API Report File for "@angular/router"

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

```ts
import { AfterContentInit } from '@angular/core';
import { ChangeDetectorRef } from '@angular/core';
import { Compiler } from '@angular/core';
import { ComponentFactoryResolver } from '@angular/core';
import { ComponentRef } from '@angular/core';
import { ElementRef } from '@angular/core';
import { EventEmitter } from '@angular/core';
import * as i0 from '@angular/core';
import { InjectionToken } from '@angular/core';
import { Injector } from '@angular/core';
import { Location as Location_2 } from '@angular/common';
import { LocationStrategy } from '@angular/common';
import { ModuleWithProviders } from '@angular/core';
import { NgModuleFactory } from '@angular/core';
import { Observable } from 'rxjs';
import { OnChanges } from '@angular/core';
import { OnDestroy } from '@angular/core';
import { OnInit } from '@angular/core';
import { QueryList } from '@angular/core';
import { Renderer2 } from '@angular/core';
import { SimpleChanges } from '@angular/core';
import { Title } from '@angular/platform-browser';
import { Type } from '@angular/core';
import { Version } from '@angular/core';
import { ViewContainerRef } from '@angular/core';

// @public
export class ActivatedRoute {
    get children(): ActivatedRoute[];
    component: Type<any> | string | null;
    data: Observable<Data>;
    get firstChild(): ActivatedRoute | null;
    fragment: Observable<string | null>;
    outlet: string;
    get paramMap(): Observable<ParamMap>;
    params: Observable<Params>;
    get parent(): ActivatedRoute | null;
    get pathFromRoot(): ActivatedRoute[];
    get queryParamMap(): Observable<ParamMap>;
    queryParams: Observable<Params>;
    get root(): ActivatedRoute;
    get routeConfig(): Route | null;
    snapshot: ActivatedRouteSnapshot;
    // (undocumented)
    toString(): string;
    url: Observable<UrlSegment[]>;
```

```
}

// @public
export class ActivatedRouteSnapshot {
    get children(): ActivatedRouteSnapshot[];
    component: Type<any> | string | null;
    data: Data;
    get firstChild(): ActivatedRouteSnapshot | null;
    fragment: string | null;
    outlet: string;
    // (undocumented)
    get paramMap(): ParamMap;
    params: Params;
    get parent(): ActivatedRouteSnapshot | null;
    get pathFromRoot(): ActivatedRouteSnapshot[];
    // (undocumented)
    get queryParamMap(): ParamMap;
    queryParams: Params;
    get root(): ActivatedRouteSnapshot;
    readonly routeConfig: Route | null;
    // (undocumented)
    toString(): string;
    url: UrlSegment[];
}

// @public
export class ActivationEnd {
    constructor(
    snapshot: ActivatedRouteSnapshot);
    // (undocumented)
    snapshot: ActivatedRouteSnapshot;
    // (undocumented)
    toString(): string;
}

// @public
export class ActivationStart {
    constructor(
    snapshot: ActivatedRouteSnapshot);
    // (undocumented)
    snapshot: ActivatedRouteSnapshot;
    // (undocumented)
    toString(): string;
}

// @public
export abstract class BaseRouteReuseStrategy implements RouteReuseStrategy {
    retrieve(route: ActivatedRouteSnapshot): DetachedRouteHandle | null;
    shouldAttach(route: ActivatedRouteSnapshot): boolean;
    shouldDetach(route: ActivatedRouteSnapshot): boolean;
    shouldReuseRoute(future: ActivatedRouteSnapshot, curr: ActivatedRouteSnapshot):
boolean;
```

```
    store(route: ActivatedRouteSnapshot, detachedTree: DetachedRouteHandle): void;
}

// @public
export interface CanActivate {
    // (undocumented)
    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree;
}

// @public
export interface CanActivateChild {
    // (undocumented)
    canActivateChild(childRoute: ActivatedRouteSnapshot, state:
RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> |
boolean | UrlTree;
}

// @public
export interface CanDeactivate<T> {
    // (undocumented)
    canDeactivate(component: T, currentRoute: ActivatedRouteSnapshot, currentState:
RouterStateSnapshot, nextState?: RouterStateSnapshot): Observable<boolean | UrlTree>
| Promise<boolean | UrlTree> | boolean | UrlTree;
}

// @public
export interface CanLoad {
    // (undocumented)
    canLoad(route: Route, segments: UrlSegment[]): Observable<boolean | UrlTree> |
Promise<boolean | UrlTree> | boolean | UrlTree;
}

// @public
export class ChildActivationEnd {
    constructor(
    snapshot: ActivatedRouteSnapshot);
    // (undocumented)
    snapshot: ActivatedRouteSnapshot;
    // (undocumented)
    toString(): string;
}

// @public
export class ChildActivationStart {
    constructor(
    snapshot: ActivatedRouteSnapshot);
    // (undocumented)
    snapshot: ActivatedRouteSnapshot;
    // (undocumented)
    toString(): string;
}
```

```
// @public
export class ChildrenOutletContexts {
    // (undocumented)
    getContext(childName: string): OutletContext | null;
    // (undocumented)
    getOrCreateContext(childName: string): OutletContext;
    onChildOutletCreated(childName: string, outlet: RouterOutletContract): void;
    onChildOutletDestroyed(childName: string): void;
    onOutletDeactivated(): Map<string, OutletContext>;
    // (undocumented)
    onOutletReAttached(contexts: Map<string, OutletContext>): void;
}

// @public
export function convertToParamMap(params: Params): ParamMap;

// @public
export type Data = {
    [key: string | symbol]: any;
};

// @public
export class DefaultTitleStrategy extends TitleStrategy {
    constructor(title: Title);
    // (undocumented)
    readonly title: Title;
    updateTitle(snapshot: RouterStateSnapshot): void;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<DefaultTitleStrategy, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<DefaultTitleStrategy>;
}

// @public
export class DefaultUrlSerializer implements UrlSerializer {
    parse(url: string): UrlTree;
    serialize(tree: UrlTree): string;
}

// @public
export type DetachedRouteHandle = {};

// @public
type Event_2 = RouterEvent | RouteConfigLoadStart | RouteConfigLoadEnd |
ChildActivationStart | ChildActivationEnd | ActivationStart | ActivationEnd |
Scroll;
export { Event_2 as Event }

// @public
export interface ExtraOptions {
    anchorScrolling?: 'disabled' | 'enabled';
```

```
    canceledNavigationResolution?: 'replace' | 'computed';
    enableTracing?: boolean;
    errorHandler?: ErrorHandler;
    initialNavigation?: InitialNavigation;
    malformedUriErrorHandler?: (error: URIError, urlSerializer: UrlSerializer, url:
string) => UrlTree;
    onSameUrlNavigation?: 'reload' | 'ignore';
    paramsInheritanceStrategy?: 'emptyOnly' | 'always';
    preloadingStrategy?: any;
    // @deprecated
    relativeLinkResolution?: 'legacy' | 'corrected';
    scrollOffset?: [number, number] | (() => [number, number]);
    scrollPositionRestoration?: 'disabled' | 'enabled' | 'top';
    urlUpdateStrategy?: 'deferred' | 'eager';
    useHash?: boolean;
}

// @public
export class GuardsCheckEnd extends RouterEvent {
    constructor(
    id: number,
    url: string,
    urlAfterRedirects: string,
    state: RouterStateSnapshot,
    shouldActivate: boolean);
    // (undocumented)
    shouldActivate: boolean;
    // (undocumented)
    state: RouterStateSnapshot;
    // (undocumented)
    toString(): string;
    // (undocumented)
    urlAfterRedirects: string;
}

// @public
export class GuardsCheckStart extends RouterEvent {
    constructor(
    id: number,
    url: string,
    urlAfterRedirects: string,
    state: RouterStateSnapshot);
    // (undocumented)
    state: RouterStateSnapshot;
    // (undocumented)
    toString(): string;
    // (undocumented)
    urlAfterRedirects: string;
}

// @public
export type InitialNavigation = 'disabled' | 'enabled' | 'enabledBlocking' |
```

```
'enabledNonBlocking';

// @public
export interface IsActiveMatchOptions {
    fragment: 'exact' | 'ignored';
    matrixParams: 'exact' | 'subset' | 'ignored';
    paths: 'exact' | 'subset';
    queryParams: 'exact' | 'subset' | 'ignored';
}

// @public
export type LoadChildren = LoadChildrenCallback;

// @public
export type LoadChildrenCallback = () => Type<any> | NgModuleFactory<any> |
Observable<Type<any>> | Promise<NgModuleFactory<any> | Type<any>>;

// @public
export interface Navigation {
    extractedUrl: UrlTree;
    extras: NavigationExtras;
    finalUrl?: UrlTree;
    id: number;
    initialUrl: UrlTree;
    previousNavigation: Navigation | null;
    trigger: 'imperative' | 'popstate' | 'hashchange';
}

// @public
export interface NavigationBehaviorOptions {
    replaceUrl?: boolean;
    skipLocationChange?: boolean;
    state?: {
        [k: string]: any;
    };
}

// @public
export class NavigationCancel extends RouterEvent {
    constructor(
    id: number,
    url: string,
    reason: string);
    // (undocumented)
    reason: string;
    // (undocumented)
    toString(): string;
}

// @public
export class NavigationEnd extends RouterEvent {
    constructor(
```

```
    id: number,
    url: string,
    urlAfterRedirects: string);
    // (undocumented)
    toString(): string;
    // (undocumented)
    urlAfterRedirects: string;
}

// @public
export class NavigationError extends RouterEvent {
    constructor(
    id: number,
    url: string,
    error: any);
    // (undocumented)
    error: any;
    // (undocumented)
    toString(): string;
}

// @public
export interface NavigationExtras extends UrlCreationOptions,
NavigationBehaviorOptions {
}

// @public
export class NavigationStart extends RouterEvent {
    constructor(
    id: number,
    url: string,
    navigationTrigger?: 'imperative' | 'popstate' | 'hashchange',
    restoredState?: {
        [k: string]: any;
        navigationId: number;
    } | null);
    navigationTrigger?: 'imperative' | 'popstate' | 'hashchange';
    restoredState?: {
        [k: string]: any;
        navigationId: number;
    } | null;
    // (undocumented)
    toString(): string;
}

// @public
export class NoPreloading implements PreloadingStrategy {
    // (undocumented)
    preload(route: Route, fn: () => Observable<any>): Observable<any>;
}

// @public
```

```
export class OutletContext {
    // (undocumented)
    attachRef: ComponentRef<any> | null;
    // (undocumented)
    children: ChildrenOutletContexts;
    // (undocumented)
    outlet: RouterOutletContract | null;
    // (undocumented)
    resolver: ComponentFactoryResolver | null;
    // (undocumented)
    route: ActivatedRoute | null;
}

// @public
export interface ParamMap {
    get(name: string): string | null;
    getAll(name: string): string[];
    has(name: string): boolean;
    readonly keys: string[];
}

// @public
export type Params = {
    [key: string]: any;
};

// @public
export class PreloadAllModules implements PreloadingStrategy {
    // (undocumented)
    preload(route: Route, fn: () => Observable<any>): Observable<any>;
}

// @public
export abstract class PreloadingStrategy {
    // (undocumented)
    abstract preload(route: Route, fn: () => Observable<any>): Observable<any>;
}

// @public
export const PRIMARY_OUTLET = "primary";

// @public
export function provideRoutes(routes: Routes): any;

// @public
export type QueryParamsHandling = 'merge' | 'preserve' | '';

// @public
export interface Resolve<T> {
    // (undocumented)
    resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<T> | Promise<T> | T;
```

```
}

// @public
export type ResolveData = {
    [key: string | symbol]: any;
};

// @public
export class ResolveEnd extends RouterEvent {
    constructor(
    id: number,
    url: string,
    urlAfterRedirects: string,
    state: RouterStateSnapshot);
    // (undocumented)
    state: RouterStateSnapshot;
    // (undocumented)
    toString(): string;
    // (undocumented)
    urlAfterRedirects: string;
}

// @public
export class ResolveStart extends RouterEvent {
    constructor(
    id: number,
    url: string,
    urlAfterRedirects: string,
    state: RouterStateSnapshot);
    // (undocumented)
    state: RouterStateSnapshot;
    // (undocumented)
    toString(): string;
    // (undocumented)
    urlAfterRedirects: string;
}

// @public
export interface Route {
    canActivate?: any[];
    canActivateChild?: any[];
    canDeactivate?: any[];
    canLoad?: any[];
    children?: Routes;
    component?: Type<any>;
    data?: Data;
    loadChildren?: LoadChildren;
    matcher?: UrlMatcher;
    outlet?: string;
    path?: string;
    pathMatch?: 'prefix' | 'full';
    redirectTo?: string;
```

```
    resolve?: ResolveData;
    runGuardsAndResolvers?: RunGuardsAndResolvers;
    title?: string | Type<Resolve<string>>;
}

// @public
export class RouteConfigLoadEnd {
    constructor(
    route: Route);
    // (undocumented)
    route: Route;
    // (undocumented)
    toString(): string;
}

// @public
export class RouteConfigLoadStart {
    constructor(
    route: Route);
    // (undocumented)
    route: Route;
    // (undocumented)
    toString(): string;
}

// @public
export class Router {
    constructor(rootComponentType: Type<any> | null, urlSerializer: UrlSerializer,
rootContexts: ChildrenOutletContexts, location: Location_2, injector: Injector,
compiler: Compiler, config: Routes);
    canceledNavigationResolution: 'replace' | 'computed';
    // (undocumented)
    config: Routes;
    createUrlTree(commands: any[], navigationExtras?: UrlCreationOptions): UrlTree;
    dispose(): void;
    errorHandler: ErrorHandler;
    readonly events: Observable<Event_2>;
    getCurrentNavigation(): Navigation | null;
    initialNavigation(): void;
    // @deprecated
    isActive(url: string | UrlTree, exact: boolean): boolean;
    isActive(url: string | UrlTree, matchOptions: IsActiveMatchOptions): boolean;
    malformedUriErrorHandler: (error: URIError, urlSerializer: UrlSerializer, url:
string) => UrlTree;
    navigate(commands: any[], extras?: NavigationExtras): Promise<boolean>;
    navigateByUrl(url: string | UrlTree, extras?: NavigationBehaviorOptions):
Promise<boolean>;
    navigated: boolean;
    // (undocumented)
    ngOnDestroy(): void;
    onSameUrlNavigation: 'reload' | 'ignore';
    paramsInheritanceStrategy: 'emptyOnly' | 'always';
```

```
    parseUrl(url: string): UrlTree;
    // @deprecated
    relativeLinkResolution: 'legacy' | 'corrected';
    resetConfig(config: Routes): void;
    routeReuseStrategy: RouteReuseStrategy;
    readonly routerState: RouterState;
    serializeUrl(url: UrlTree): string;
    setUpLocationChangeListener(): void;
    titleStrategy?: TitleStrategy;
    get url(): string;
    urlHandlingStrategy: UrlHandlingStrategy;
    urlUpdateStrategy: 'deferred' | 'eager';
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<Router, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<Router>;
}

// @public
export const ROUTER_CONFIGURATION: InjectionToken<ExtraOptions>;

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

// @public
export abstract class RouteReuseStrategy {
    abstract retrieve(route: ActivatedRouteSnapshot): DetachedRouteHandle | null;
    abstract shouldAttach(route: ActivatedRouteSnapshot): boolean;
    abstract shouldDetach(route: ActivatedRouteSnapshot): boolean;
    abstract shouldReuseRoute(future: ActivatedRouteSnapshot, curr:
ActivatedRouteSnapshot): boolean;
    abstract store(route: ActivatedRouteSnapshot, handle: DetachedRouteHandle |
null): void;
}

// @public
export class RouterEvent {
    constructor(
    id: number,
    url: string);
    id: number;
    url: string;
}

// @public
export class RouterLink implements OnChanges {
    constructor(router: Router, route: ActivatedRoute, tabIndexAttribute: string |
null | undefined, renderer: Renderer2, el: ElementRef);
    fragment?: string;
    // (undocumented)
    ngOnChanges(changes: SimpleChanges): void;
```

```
    // (undocumented)
    onClick(): boolean;
    preserveFragment: boolean;
    queryParams?: Params | null;
    queryParamsHandling?: QueryParamsHandling | null;
    relativeTo?: ActivatedRoute | null;
    replaceUrl: boolean;
    set routerLink(commands: any[] | string | null | undefined);
    skipLocationChange: boolean;
    state?: {
        [k: string]: any;
    };
    // (undocumented)
    get urlTree(): UrlTree | null;
    // (undocumented)
    static edir: i0.eeDirectiveDeclaration<RouterLink, ":not(a):not(area)
[routerLink]", never, { "queryParams": "queryParams"; "fragment": "fragment";
"queryParamsHandling": "queryParamsHandling"; "preserveFragment":
"preserveFragment"; "skipLocationChange": "skipLocationChange"; "replaceUrl":
"replaceUrl"; "state": "state"; "relativeTo": "relativeTo"; "routerLink":
"routerLink"; }, {}, never>;
    // (undocumented)
    static efac: i0.eeFactoryDeclaration<RouterLink, [null, null, { attribute:
"tabindex"; }, null, null]>;
}

// @public
export class RouterLinkActive implements OnChanges, OnDestroy, AfterContentInit {
    constructor(router: Router, element: ElementRef, renderer: Renderer2, cdr:
ChangeDetectorRef, link?: RouterLink | undefined, linkWithHref?: RouterLinkWithHref
| undefined);
    // (undocumented)
    readonly isActive: boolean;
    readonly isActiveChange: EventEmitter<boolean>;
    // (undocumented)
    links: QueryList<RouterLink>;
    // (undocumented)
    linksWithHrefs: QueryList<RouterLinkWithHref>;
    // (undocumented)
    ngAfterContentInit(): void;
    // (undocumented)
    ngOnChanges(changes: SimpleChanges): void;
    // (undocumented)
    ngOnDestroy(): void;
    // (undocumented)
    set routerLinkActive(data: string[] | string);
    routerLinkActiveOptions: {
        exact: boolean;
    } | IsActiveMatchOptions;
    // (undocumented)
    static edir: i0.eeDirectiveDeclaration<RouterLinkActive, "[routerLinkActive]",
["routerLinkActive"], { "routerLinkActiveOptions": "routerLinkActiveOptions";
```

```
"routerLinkActive": "routerLinkActive"; }, { "isActiveChange": "isActiveChange"; },
["links", "linksWithHrefs"]>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<RouterLinkActive, [null, null, null, null,
{ optional: true; }, { optional: true; }]>;
}

// @public
export class RouterLinkWithHref implements OnChanges, OnDestroy {
    constructor(router: Router, route: ActivatedRoute, locationStrategy:
LocationStrategy);
    fragment?: string;
    // (undocumented)
    href: string | null;
    // (undocumented)
    ngOnChanges(changes: SimpleChanges): any;
    // (undocumented)
    ngOnDestroy(): any;
    // (undocumented)
    onClick(button: number, ctrlKey: boolean, shiftKey: boolean, altKey: boolean,
metaKey: boolean): boolean;
    preserveFragment: boolean;
    queryParams?: Params | null;
    queryParamsHandling?: QueryParamsHandling | null;
    relativeTo?: ActivatedRoute | null;
    replaceUrl: boolean;
    set routerLink(commands: any[] | string | null | undefined);
    skipLocationChange: boolean;
    state?: {
        [k: string]: any;
    };
    // (undocumented)
    target: string;
    // (undocumented)
    get urlTree(): UrlTree | null;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<RouterLinkWithHref,
"a[routerLink],area[routerLink]", never, { "target": "target"; "queryParams":
"queryParams"; "fragment": "fragment"; "queryParamsHandling": "queryParamsHandling";
"preserveFragment": "preserveFragment"; "skipLocationChange": "skipLocationChange";
"replaceUrl": "replaceUrl"; "state": "state"; "relativeTo": "relativeTo";
"routerLink": "routerLink"; }, {}, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<RouterLinkWithHref, never>;
}

// @public
export class RouterModule {
    constructor(guard: any, router: Router);
    static forChild(routes: Routes): ModuleWithProviders<RouterModule>;
    static forRoot(routes: Routes, config?: ExtraOptions):
ModuleWithProviders<RouterModule>;
```

```
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<RouterModule, [{ optional: true; }, {
optional: true; }]>;
    // (undocumented)
    static ɵinj: i0.ɵɵInjectorDeclaration<RouterModule>;
    // (undocumented)
    static ɵmod: i0.ɵɵNgModuleDeclaration<RouterModule, [typeof i1.RouterOutlet,
typeof i2.RouterLink, typeof i2.RouterLinkWithHref, typeof i3.RouterLinkActive,
typeof i4.ɵEmptyOutletComponent], never, [typeof i1.RouterOutlet, typeof
i2.RouterLink, typeof i2.RouterLinkWithHref, typeof i3.RouterLinkActive, typeof
i4.ɵEmptyOutletComponent]>;
}

// @public
export class RouterOutlet implements OnDestroy, OnInit, RouterOutletContract {
    constructor(parentContexts: ChildrenOutletContexts, location: ViewContainerRef,
resolver: ComponentFactoryResolver, name: string, changeDetector:
ChangeDetectorRef);
    // (undocumented)
    get activatedRoute(): ActivatedRoute;
    // (undocumented)
    get activatedRouteData(): Data;
    // (undocumented)
    activateEvents: EventEmitter<any>;
    // (undocumented)
    activateWith(activatedRoute: ActivatedRoute, resolver: ComponentFactoryResolver
| null): void;
    attach(ref: ComponentRef<any>, activatedRoute: ActivatedRoute): void;
    attachEvents: EventEmitter<unknown>;
    // (undocumented)
    get component(): Object;
    // (undocumented)
    deactivate(): void;
    // (undocumented)
    deactivateEvents: EventEmitter<any>;
    detach(): ComponentRef<any>;
    detachEvents: EventEmitter<unknown>;
    // (undocumented)
    get isActivated(): boolean;
    // (undocumented)
    ngOnDestroy(): void;
    // (undocumented)
    ngOnInit(): void;
    // (undocumented)
    static ɵdir: i0.ɵɵDirectiveDeclaration<RouterOutlet, "router-outlet",
["outlet"], {}, { "activateEvents": "activate"; "deactivateEvents": "deactivate";
"attachEvents": "attach"; "detachEvents": "detach"; }, never>;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<RouterOutlet, [null, null, null, {
attribute: "name"; }, null]>;
}
```

```
// @public
export interface RouterOutletContract {
    activatedRoute: ActivatedRoute | null;
    activatedRouteData: Data;
    activateEvents?: EventEmitter<unknown>;
    activateWith(activatedRoute: ActivatedRoute, resolver: ComponentFactoryResolver
| null): void;
    attach(ref: ComponentRef<unknown>, activatedRoute: ActivatedRoute): void;
    attachEvents?: EventEmitter<unknown>;
    component: Object | null;
    deactivate(): void;
    deactivateEvents?: EventEmitter<unknown>;
    detach(): ComponentRef<unknown>;
    detachEvents?: EventEmitter<unknown>;
    isActivated: boolean;
}

// @public
export class RouterPreloader implements OnDestroy {
    constructor(router: Router, compiler: Compiler, injector: Injector,
preloadingStrategy: PreloadingStrategy);
    // (undocumented)
    ngOnDestroy(): void;
    // (undocumented)
    preload(): Observable<any>;
    // (undocumented)
    setUpPreloading(): void;
    // (undocumented)
    static ɵfac: i0.ɵɵFactoryDeclaration<RouterPreloader, never>;
    // (undocumented)
    static ɵprov: i0.ɵɵInjectableDeclaration<RouterPreloader>;
}

// @public
export class RouterState extends Tree<ActivatedRoute> {
    snapshot: RouterStateSnapshot;
    // (undocumented)
    toString(): string;
}

// @public
export class RouterStateSnapshot extends Tree<ActivatedRouteSnapshot> {
    // (undocumented)
    toString(): string;
    url: string;
}

// @public
export const ROUTES: InjectionToken<Route[][]>;

// @public
export type Routes = Route[];
```

```
// @public
export class RoutesRecognized extends RouterEvent {
    constructor(
    id: number,
    url: string,
    urlAfterRedirects: string,
    state: RouterStateSnapshot);
    // (undocumented)
    state: RouterStateSnapshot;
    // (undocumented)
    toString(): string;
    // (undocumented)
    urlAfterRedirects: string;
}

// @public
export type RunGuardsAndResolvers = 'pathParamsChange' |
'pathParamsOrQueryParamsChange' | 'paramsChange' | 'paramsOrQueryParamsChange' |
'always' | ((from: ActivatedRouteSnapshot, to: ActivatedRouteSnapshot) => boolean);

// @public
export class Scroll {
    constructor(
    routerEvent: NavigationEnd,
    position: [number, number] | null,
    anchor: string | null);
    // (undocumented)
    readonly anchor: string | null;
    // (undocumented)
    readonly position: [number, number] | null;
    // (undocumented)
    readonly routerEvent: NavigationEnd;
    // (undocumented)
    toString(): string;
}

// @public
export abstract class TitleStrategy {
    // (undocumented)
    buildTitle(snapshot: RouterStateSnapshot): string | undefined;
    getResolvedTitleForRoute(snapshot: ActivatedRouteSnapshot): any;
    abstract updateTitle(snapshot: RouterStateSnapshot): void;
}

// @public
export interface UrlCreationOptions {
    fragment?: string;
    preserveFragment?: boolean;
    queryParams?: Params | null;
    queryParamsHandling?: QueryParamsHandling | null;
    relativeTo?: ActivatedRoute | null;
```

```
}

// @public
export abstract class UrlHandlingStrategy {
    abstract extract(url: UrlTree): UrlTree;
    abstract merge(newUrlPart: UrlTree, rawUrl: UrlTree): UrlTree;
    abstract shouldProcessUrl(url: UrlTree): boolean;
}

// @public
export type UrlMatcher = (segments: UrlSegment[], group: UrlSegmentGroup, route:
Route) => UrlMatchResult | null;

// @public
export type UrlMatchResult = {
    consumed: UrlSegment[];
    posParams?: {
        [name: string]: UrlSegment;
    };
};

// @public
export class UrlSegment {
    constructor(
    path: string,
    parameters: {
        [name: string]: string;
    });
    // (undocumented)
    get parameterMap(): ParamMap;
    parameters: {
        [name: string]: string;
    };
    path: string;
    // (undocumented)
    toString(): string;
}

// @public
export class UrlSegmentGroup {
    constructor(
    segments: UrlSegment[],
    children: {
        [key: string]: UrlSegmentGroup;
    });
    children: {
        [key: string]: UrlSegmentGroup;
    };
    hasChildren(): boolean;
    get numberOfChildren(): number;
    parent: UrlSegmentGroup | null;
    segments: UrlSegment[];
```

```
    // (undocumented)
    toString(): string;
}

// @public
export abstract class UrlSerializer {
    abstract parse(url: string): UrlTree;
    abstract serialize(tree: UrlTree): string;
}

// @public
export class UrlTree {
    fragment: string | null;
    // (undocumented)
    get queryParamMap(): ParamMap;
    queryParams: Params;
    root: UrlSegmentGroup;
    // (undocumented)
    toString(): string;
}

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

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