TypeScript Utility Types

TypeScript comes with a large number of types that can help with some common type manipulation, usually referred to as utility types.

Partial

`Partial` changes all the properties in an object to be optional.

Example

```
interface Point {
  x: number;
  y: number;
}

let pointPart: Partial<Point> = {}; // `Partial` allows x and y to be
optional
pointPart.x = 10;
```

Required

`Required` changes all the properties in an object to be required.

Example

```
interface Car {
  make: string;
  model: string;
  mileage?: number;
}

let myCar: Required<Car> = {
  make: 'Ford',
  model: 'Focus',
  mileage: 12000 // `Required` forces mileage to be defined
};
```

Record

`Record` is a shortcut to defining an object type with a specific key type and value type.

Example

```
const nameAgeMap: Record<string, number> = {
  'Alice': 21,
  'Bob': 25
};
```

`Record<string, number>` is equivalent to `{ [key: string]: number }`

Omit

`Omit` removes keys from an object type.

Example

```
interface Person {
  name: string;
  age: number;
  location?: string;
}

const bob: Omit<Person, 'age' | 'location'> = {
  name: 'Bob'
  // `Omit` has removed age and location from the type and they can't
be defined here
};
```

Pick

`Pick` removes all but the specified keys from an object type.

Example

```
interface Person {
  name: string;
  age: number;
  location?: string;
}

const bob: Pick<Person, 'name'> = {
  name: 'Bob'
  // `Pick` has only kept name, so age and location were removed from
the type and they can't be defined here
};
```

Exclude

`Exclude` removes types from a union.

Example

```
type Primitive = string | number | boolean
const value: Exclude<Primitive, string> = true; // a string cannot be
used here since Exclude removed it from the type.
```

ReturnType

`ReturnType` extracts the return type of a function type.

Example

```
type PointGenerator = () => { x: number; y: number; };
const point: ReturnType<PointGenerator> = {
  x: 10,
  y: 20
};
```

Parameters

`Parameters` extracts the parameter types of a function type as an array.

Example

```
type PointPrinter = (p: { x: number; y: number; }) => void;
const point: Parameters<PointPrinter>[0] = {
  x: 10,
  y: 20
};
```

Readonly

`Readonly` is used to create a new type where all properties are readonly, meaning they cannot be modified once assigned a value.

Keep in mind TypeScript will prevent this at compile time, but in theory since it is compiled down to JavaScript you can still override a readonly property.

Example

```
interface Person {
  name: string;
  age: number;
}
const person: Readonly<Person> = {
  name: "Dylan",
  age: 35,
};
person.name = 'Israel'; // prog.ts(11,8): error TS2540: Cannot assign
to 'name' because it is a read-only property.
```