# The TypeScript Handbook Reference

## Utility Types

TypeScript provides several utility types to facilitate common type transformations. These utilities are available globally.

## Partial<Type>

Constructs a type with all properties of Type set to optional. This utility will return a type that represents all subsets of a given type.

##### Example

interface Todo {

title: string;

description: string;

}

function updateTodo(todo: Todo, fieldsToUpdate: Partial<Todo>) {

return { ...todo, ...fieldsToUpdate };

}

const todo1 = {

title: "organize desk",

description: "clear clutter",

};

const todo2 = updateTodo(todo1, {

description: "throw out trash",

});[Try](https://www.typescriptlang.org/play/#code/JYOwLgpgTgZghgYwgAgCoHsAm7kG8BQyyYwYANhAFzIDOYUoA5gNyHKYQ0IMAOJ6IanQYgW+AL758MAK4gE-EMhk9McSBmwAKMFnTVN6ADTIYwCGUw0MAVVXqqyAApwoJOGQA8hgHwBKPDYoCDAZKCVcZAA6GN1sExioswsrW3tIZHFWSXwEATpiPQBGZABeQKISckcAInQoRjgQYAAvFA4aAGsaozYO7mA+YAFqGoQKV2RxmTBIKB6JVlz8sELsACYy5XSIQx1ikwIift5FUbAACyh0AHdkdBniKDgaC4XxP2YgA)

## Readonly<Type>

Constructs a type with all properties of Type set to readonly, meaning the properties of the constructed type cannot be reassigned.

##### Example

interface Todo {

title: string;

}

const todo: Readonly<Todo> = {

title: "Delete inactive users",

};

todo.title = "Hello";

Cannot assign to 'title' because it is a read-only property.Cannot assign to 'title' because it is a read-only property.[Try](https://www.typescriptlang.org/play/#code/PTAEAEFMCdoe2gZwFygEwFYAsAGAUAJYB2ALjAGYCGAxpKACpwAmcoA3nqKCQSQDaRUiEtGIBzANx4AvnjzU4RYd2ZxUAJUiUWRPgE8APIxYA+UAF52nbrwGoARABFIAsqGI0eANzoBXRDCI9gA0MlJ4JKoAdDz8dJb2ABIufHD2EkA)

This utility is useful for representing assignment expressions that will fail at runtime (i.e. when attempting to reassign properties of a [frozen object](https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze)).

##### Object.freeze

function freeze<Type>(obj: Type): Readonly<Type>;

## Record<Keys,Type>

Constructs an object type whose property keys are Keys and whose property values are Type. This utility can be used to map the properties of a type to another type.

##### Example

interface PageInfo {

title: string;

}

type Page = "home" | "about" | "contact";

const nav: Record<Page, PageInfo> = {

about: { title: "about" },

contact: { title: "contact" },

home: { title: "home" },

};

nav.about;

// ^ = const nav: Record[Try](https://www.typescriptlang.org/play/#code/JYOwLgpgTgZghgYwgAgApwOYQJIhge2QG8AoZZMYMAGwgC5kBnMKUDAbhIF8SSwBPAA4p0WZAF5kAIgAW+ALYQpyAD7S4AI3wBXMMrVSE+cIj2cSRkM2Qg4ANwYAlCEagATADyiIAGjSYcPHwAPgliMmRNHTAGIgoqWgYpKN1lLh8IyzBTWPiaemks0zSM8jlFXMp8pPKlZHTuc1s7ADoUsE4Aek7kAD0AfiA)

## Pick<Type, Keys>

Constructs a type by picking the set of properties Keys from Type.

##### Example

interface Todo {

title: string;

description: string;

completed: boolean;

}

type TodoPreview = Pick<Todo, "title" | "completed">;

const todo: TodoPreview = {

title: "Clean room",

completed: false,

};

todo;

// ^ = const todo: TodoPreview[Try](https://www.typescriptlang.org/play/#code/JYOwLgpgTgZghgYwgAgCoHsAm7kG8BQyyYwYANhAFzIDOYUoA5gNyHKYQ0IMAOJ6IanQYgWbBOgC2PCpEzUARunQU4IVgF98+MAE8eKDNgAKUCADdgEAO7IAvMmPAEAawA8R9ABpkAIhLkEL7IAD5+EtKyEJi+AHys+BIgdMRY6NSephZWtg4ERAEU1L4AwqogyFDKkr5e4lIyEHLU8GQ0EHUaCWBprAD0fcgAegD8QA)

## Omit<Type, Keys>

Constructs a type by picking all properties from Type and then removing Keys.

##### Example

interface Todo {

title: string;

description: string;

completed: boolean;

}

type TodoPreview = Omit<Todo, "description">;

const todo: TodoPreview = {

title: "Clean room",

completed: false,

};

todo;

// ^ = const todo: TodoPreview[Try](https://www.typescriptlang.org/play/#code/JYOwLgpgTgZghgYwgAgCoHsAm7kG8BQyyYwYANhAFzIDOYUoA5gNyHKYQ0IMAOJ6IanQYgWbBOgC2PCpEzUARunQU4IVgF98+MAE8eKDNgAKUCADdgEAO7IAvMgDyk0gB4j6ADTIARBy68-CA+AHys+BIgdMRY6NQephZWtg4ERCTkVL4AwqogyFDKkj6e4lIyEHLU8GQ0EKUa4WCxrAD0rcgAegD8QA)

## Exclude<Type, ExcludedUnion>

Constructs a type by excluding from Type all union members that are assignable to ExcludedUnion.

##### Example

type T0 = Exclude<"a" | "b" | "c", "a">;

// ^ = type T0 = "b" | "c"

type T1 = Exclude<"a" | "b" | "c", "a" | "b">;

// ^ = type T1 = "c"

type T2 = Exclude<string | number | (() => void), Function>;

// ^ = type T2 = string | number[Try](https://www.typescriptlang.org/play/#code/C4TwDgpgBAKgDFAvFAogDwMYBsCuATCAHgCIBDYqAHymICMLriNiAaG8gPgG4AoAej5QhUAHoB+HqEiwAjElSZcBEuSo16apq3YN1xbv0HDxk8NBgAmeemz4iAZ2AAnAJYA7AOZq3OALa0IJzUACmCASiQOKAA3AHsXPDC2ADEcNwxgF1i3AwFhUTEgA)

## Extract<Type, Union>

Constructs a type by extracting from Type all union members that are assignable to Union.

##### Example

type T0 = Extract<"a" | "b" | "c", "a" | "f">;

// ^ = type T0 = "a"

type T1 = Extract<string | number | (() => void), Function>;

// ^ = type T1 = () => void[Try](https://www.typescriptlang.org/play/#code/C4TwDgpgBAKgDFAvFAogD2AJwIYGNgA8ARNkVAD5REBGZlRuRANFaRVQGZEB8A3AFAB6QVFFQAegH5+oSLACMSVBhz4CAZywBLAHYBzdjoCuAW2oRM7ABRWAlEm5QAbgHstAE1ssAYkZ34tFx0+IRExKSA)

## NonNullable<Type>

Constructs a type by excluding null and undefined from Type.

##### Example

type T0 = NonNullable<string | number | undefined>;

// ^ = type T0 = string | number

type T1 = NonNullable<string[] | null | undefined>;

// ^ = type T1 = string[][Try](https://www.typescriptlang.org/play/#code/C4TwDgpgBAKgDFAvFAcgewHYoK4BtcCGARrhADwDOwATgJYYDmUAPlBtgLZETUtTYYAJhABm9CIIB8AbgBQAenlRlUAHoB+WaEiwAjElSYc+YqUo16DANoBdPu3x8BwsRgkyFSlRqA)

## Parameters<Type>

Constructs a tuple type from the types used in the parameters of a function type Type.

##### Example

declare function f1(arg: { a: number; b: string }): void;

type T0 = Parameters<() => string>;

// ^ = type T0 = []

type T1 = Parameters<(s: string) => void>;

// ^ = type T1 = [s: string]

type T2 = Parameters<<T>(arg: T) => T>;

// ^ = type T2 = [arg: unknown]

type T3 = Parameters<typeof f1>;

// ^ = type T3 = [arg: {

// a: number;

// b: string;

// }]

type T4 = Parameters<any>;

// ^ = type T4 = unknown[]

type T5 = Parameters<never>;

// ^ = type T5 = never

type T6 = Parameters<string>;

Type 'string' does not satisfy the constraint '(...args: any) => any'.Type 'string' does not satisfy the constraint '(...args: any) => any'.// ^ = type T6 = never

type T7 = Parameters<Function>;

Type 'Function' does not satisfy the constraint '(...args: any) => any'.

Type 'Function' provides no match for the signature '(...args: any): any'.Type 'Function' does not satisfy the constraint '(...args: any) => any'.

Type 'Function' provides no match for the signature '(...args: any): any'.// ^ = type T7 = never[Try](https://www.typescriptlang.org/play/#code/PTAEAEFMCdoe2gZwFygEwGYAsWBQATSAYwBsBDaSUAMwFcA7IgFwEs56aBGACgoHNUAb1BlU9WgFsARjADcoKakRNoLen1ABfAJSoAbnBb5ZuXEwCeAByoAVAAygAvKAAKFMhMhMYiADzdtJwA+UGVVdSCTEFAY0AA9AH4zK1tOJ1d3T28kfxRQlTU+QMcQgyNI3GjYxOTrUBs0dLdoDy8fX18bIN5oAXrikK6osGqkizqbDCbMtpzxyDhqLgqqmJr5+qxplqz2snpzFZG1sZT6gFZt1uy-ekg9GCPY+NOJgDYr3ZywwqfR2tsAHZPrM-AAxBjMNj0P4nIA)

## ConstructorParameters<Type>

Constructs a tuple or array type from the types of a constructor function type. It produces a tuple type with all the parameter types (or the type never if Type is not a function).

##### Example

type T0 = ConstructorParameters<ErrorConstructor>;

// ^ = type T0 = [message?: string]

type T1 = ConstructorParameters<FunctionConstructor>;

// ^ = type T1 = string[]

type T2 = ConstructorParameters<RegExpConstructor>;

// ^ = type T2 = [pattern: string | RegExp, flags?: string]

type T3 = ConstructorParameters<any>;

// ^ = type T3 = unknown[]

type T4 = ConstructorParameters<Function>;

Type 'Function' does not satisfy the constraint 'new (...args: any) => any'.

Type 'Function' provides no match for the signature 'new (...args: any): any'.Type 'Function' does not satisfy the constraint 'new (...args: any) => any'.

Type 'Function' provides no match for the signature 'new (...args: any): any'.// ^ = type T4 = never[Try](https://www.typescriptlang.org/play/#code/PTAEAEFMCdoe2gZwFygEwGYAsWBQIJEAXaASwGMjUAzAQwBtFJciBPAB0lABUAGUALygAwnAB2xaAFdKCAAq1otALaQiMRAB4AorASiJJGUQQA+ANz4woGwD0A-Cw5duARkEjxk4-MUq1GpoAYlJilKTiBt6y0BZWNnaObJw8aB5RRjEKSqrqSJoASpAA5toAHuwZ0jFxBAkOTincGOlemSbQ2f55WrRirLXWibiNLlithtUdXbmBIWFEEWKDCaAOQA)

## ReturnType<Type>

Constructs a type consisting of the return type of function Type.

##### Example

declare function f1(): { a: number; b: string };

type T0 = ReturnType<() => string>;

// ^ = type T0 = string

type T1 = ReturnType<(s: string) => void>;

// ^ = type T1 = void

type T2 = ReturnType<<T>() => T>;

// ^ = type T2 = unknown

type T3 = ReturnType<<T extends U, U extends number[]>() => T>;

// ^ = type T3 = number[]

type T4 = ReturnType<typeof f1>;

// ^ = type T4 = {

// a: number;

// b: string;

// }

type T5 = ReturnType<any>;

// ^ = type T5 = any

type T6 = ReturnType<never>;

// ^ = type T6 = never

type T7 = ReturnType<string>;

Type 'string' does not satisfy the constraint '(...args: any) => any'.Type 'string' does not satisfy the constraint '(...args: any) => any'.// ^ = type T7 = any

type T8 = ReturnType<Function>;

Type 'Function' does not satisfy the constraint '(...args: any) => any'.

Type 'Function' provides no match for the signature '(...args: any): any'.Type 'Function' does not satisfy the constraint '(...args: any) => any'.

Type 'Function' provides no match for the signature '(...args: any): any'.// ^ = type T8 = any[Try](https://www.typescriptlang.org/play/#code/PTAEAEFMCdoe2gZwFygEwGYAsX3awFAAmkAxgDYCG0koAZgK4B2pALgJZxP0CMAFAEpUAb1CVUTBgFsARjADcoGakSto7JgHNQAX3kECrAJ4AHWgBUADKAC8oAEqRWDaE3OnIAHkG2AfKFV1LV99EFBw0AA9AH5DD1BzHlsHJxc3D28UALUNTQE-UAA3OHYiEIIwiJi4swS0ZMdnV3czT09zXx8bfw7QsCrY41rzDAbU5oz20EgAD1ZIJiJEUABVABpV6bmFpdBJWRgAbQBdTvzuhPLK8Oqhi1w7RrSWrzu4Ol4r-pvB+PMAVjGTXSrUoTCMXwiUV+wwAbEDnhkmJBCjBIQMahYAOwIiatQK5dE-TEJAAcuJBXgAYsw2JwmEToUA)

## InstanceType<Type>

Constructs a type consisting of the instance type of a constructor function in Type.

##### Example

class C {

x = 0;

y = 0;

}

type T0 = InstanceType<typeof C>;

// ^ = type T0 = C

type T1 = InstanceType<any>;

// ^ = type T1 = any

type T2 = InstanceType<never>;

// ^ = type T2 = never

type T3 = InstanceType<string>;

Type 'string' does not satisfy the constraint 'new (...args: any) => any'.Type 'string' does not satisfy the constraint 'new (...args: any) => any'.// ^ = type T3 = any

type T4 = InstanceType<Function>;

Type 'Function' does not satisfy the constraint 'new (...args: any) => any'.

Type 'Function' provides no match for the signature 'new (...args: any): any'.Type 'Function' does not satisfy the constraint 'new (...args: any) => any'.

Type 'Function' provides no match for the signature 'new (...args: any): any'.// ^ = type T4 = any[Try](https://www.typescriptlang.org/play/#code/PTAEAEFMCdoe2gZwFygEwGYAsX3awFAgSIAu0AlgMamoBmAhgDaKQFVMOKKgDCoAbwKhQAD1ABeUAAYA3MNABPSTPkBfAgVKKADpFAAVaSoCSAOzIMzVSAd2QAPNr1w6fAHzziI0AD0A-Fr2hgCMphakVjZ2eg5Wip5EYD4BQXqGaOGW1rb2DmaQAG4wid4iqc76BhhZkTkxjmSUZgDmpcnlgZWGuFLm2dF5AGIArtakFHBm7T5+-kA)

## Required<Type>

Constructs a type consisting of all properties of Type set to required. The opposite of [Partial](https://www.typescriptlang.org/docs/handbook/utility-types.html#partialtype).

##### Example

interface Props {

a?: number;

b?: string;

}

const obj: Props = { a: 5 };

const obj2: Required<Props> = { a: 5 };

Property 'b' is missing in type '{ a: number; }' but required in type 'Required<Props>'.Property 'b' is missing in type '{ a: number; }' but required in type 'Required<Props>'.[Try](https://www.typescriptlang.org/play/#code/PTAEAEFMCdoe2gZwFygEwHYAsBGAUAJYB2ALjAGYCGAxpKAArwAOioA3nqKJQPypEBXALYAjGAG5OoEX1CIS0YgHNJAXzx5qcIvNBwRAK1SM4LUAF523VAFZQqyZu279BtKgBKkAI4CC0SAATAB4TFgA+CytKW3txIA)

## ThisParameterType<Type>

Extracts the type of the [this](https://www.typescriptlang.org/docs/handbook/functions.html#this-parameters) parameter for a function type, or [unknown](https://www.typescriptlang.org/docs/handbook/release-notes/typescript-3-0.html#new-unknown-top-type) if the function type has no this parameter.

##### Example

function toHex(this: Number) {

return this.toString(16);

}

function numberToString(n: ThisParameterType<typeof toHex>) {

return toHex.apply(n);

}[Try](https://www.typescriptlang.org/play/#code/GYVwdgxgLglg9mABFOAJApgDwBRQBYwDOAXIgHIgC2ARugE4CUiA3gFCKJ3pQh1L5EAdCgDKUOjDABzbAEYAbAwDcrAL6tWoSLASIwVWnQAqcMROnYwpIwUIAFAIZ0HlbvSMBPAA7oAPFG90OGBkNCwAPiY2Di4ePlCMTEEHLy8AGw9LZTUgA)

## OmitThisParameter<Type>

Removes the [this](https://www.typescriptlang.org/docs/handbook/functions.html#this-parameters) parameter from Type. If Type has no explicitly declared this parameter, the result is simply Type. Otherwise, a new function type with no this parameter is created from Type. Generics are erased and only the last overload signature is propagated into the new function type.

##### Example

function toHex(this: Number) {

return this.toString(16);

}

const fiveToHex: OmitThisParameter<typeof toHex> = toHex.bind(5);

console.log(fiveToHex());[Try](https://www.typescriptlang.org/play/#code/GYVwdgxgLglg9mABFOAJApgDwBRQBYwDOAXIgHIgC2ARugE4CUiA3gFCKJ3pQh1L5EAdCgDKUOjDABzbAEYAbAwDcrAL6tWEBISiJgMAG7oAKmiykA8pRhRjBQgAUAhnSeVu9ADxQAngAd0OGBkM0wAPkQAXhCMTEFqSQATbABWZQ0tMEI4ABt0QRy4GX0jU1jsBmUgA)

## ThisType<Type>

This utility does not return a transformed type. Instead, it serves as a marker for a contextual [this](https://www.typescriptlang.org/docs/handbook/functions.html#this) type. Note that the --noImplicitThis flag must be enabled to use this utility.

##### Example

type ObjectDescriptor<D, M> = {

data?: D;

methods?: M & ThisType<D & M>; // Type of 'this' in methods is D & M

};

function makeObject<D, M>(desc: ObjectDescriptor<D, M>): D & M {

let data: object = desc.data || {};

let methods: object = desc.methods || {};

return { ...data, ...methods } as D & M;

}

let obj = makeObject({

data: { x: 0, y: 0 },

methods: {

moveBy(dx: number, dy: number) {

this.x += dx; // Strongly typed this

this.y += dy; // Strongly typed this

},

},

});

obj.x = 10;

obj.y = 20;

obj.moveBy(5, 5);[Try](https://www.typescriptlang.org/play/#)

In the example above, the methods object in the argument to makeObject has a contextual type that includes ThisType<D & M> and therefore the type of [this](https://www.typescriptlang.org/docs/handbook/functions.html#this) in methods within the methods object is { x: number, y: number } & { moveBy(dx: number, dy: number): number }. Notice how the type of the methods property simultaneously is an inference target and a source for the this type in methods.

The ThisType<T> marker interface is simply an empty interface declared in lib.d.ts. Beyond being recognized in the contextual type of an object literal, the interface acts like any empty interface.

## Intrinsic String Manipulation Types

To help with string manipulation around template string literals, TypeScript includes a set of types which can be used in string manipulation within the type system. You can find those in the [Template Literal Types](https://www.typescriptlang.org/docs/handbook/2/template-literal-types.html#uppercasestringtype) documentation.