Subscription } from 'rxjs';

// @public
export interface AbstractType<T> extends Function {
    // (undocumented)
    prototype: T;
}

// @public
export interface AfterContentChecked {
    ngAfterContentChecked(): void;
}

// @public
export interface AfterContentInit {
    ngAfterContentInit(): void;
}

// @public
export interface AfterViewChecked {
    ngAfterViewChecked(): void;
}

// @public
export interface AfterViewInit {
    ngAfterViewInit(): void;
}

// @public @deprecated
export const ANALYZE_FOR_ENTRY_COMPONENTS: InjectionToken<any>;

// @public
export const ANIMATION_MODULE_TYPE: InjectionToken<"NoopAnimations" |
"BrowserAnimations">;

// @public
export const APP_BOOTSTRAP_LISTENER: InjectionToken<((compRef: ComponentRef<any>) =>
void)[]>;

// @public
export const APP_ID: InjectionToken<string>;

// @public
export const APP_INITIALIZER: InjectionToken<readonly (() => Observable<unknown> |
```

```
Promise<unknown> | void)[]>;

// @public
export class ApplicationInitStatus {
    constructor(appInits: ReadonlyArray<() => Observable<unknown> | Promise<unknown>
| void>);
    // (undocumented)
    readonly done = false;
    // (undocumented)
    readonly donePromise: Promise<any>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<ApplicationInitStatus, [{ optional: true;
}]>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<ApplicationInitStatus>;
}

// @public
export class ApplicationModule {
    constructor(appRef: ApplicationRef);
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<ApplicationModule, never>;
    // (undocumented)
    static ɵinj: i0.ɵɵInjectorDeclaration<ApplicationModule>;
    // (undocumented)
    static ɵmod: i0.ɵɵNgModuleDeclaration<ApplicationModule, never, never, never>;
}

// @public
export class ApplicationRef {
    attachView(viewRef: ViewRef): void;
    bootstrap<C>(component: Type<C>, rootSelectorOrNode?: string | any):
ComponentRef<C>;
    // @deprecated
    bootstrap<C>(componentFactory: ComponentFactory<C>, rootSelectorOrNode?: string
| any): ComponentRef<C>;
    readonly components: ComponentRef<any>[];
    readonly componentTypes: Type<any>[];
    detachView(viewRef: ViewRef): void;
    readonly isStable: Observable<boolean>;
    tick(): void;
    get viewCount(): number;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<ApplicationRef, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<ApplicationRef>;
}

// @public (undocumented)
export function asNativeElements(debugEls: DebugElement[]): any;

// @public
```

```
export function assertPlatform(requiredToken: any): PlatformRef;

// @public
export interface Attribute {
    attributeName: string;
}

// @public
export const Attribute: AttributeDecorator;

// @public
export interface AttributeDecorator {
    (name: string): any;
    // (undocumented)
    new (name: string): Attribute;
}

// @public
export enum ChangeDetectionStrategy {
    Default = 1,
    OnPush = 0
}

// @public
export abstract class ChangeDetectorRef {
    abstract checkNoChanges(): void;
    abstract detach(): void;
    abstract detectChanges(): void;
    abstract markForCheck(): void;
    abstract reattach(): void;
}

// @public
export interface ClassProvider extends ClassSansProvider {
    multi?: boolean;
    provide: any;
}

// @public
export interface ClassSansProvider {
    useClass: Type<any>;
}

// @public @deprecated
export class Compiler {
    clearCache(): void;
    clearCacheFor(type: Type<any>): void;
    compileModuleAndAllComponentsAsync<T>(moduleType: Type<T>):
Promise<ModuleWithComponentFactories<T>>;
    compileModuleAndAllComponentsSync<T>(moduleType: Type<T>):
ModuleWithComponentFactories<T>;
    compileModuleAsync<T>(moduleType: Type<T>): Promise<NgModuleFactory<T>>;
```

```
    compileModuleSync<T>(moduleType: Type<T>): NgModuleFactory<T>;
    getModuleId(moduleType: Type<any>): string | undefined;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<Compiler, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<Compiler>;
}

// @public
export const COMPILER_OPTIONS: InjectionToken<CompilerOptions[]>;

// @public @deprecated
export abstract class CompilerFactory {
    // (undocumented)
    abstract createCompiler(options?: CompilerOptions[]): Compiler;
}

// @public
export type CompilerOptions = {
    useJit?: boolean;
    defaultEncapsulation?: ViewEncapsulation;
    providers?: StaticProvider[];
    missingTranslation?: MissingTranslationStrategy;
    preserveWhitespaces?: boolean;
};

// @public
export interface Component extends Directive {
    animations?: any[];
    changeDetection?: ChangeDetectionStrategy;
    encapsulation?: ViewEncapsulation;
    // @deprecated
    entryComponents?: Array<Type<any> | any[]>;
    interpolation?: [string, string];
    moduleId?: string;
    preserveWhitespaces?: boolean;
    styles?: string[];
    styleUrls?: string[];
    template?: string;
    templateUrl?: string;
    viewProviders?: Provider[];
}

// @public
export const Component: ComponentDecorator;

// @public
export interface ComponentDecorator {
    (obj: Component): TypeDecorator;
    new (obj: Component): Component;
}
```

```
// @public @deprecated
export abstract class ComponentFactory<C> {
    abstract get componentType(): Type<any>;
    abstract create(injector: Injector, projectableNodes?: any[][],
rootSelectorOrNode?: string | any, ngModule?: NgModuleRef<any>): ComponentRef<C>;
    abstract get inputs(): {
        propName: string;
        templateName: string;
    }[];
    abstract get ngContentSelectors(): string[];
    abstract get outputs(): {
        propName: string;
        templateName: string;
    }[];
    abstract get selector(): string;
}

// @public @deprecated
export abstract class ComponentFactoryResolver {
    // (undocumented)
    static NULL: ComponentFactoryResolver;
    abstract resolveComponentFactory<T>(component: Type<T>): ComponentFactory<T>;
}

// @public
export abstract class ComponentRef<C> {
    abstract get changeDetectorRef(): ChangeDetectorRef;
    abstract get componentType(): Type<any>;
    abstract destroy(): void;
    abstract get hostView(): ViewRef;
    abstract get injector(): Injector;
    abstract get instance(): C;
    abstract get location(): ElementRef;
    abstract onDestroy(callback: Function): void;
}

// @public
export interface ConstructorProvider extends ConstructorSansProvider {
    multi?: boolean;
    provide: Type<any>;
}

// @public
export interface ConstructorSansProvider {
    deps?: any[];
}

// @public
export type ContentChild = Query;

// @public
export const ContentChild: ContentChildDecorator;
```

```
// @public
export interface ContentChildDecorator {
    (selector: ProviderToken<unknown> | Function | string, opts?: {
        read?: any;
        static?: boolean;
    }): any;
    // (undocumented)
    new (selector: ProviderToken<unknown> | Function | string, opts?: {
        read?: any;
        static?: boolean;
    }): ContentChild;
}

// @public
export type ContentChildren = Query;

// @public
export const ContentChildren: ContentChildrenDecorator;

// @public
export interface ContentChildrenDecorator {
    (selector: ProviderToken<unknown> | Function | string, opts?: {
        descendants?: boolean;
        emitDistinctChangesOnly?: boolean;
        read?: any;
    }): any;
    // (undocumented)
    new (selector: ProviderToken<unknown> | Function | string, opts?: {
        descendants?: boolean;
        emitDistinctChangesOnly?: boolean;
        read?: any;
    }): Query;
}

// @public
export function createNgModuleRef<T>(ngModule: Type<T>, parentInjector?: Injector):
NgModuleRef<T>;

// @public
export function createPlatform(injector: Injector): PlatformRef;

// @public
export function createPlatformFactory(parentPlatformFactory: ((extraProviders?:
StaticProvider[]) => PlatformRef) | null, name: string, providers?:
StaticProvider[]): (extraProviders?: StaticProvider[]) => PlatformRef;

// @public
export const CUSTOM_ELEMENTS_SCHEMA: SchemaMetadata;

// @public (undocumented)
export class DebugElement extends DebugNode {
```

```
    constructor(nativeNode: Element);
    get attributes(): {
        [key: string]: string | null;
    };
    get childNodes(): DebugNode[];
    get children(): DebugElement[];
    get classes(): {
        [key: string]: boolean;
    };
    get name(): string;
    get nativeElement(): any;
    get properties(): {
        [key: string]: any;
    };
    // (undocumented)
    query(predicate: Predicate<DebugElement>): DebugElement;
    // (undocumented)
    queryAll(predicate: Predicate<DebugElement>): DebugElement[];
    // (undocumented)
    queryAllNodes(predicate: Predicate<DebugNode>): DebugNode[];
    get styles(): {
        [key: string]: string | null;
    };
    triggerEventHandler(eventName: string, eventObj?: any): void;
}

// @public (undocumented)
export class DebugEventListener {
    constructor(name: string, callback: Function);
    // (undocumented)
    callback: Function;
    // (undocumented)
    name: string;
}

// @public (undocumented)
export class DebugNode {
    constructor(nativeNode: Node);
    get componentInstance(): any;
    get context(): any;
    get injector(): Injector;
    get listeners(): DebugEventListener[];
    readonly nativeNode: any;
    get parent(): DebugElement | null;
    get providerTokens(): any[];
    get references(): {
        [key: string]: any;
    };
}

// @public
export const DEFAULT_CURRENCY_CODE: InjectionToken<string>;
```

```
// @public @deprecated (undocumented)
export class DefaultIterableDiffer<V> implements IterableDiffer<V>,
IterableChanges<V> {
    constructor(trackByFn?: TrackByFunction<V>);
    // (undocumented)
    check(collection: NgIterable<V>): boolean;
    // (undocumented)
    readonly collection: V[] | Iterable<V> | null;
    // (undocumented)
    diff(collection: NgIterable<V> | null | undefined): DefaultIterableDiffer<V> |
null;
    // (undocumented)
    forEachAddedItem(fn: (record: IterableChangeRecord_<V>) => void): void;
    // (undocumented)
    forEachIdentityChange(fn: (record: IterableChangeRecord_<V>) => void): void;
    // (undocumented)
    forEachItem(fn: (record: IterableChangeRecord_<V>) => void): void;
    // (undocumented)
    forEachMovedItem(fn: (record: IterableChangeRecord_<V>) => void): void;
    // (undocumented)
    forEachOperation(fn: (item: IterableChangeRecord<V>, previousIndex: number |
null, currentIndex: number | null) => void): void;
    // (undocumented)
    forEachPreviousItem(fn: (record: IterableChangeRecord_<V>) => void): void;
    // (undocumented)
    forEachRemovedItem(fn: (record: IterableChangeRecord_<V>) => void): void;
    // (undocumented)
    get isDirty(): boolean;
    // (undocumented)
    readonly length: number;
    // (undocumented)
    onDestroy(): void;
}

// @public @deprecated (undocumented)
export const defineInjectable: typeof eedefineInjectable;

// @public
export function destroyPlatform(): void;

// @public
export interface Directive {
    exportAs?: string;
    host?: {
        [key: string]: string;
    };
    inputs?: string[];
    jit?: true;
    outputs?: string[];
    providers?: Provider[];
    queries?: {
```

```
        [key: string]: any;
    };
    selector?: string;
}

// @public
export const Directive: DirectiveDecorator;

// @public
export interface DirectiveDecorator {
    (obj?: Directive): TypeDecorator;
    new (obj?: Directive): Directive;
}

// @public
export interface DoBootstrap {
    // (undocumented)
    ngDoBootstrap(appRef: ApplicationRef): void;
}

// @public
export interface DoCheck {
    ngDoCheck(): void;
}

// @public
export class ElementRef<T = any> {
    constructor(nativeElement: T);
    nativeElement: T;
}

// @public
export abstract class EmbeddedViewRef<C> extends ViewRef {
    abstract context: C;
    abstract get rootNodes(): any[];
}

// @public
export function enableProdMode(): void;

// @public
export class ErrorHandler {
    // (undocumented)
    handleError(error: any): void;
}

// @public
export interface EventEmitter<T> extends Subject<T> {
    new (isAsync?: boolean): EventEmitter<T>;
    emit(value?: T): void;
    subscribe(next?: (value: T) => void, error?: (error: any) => void, complete?: ()
=> void): Subscription;
```

```
    subscribe(observerOrNext?: any, error?: any, complete?: any): Subscription;
}

// @public (undocumented)
export const EventEmitter: {
    new (isAsync?: boolean): EventEmitter<any>;
    new <T>(isAsync?: boolean): EventEmitter<T>;
    readonly prototype: EventEmitter<any>;
};

// @public
export interface ExistingProvider extends ExistingSansProvider {
    multi?: boolean;
    provide: any;
}

// @public
export interface ExistingSansProvider {
    useExisting: any;
}

// @public
export interface FactoryProvider extends FactorySansProvider {
    multi?: boolean;
    provide: any;
}

// @public
export interface FactorySansProvider {
    deps?: any[];
    useFactory: Function;
}

// @public
export function forwardRef(forwardRefFn: ForwardRefFn): Type<any>;

// @public
export interface ForwardRefFn {
    // (undocumented)
    (): any;
}

// @public (undocumented)
export function getDebugNode(nativeNode: any): DebugNode | null;

// @public @deprecated
export function getModuleFactory(id: string): NgModuleFactory<any>;

// @public
export function getNgModuleById<T>(id: string): Type<T>;

// @public
```

```
export function getPlatform(): PlatformRef | null;

// @public
export interface GetTestability {
    // (undocumented)
    addToWindow(registry: TestabilityRegistry): void;
    // (undocumented)
    findTestabilityInTree(registry: TestabilityRegistry, elem: any, findInAncestors:
boolean): Testability | null;
}

// @public
export interface Host {
}

// @public
export const Host: HostDecorator;

// @public
export interface HostBinding {
    hostPropertyName?: string;
}

// @public (undocumented)
export const HostBinding: HostBindingDecorator;

// @public
export interface HostBindingDecorator {
    (hostPropertyName?: string): any;
    // (undocumented)
    new (hostPropertyName?: string): any;
}

// @public
export interface HostDecorator {
    (): any;
    // (undocumented)
    new (): Host;
}

// @public
export interface HostListener {
    args?: string[];
    eventName?: string;
}

// @public
export const HostListener: HostListenerDecorator;

// @public
export interface HostListenerDecorator {
    (eventName: string, args?: string[]): any;
```

```
    // (undocumented)
    new (eventName: string, args?: string[]): any;
}

// @public
export interface Inject {
    token: any;
}

// @public
export const Inject: InjectDecorator;

// @public
export const inject: typeof ɵɵinject;

// @public
export interface Injectable {
    providedIn?: Type<any> | 'root' | 'platform' | 'any' | null;
}

// @public
export const Injectable: InjectableDecorator;

// @public
export interface InjectableDecorator {
    (): TypeDecorator;
    // (undocumented)
    (options?: {
        providedIn: Type<any> | 'root' | 'platform' | 'any' | null;
    } & InjectableProvider): TypeDecorator;
    // (undocumented)
    new (): Injectable;
    // (undocumented)
    new (options?: {
        providedIn: Type<any> | 'root' | 'platform' | 'any' | null;
    } & InjectableProvider): Injectable;
}

// @public
export type InjectableProvider = ValueSansProvider | ExistingSansProvider |
StaticClassSansProvider | ConstructorSansProvider | FactorySansProvider |
ClassSansProvider;

// @public
export interface InjectableType<T> extends Type<T> {
    ɵprov: unknown;
}

// @public
export interface InjectDecorator {
    (token: any): any;
    // (undocumented)
```

```
    new (token: any): Inject;
}

// @public
export enum InjectFlags {
    Default = 0,
    Host = 1,
    Optional = 8,
    Self = 2,
    SkipSelf = 4
}

// @public
export class InjectionToken<T> {
    constructor(_desc: string, options?: {
        providedIn?: Type<any> | 'root' | 'platform' | 'any' | null;
        factory: () => T;
    });
    // (undocumented)
    protected _desc: string;
    // (undocumented)
    toString(): string;
    // (undocumented)
    readonly ɵprov: unknown;
}

// @public
export const INJECTOR: InjectionToken<Injector>;

// @public
export abstract class Injector {
    // @deprecated (undocumented)
    static create(providers: StaticProvider[], parent?: Injector): Injector;
    static create(options: {
        providers: StaticProvider[];
        parent?: Injector;
        name?: string;
    }): Injector;
    abstract get<T>(token: ProviderToken<T>, notFoundValue?: T, flags?:
InjectFlags): T;
    // @deprecated (undocumented)
    abstract get(token: any, notFoundValue?: any): any;
    // (undocumented)
    static NULL: Injector;
    // (undocumented)
    static THROW_IF_NOT_FOUND: {};
    // (undocumented)
    static ɵprov: unknown;
}

// @public
export interface InjectorType<T> extends Type<T> {
```

```
    // (undocumented)
    ɵfac?: unknown;
    // (undocumented)
    ɵinj: unknown;
}

// @public
export interface Input {
    bindingPropertyName?: string;
}

// @public (undocumented)
export const Input: InputDecorator;

// @public (undocumented)
export interface InputDecorator {
    (bindingPropertyName?: string): any;
    // (undocumented)
    new (bindingPropertyName?: string): any;
}

// @public
export function isDevMode(): boolean;

// @public
export interface IterableChangeRecord<V> {
    readonly currentIndex: number | null;
    readonly item: V;
    readonly previousIndex: number | null;
    readonly trackById: any;
}

// @public
export interface IterableChanges<V> {
    forEachAddedItem(fn: (record: IterableChangeRecord<V>) => void): void;
    forEachIdentityChange(fn: (record: IterableChangeRecord<V>) => void): void;
    forEachItem(fn: (record: IterableChangeRecord<V>) => void): void;
    forEachMovedItem(fn: (record: IterableChangeRecord<V>) => void): void;
    forEachOperation(fn: (record: IterableChangeRecord<V>, previousIndex: number |
null, currentIndex: number | null) => void): void;
    forEachPreviousItem(fn: (record: IterableChangeRecord<V>) => void): void;
    forEachRemovedItem(fn: (record: IterableChangeRecord<V>) => void): void;
}

// @public
export interface IterableDiffer<V> {
    diff(object: NgIterable<V> | undefined | null): IterableChanges<V> | null;
}

// @public
export interface IterableDifferFactory {
    // (undocumented)
```

```
    create<V>(trackByFn?: TrackByFunction<V>): IterableDiffer<V>;
    // (undocumented)
    supports(objects: any): boolean;
}

// @public
export class IterableDiffers {
    constructor(factories: IterableDifferFactory[]);
    // (undocumented)
    static create(factories: IterableDifferFactory[], parent?: IterableDiffers):
IterableDiffers;
    static extend(factories: IterableDifferFactory[]): StaticProvider;
    // @deprecated (undocumented)
    factories: IterableDifferFactory[];
    // (undocumented)
    find(iterable: any): IterableDifferFactory;
    // (undocumented)
    static ɵprov: unknown;
}

// @public
export interface KeyValueChangeRecord<K, V> {
    readonly currentValue: V | null;
    readonly key: K;
    readonly previousValue: V | null;
}

// @public
export interface KeyValueChanges<K, V> {
    forEachAddedItem(fn: (r: KeyValueChangeRecord<K, V>) => void): void;
    forEachChangedItem(fn: (r: KeyValueChangeRecord<K, V>) => void): void;
    forEachItem(fn: (r: KeyValueChangeRecord<K, V>) => void): void;
    forEachPreviousItem(fn: (r: KeyValueChangeRecord<K, V>) => void): void;
    forEachRemovedItem(fn: (r: KeyValueChangeRecord<K, V>) => void): void;
}

// @public
export interface KeyValueDiffer<K, V> {
    diff(object: Map<K, V>): KeyValueChanges<K, V> | null;
    diff(object: {
        [key: string]: V;
    }): KeyValueChanges<string, V> | null;
}

// @public
export interface KeyValueDifferFactory {
    create<K, V>(): KeyValueDiffer<K, V>;
    supports(objects: any): boolean;
}

// @public
export class KeyValueDiffers {
```

```
    constructor(factories: KeyValueDifferFactory[]);
    // (undocumented)
    static create<S>(factories: KeyValueDifferFactory[], parent?: KeyValueDiffers):
KeyValueDiffers;
    static extend<S>(factories: KeyValueDifferFactory[]): StaticProvider;
    // @deprecated (undocumented)
    factories: KeyValueDifferFactory[];
    // (undocumented)
    find(kv: any): KeyValueDifferFactory;
    // (undocumented)
    static ɵprov: unknown;
}

// @public
export const LOCALE_ID: InjectionToken<string>;

// @public
export enum MissingTranslationStrategy {
    // (undocumented)
    Error = 0,
    // (undocumented)
    Ignore = 2,
    // (undocumented)
    Warning = 1
}

// @public @deprecated
export class ModuleWithComponentFactories<T> {
    constructor(ngModuleFactory: NgModuleFactory<T>, componentFactories:
ComponentFactory<any>[]);
    // (undocumented)
    componentFactories: ComponentFactory<any>[];
    // (undocumented)
    ngModuleFactory: NgModuleFactory<T>;
}

// @public
export interface ModuleWithProviders<T> {
    // (undocumented)
    ngModule: Type<T>;
    // (undocumented)
    providers?: Provider[];
}

// @public
export type NgIterable<T> = Array<T> | Iterable<T>;

// @public
export interface NgModule {
    bootstrap?: Array<Type<any> | any[]>;
    declarations?: Array<Type<any> | any[]>;
    // @deprecated
```

```
    entryComponents?: Array<Type<any> | any[]>;
    exports?: Array<Type<any> | any[]>;
    id?: string;
    imports?: Array<Type<any> | ModuleWithProviders<{}> | any[]>;
    jit?: true;
    providers?: Provider[];
    schemas?: Array<SchemaMetadata | any[]>;
}

// @public (undocumented)
export const NgModule: NgModuleDecorator;

// @public
export interface NgModuleDecorator {
    (obj?: NgModule): TypeDecorator;
    // (undocumented)
    new (obj?: NgModule): NgModule;
}

// @public @deprecated (undocumented)
export abstract class NgModuleFactory<T> {
    // (undocumented)
    abstract create(parentInjector: Injector | null): NgModuleRef<T>;
    // (undocumented)
    abstract get moduleType(): Type<T>;
}

// @public
export abstract class NgModuleRef<T> {
    // @deprecated
    abstract get componentFactoryResolver(): ComponentFactoryResolver;
    abstract destroy(): void;
    abstract get injector(): Injector;
    abstract get instance(): T;
    abstract onDestroy(callback: () => void): void;
}

// @public
export class NgProbeToken {
    constructor(name: string, token: any);
    // (undocumented)
    name: string;
    // (undocumented)
    token: any;
}

// @public
export class NgZone {
    constructor({ enableLongStackTrace, shouldCoalesceEventChangeDetection,
shouldCoalesceRunChangeDetection }: {
        enableLongStackTrace?: boolean | undefined;
        shouldCoalesceEventChangeDetection?: boolean | undefined;
```

```
        shouldCoalesceRunChangeDetection?: boolean | undefined;
    });
    // (undocumented)
    static assertInAngularZone(): void;
    // (undocumented)
    static assertNotInAngularZone(): void;
    // (undocumented)
    readonly hasPendingMacrotasks: boolean;
    // (undocumented)
    readonly hasPendingMicrotasks: boolean;
    // (undocumented)
    static isInAngularZone(): boolean;
    readonly isStable: boolean;
    readonly onError: EventEmitter<any>;
    readonly onMicrotaskEmpty: EventEmitter<any>;
    readonly onStable: EventEmitter<any>;
    readonly onUnstable: EventEmitter<any>;
    run<T>(fn: (...args: any[]) => T, applyThis?: any, applyArgs?: any[]): T;
    runGuarded<T>(fn: (...args: any[]) => T, applyThis?: any, applyArgs?: any[]): T;
    runOutsideAngular<T>(fn: (...args: any[]) => T): T;
    runTask<T>(fn: (...args: any[]) => T, applyThis?: any, applyArgs?: any[], name?:
string): T;
}

// @public
export const NO_ERRORS_SCHEMA: SchemaMetadata;

// @public
export interface OnChanges {
    ngOnChanges(changes: SimpleChanges): void;
}

// @public
export interface OnDestroy {
    ngOnDestroy(): void;
}

// @public
export interface OnInit {
    ngOnInit(): void;
}

// @public
export interface Optional {
}

// @public
export const Optional: OptionalDecorator;

// @public
export interface OptionalDecorator {
    (): any;
```

```
    // (undocumented)
    new (): Optional;
}

// @public
export interface Output {
    bindingPropertyName?: string;
}

// @public (undocumented)
export const Output: OutputDecorator;

// @public
export interface OutputDecorator {
    (bindingPropertyName?: string): any;
    // (undocumented)
    new (bindingPropertyName?: string): any;
}

// @public
export const PACKAGE_ROOT_URL: InjectionToken<string>;

// @public
export interface Pipe {
    name: string;
    pure?: boolean;
}

// @public (undocumented)
export const Pipe: PipeDecorator;

// @public
export interface PipeDecorator {
    (obj: Pipe): TypeDecorator;
    new (obj: Pipe): Pipe;
}

// @public
export interface PipeTransform {
    // (undocumented)
    transform(value: any, ...args: any[]): any;
}

// @public
export const PLATFORM_ID: InjectionToken<Object>;

// @public
export const PLATFORM_INITIALIZER: InjectionToken<(() => void)[]>;

// @public
export const platformCore: (extraProviders?: StaticProvider[] | undefined) =>
PlatformRef;
```

```
// @public
export class PlatformRef {
    bootstrapModule<M>(moduleType: Type<M>, compilerOptions?: (CompilerOptions &
BootstrapOptions) | Array<CompilerOptions & BootstrapOptions>):
Promise<NgModuleRef<M>>;
    // @deprecated
    bootstrapModuleFactory<M>(moduleFactory: NgModuleFactory<M>, options?:
BootstrapOptions): Promise<NgModuleRef<M>>;
    destroy(): void;
    // (undocumented)
    get destroyed(): boolean;
    get injector(): Injector;
    onDestroy(callback: () => void): void;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<PlatformRef, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<PlatformRef>;
}

// @public
export interface Predicate<T> {
    // (undocumented)
    (value: T): boolean;
}

// @public
export type Provider = TypeProvider | ValueProvider | ClassProvider |
ConstructorProvider | ExistingProvider | FactoryProvider | any[];

// @public
export type ProviderToken<T> = Type<T> | AbstractType<T> | InjectionToken<T>;

// @public
export interface Query {
    // (undocumented)
    descendants: boolean;
    // (undocumented)
    emitDistinctChangesOnly: boolean;
    // (undocumented)
    first: boolean;
    // (undocumented)
    isViewQuery: boolean;
    // (undocumented)
    read: any;
    // (undocumented)
    selector: any;
    // (undocumented)
    static?: boolean;
}

// @public
```

```
export abstract class Query {
}

// @public
export class QueryList<T> implements Iterable<T> {
    // (undocumented)
    [Symbol.iterator]: () => Iterator<T>;
    constructor(_emitDistinctChangesOnly?: boolean);
    get changes(): Observable<any>;
    destroy(): void;
    // (undocumented)
    readonly dirty = true;
    filter(fn: (item: T, index: number, array: T[]) => boolean): T[];
    find(fn: (item: T, index: number, array: T[]) => boolean): T | undefined;
    // (undocumented)
    readonly first: T;
    forEach(fn: (item: T, index: number, array: T[]) => void): void;
    get(index: number): T | undefined;
    // (undocumented)
    readonly last: T;
    // (undocumented)
    readonly length: number;
    map<U>(fn: (item: T, index: number, array: T[]) => U): U[];
    notifyOnChanges(): void;
    reduce<U>(fn: (prevValue: U, curValue: T, curIndex: number, array: T[]) => U,
init: U): U;
    reset(resultsTree: Array<T | any[]>, identityAccessor?: (value: T) => unknown):
void;
    setDirty(): void;
    some(fn: (value: T, index: number, array: T[]) => boolean): boolean;
    toArray(): T[];
    // (undocumented)
    toString(): string;
}

// @public @deprecated
export abstract class ReflectiveInjector implements Injector {
    abstract createChildFromResolved(providers: ResolvedReflectiveProvider[]):
ReflectiveInjector;
    static fromResolvedProviders(providers: ResolvedReflectiveProvider[], parent?:
Injector): ReflectiveInjector;
    // (undocumented)
    abstract get(token: any, notFoundValue?: any): any;
    abstract instantiateResolved(provider: ResolvedReflectiveProvider): any;
    abstract get parent(): Injector | null;
    static resolve(providers: Provider[]): ResolvedReflectiveProvider[];
    static resolveAndCreate(providers: Provider[], parent?: Injector):
ReflectiveInjector;
    abstract resolveAndCreateChild(providers: Provider[]): ReflectiveInjector;
    abstract resolveAndInstantiate(provider: Provider): any;
}
```

```
// @public @deprecated
export class ReflectiveKey {
    constructor(token: Object, id: number);
    // (undocumented)
    readonly displayName: string;
    static get(token: Object): ReflectiveKey;
    // (undocumented)
    id: number;
    // (undocumented)
    static get numberOfKeys(): number;
    // (undocumented)
    token: Object;
}

// @public
export abstract class Renderer2 {
    abstract addClass(el: any, name: string): void;
    abstract appendChild(parent: any, newChild: any): void;
    abstract createComment(value: string): any;
    abstract createElement(name: string, namespace?: string | null): any;
    abstract createText(value: string): any;
    abstract get data(): {
        [key: string]: any;
    };
    abstract destroy(): void;
    destroyNode: ((node: any) => void) | null;
    abstract insertBefore(parent: any, newChild: any, refChild: any, isMove?:
boolean): void;
    abstract listen(target: 'window' | 'document' | 'body' | any, eventName: string,
callback: (event: any) => boolean | void): () => void;
    abstract nextSibling(node: any): any;
    abstract parentNode(node: any): any;
    abstract removeAttribute(el: any, name: string, namespace?: string | null):
void;
    abstract removeChild(parent: any, oldChild: any, isHostElement?: boolean): void;
    abstract removeClass(el: any, name: string): void;
    abstract removeStyle(el: any, style: string, flags?: RendererStyleFlags2): void;
    abstract selectRootElement(selectorOrNode: string | any, preserveContent?:
boolean): any;
    abstract setAttribute(el: any, name: string, value: string, namespace?: string |
null): void;
    abstract setProperty(el: any, name: string, value: any): void;
    abstract setStyle(el: any, style: string, value: any, flags?:
RendererStyleFlags2): void;
    abstract setValue(node: any, value: string): void;
}

// @public
export abstract class RendererFactory2 {
    abstract begin?(): void;
    abstract createRenderer(hostElement: any, type: RendererType2 | null):
Renderer2;
```

```
    abstract end?(): void;
    abstract whenRenderingDone?(): Promise<any>;
}

// @public
export enum RendererStyleFlags2 {
    DashCase = 2,
    Important = 1
}

// @public
export interface RendererType2 {
    data: {
        [kind: string]: any;
    };
    encapsulation: ViewEncapsulation;
    id: string;
    styles: (string | any[])[];
}

// @public
export class ResolvedReflectiveFactory {
    constructor(
    factory: Function,
    dependencies: ReflectiveDependency[]);
    dependencies: ReflectiveDependency[];
    factory: Function;
}

// @public
export interface ResolvedReflectiveProvider {
    key: ReflectiveKey;
    multiProvider: boolean;
    resolvedFactories: ResolvedReflectiveFactory[];
}

// @public
export function resolveForwardRef<T>(type: T): T;

// @public
export abstract class Sanitizer {
    // (undocumented)
    abstract sanitize(context: SecurityContext, value: {} | string | null): string |
null;
    // (undocumented)
    static eprov: unknown;
}

// @public
export interface SchemaMetadata {
    // (undocumented)
    name: string;
```

```
}

// @public
export enum SecurityContext {
    // (undocumented)
    HTML = 1,
    // (undocumented)
    NONE = 0,
    // (undocumented)
    RESOURCE_URL = 5,
    // (undocumented)
    SCRIPT = 3,
    // (undocumented)
    STYLE = 2,
    // (undocumented)
    URL = 4
}

// @public
export interface Self {
}

// @public
export const Self: SelfDecorator;

// @public
export interface SelfDecorator {
    (): any;
    // (undocumented)
    new (): Self;
}

// @public
export function setTestabilityGetter(getter: GetTestability): void;

// @public
export class SimpleChange {
    constructor(previousValue: any, currentValue: any, firstChange: boolean);
    // (undocumented)
    currentValue: any;
    // (undocumented)
    firstChange: boolean;
    isFirstChange(): boolean;
    // (undocumented)
    previousValue: any;
}

// @public
export interface SimpleChanges {
    // (undocumented)
    [propName: string]: SimpleChange;
}
```

```
// @public
export interface SkipSelf {
}

// @public
export const SkipSelf: SkipSelfDecorator;

// @public
export interface SkipSelfDecorator {
    (): any;
    // (undocumented)
    new (): SkipSelf;
}

// @public
export interface StaticClassProvider extends StaticClassSansProvider {
    multi?: boolean;
    provide: any;
}

// @public
export interface StaticClassSansProvider {
    deps: any[];
    useClass: Type<any>;
}

// @public
export type StaticProvider = ValueProvider | ExistingProvider | StaticClassProvider
| ConstructorProvider | FactoryProvider | any[];

// @public
export abstract class TemplateRef<C> {
    abstract createEmbeddedView(context: C, injector?: Injector):
EmbeddedViewRef<C>;
    abstract readonly elementRef: ElementRef;
}

// @public
export class Testability implements PublicTestability {
    constructor(_ngZone: NgZone);
    // @deprecated
    decreasePendingRequestCount(): number;
    findProviders(using: any, provider: string, exactMatch: boolean): any[];
    // @deprecated
    getPendingRequestCount(): number;
    // @deprecated
    increasePendingRequestCount(): number;
    isStable(): boolean;
    whenStable(doneCb: Function, timeout?: number, updateCb?: Function): void;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<Testability, never>;
```

```
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<Testability>;
}

// @public
export class TestabilityRegistry {
    constructor();
    findTestabilityInTree(elem: Node, findInAncestors?: boolean): Testability |
null;
    getAllRootElements(): any[];
    getAllTestabilities(): Testability[];
    getTestability(elem: any): Testability | null;
    registerApplication(token: any, testability: Testability): void;
    unregisterAllApplications(): void;
    unregisterApplication(token: any): void;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<TestabilityRegistry, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<TestabilityRegistry>;
}

// @public
export interface TrackByFunction<T> {
    // (undocumented)
    <U extends T>(index: number, item: T & U): any;
}

// @public
export const TRANSLATIONS: InjectionToken<string>;

// @public
export const TRANSLATIONS_FORMAT: InjectionToken<string>;

// @public
export const Type: FunctionConstructor;

// @public (undocumented)
export interface Type<T> extends Function {
    // (undocumented)
    new (...args: any[]): T;
}

// @public
export interface TypeDecorator {
    <T extends Type<any>>(type: T): T;
    // (undocumented)
    (target: Object, propertyKey?: string | symbol, parameterIndex?: number): void;
}

// @public
export interface TypeProvider extends Type<any> {
}
```

```
// @public
export interface ValueProvider extends ValueSansProvider {
    multi?: boolean;
    provide: any;
}

// @public
export interface ValueSansProvider {
    useValue: any;
}

// @public (undocumented)
export const VERSION: Version;

// @public
export class Version {
    constructor(full: string);
    // (undocumented)
    full: string;
    // (undocumented)
    readonly major: string;
    // (undocumented)
    readonly minor: string;
    // (undocumented)
    readonly patch: string;
}

// @public
export type ViewChild = Query;

// @public
export const ViewChild: ViewChildDecorator;

// @public
export interface ViewChildDecorator {
    (selector: ProviderToken<unknown> | Function | string, opts?: {
        read?: any;
        static?: boolean;
    }): any;
    // (undocumented)
    new (selector: ProviderToken<unknown> | Function | string, opts?: {
        read?: any;
        static?: boolean;
    }): ViewChild;
}

// @public
export type ViewChildren = Query;

// @public
export const ViewChildren: ViewChildrenDecorator;
```

```
// @public
export interface ViewChildrenDecorator {
    (selector: ProviderToken<unknown> | Function | string, opts?: {
        read?: any;
        emitDistinctChangesOnly?: boolean;
    }): any;
    // (undocumented)
    new (selector: ProviderToken<unknown> | Function | string, opts?: {
        read?: any;
        emitDistinctChangesOnly?: boolean;
    }): ViewChildren;
}

// @public
export abstract class ViewContainerRef {
    abstract clear(): void;
    abstract createComponent<C>(componentType: Type<C>, options?: {
        index?: number;
        injector?: Injector;
        ngModuleRef?: NgModuleRef<unknown>;
        projectableNodes?: Node[][];
    }): ComponentRef<C>;
    // @deprecated
    abstract createComponent<C>(componentFactory: ComponentFactory<C>, index?:
number, injector?: Injector, projectableNodes?: any[][], ngModuleRef?:
NgModuleRef<any>): ComponentRef<C>;
    abstract createEmbeddedView<C>(templateRef: TemplateRef<C>, context?: C,
options?: {
        index?: number;
        injector?: Injector;
    }): EmbeddedViewRef<C>;
    abstract createEmbeddedView<C>(templateRef: TemplateRef<C>, context?: C, index?:
number): EmbeddedViewRef<C>;
    abstract detach(index?: number): ViewRef | null;
    abstract get element(): ElementRef;
    abstract get(index: number): ViewRef | null;
    abstract indexOf(viewRef: ViewRef): number;
    abstract get injector(): Injector;
    abstract insert(viewRef: ViewRef, index?: number): ViewRef;
    abstract get length(): number;
    abstract move(viewRef: ViewRef, currentIndex: number): ViewRef;
    // @deprecated (undocumented)
    abstract get parentInjector(): Injector;
    abstract remove(index?: number): void;
}

// @public
export enum ViewEncapsulation {
    Emulated = 0,
    None = 2,
    ShadowDom = 3
```

```
}

// @public
export abstract class ViewRef extends ChangeDetectorRef {
    abstract destroy(): void;
    abstract get destroyed(): boolean;
    abstract onDestroy(callback: Function): any /** TODO #9100 */;
}

// @public
export function eedefineInjectable<T>(opts: {
    token: unknown;
    providedIn?: Type<any> | 'root' | 'platform' | 'any' | null;
    factory: () => T;
}): unknown;

// @public
export function eeinject<T>(token: ProviderToken<T>): T;

// @public (undocumented)
export function eeinject<T>(token: ProviderToken<T>, flags?: InjectFlags): T | null;

// @public
export function eeinjectAttribute(attrNameToInject: string): string | null;

// (No @packageDocumentation comment for this package)
```