Keys with Constants and Symbols

In this lesson, we will study about using keys with constants and symbols.

WE'LL COVER THE FOLLOWING

Constant

Version 2.7 brings the possibility of accessing a property using constant and symbol. This is a minor improvement because before 2.7, it was possible to access (read and write) a constant or symbol but not to define a field in an interface or type.

```
const Foo = "Foo"; // Constant value
const Bar = "Bar"; // Constant value
const Zaz = "Zaz"; // Constant value

const objectWithConstantProperties = { [Foo]: 100, [Bar]: "hello", [Zaz]: () => {} };

let a12: number = objectWithConstantProperties[Foo];
let b2334: string = objectWithConstantProperties[Bar];

console.log(a12);
console.log(b2334);
```

The example defined three constant at line **1, 2 and 3**. At **line 5**, the constants are used in the identifier part (between the square brackets) to assign value. At **line 7-8** the constant is reused for accessing the values.

The idea behind using a constant is to avoid miswriting a string.

```
const objNoConst: { [id: string]: number } = { ["key1"]: 1 };

objNoConst["key2"] = 2;

console.log(objNoConst["key1"]);
console.log(objNoConst["key2"]);
console.log(objNoConst["key11"]);
```

The example above at **line** 7 demonstrate that the flexibility of the index signature can be easily correct for TypeScript but incorrect in execution. The problem is that there is no key defined for the identifier `key11`. It is probably a typo. Forcing developer to use constant is a way to avoid a mistake of name.

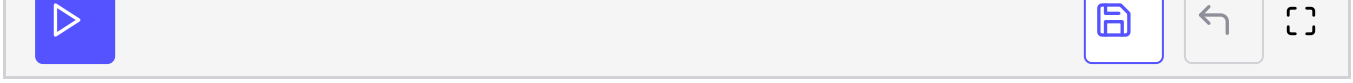
Symbol

The syntax is the same as when accessing the field for the consumption of a constant or a symbol: square brackets.

```
const SERIALIZE_1 = Symbol("serialize-method-key"); // Symbol
const SERIALIZE_2 = "serialize-method-key"; // Constant

const o1 = { [SERIALIZE_1]: "1", [SERIALIZE_2]: "2" };

let s1: string = o1[SERIALIZE_1];
let s2: string = o1[SERIALIZE_2];
console.log(s1);
console.log(s2);
```

It is also possible to use a constant or a symbol to define an interface with an index signature.

```
const SERIALIZE1 = Symbol("serialize-method-key"); // Symbol
interface Serializable1 {
    [SERIALIZE1]: string;
}

let seria1: Serializable1 = { [SERIALIZE1] : "" };
```



Caveat

You may have a defined set of keys that you want to enforce. Specifying `string` would be too broad. The following code shows that it is not possible to define an array of strings (see commented `MyTypeB`) and requires you to have a separate type with the property's names.

```
interface MyTypeA {
  [k: string]: string;
}

// interface MyTypeB {
//   [k: "id1" | "id2"]: string;
// }

type AllowedKey = "Id1" | "Id2";
type MyTypeC = {
  [k in AllowedKey]: string;
}
let c: MyTypeC = { "Id1": "123", "Id2": "ABC" };
```