With Members of the Same Type

This lesson explains how to have members in a type defined with an index signature.

WE'LL COVER THE FOLLOWING

Same type

The interface defined by members and an index type that uses a string as the key, must have all its members to be of type string.

> 📜 **Note**: the code below throws an error ✕

```
interface MyStringDictionaryWithMembers {
    [key: string]: string;
    m1: string;
    m2: number; // Won't transpile, must be a string
}
```

If we correct the previous example, to have `m2` to be a `string` at **line 4**, the code compiles.

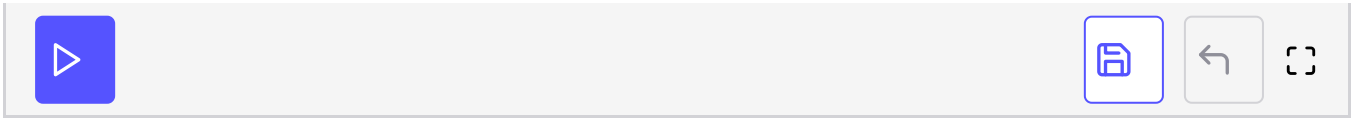
```
interface MyStringDictionaryWithMembers {
    [key: string]: string;
    m1: string;
    m2: string; // Fixed!
}
```

From here, it is possible to use by create an object that is defined with the interface name.

```
interface MyStringDictionaryWithMembers {
  [key: string]: string;
  m1: string;
  m2: string; // Fixed!
}

let map: MyStringDictionaryWithMembers = {
  m1: "value1",
  m2: "value2",
  ["stringHere"]: "stringValue"
};

console.log(map);
```

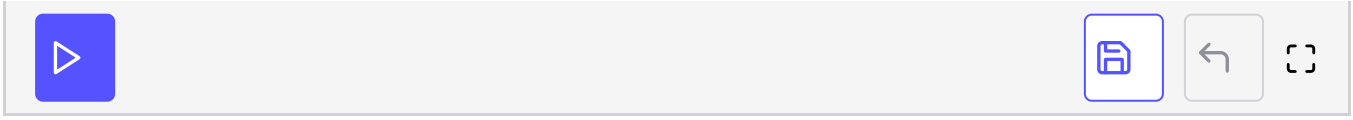


It is also possible to implement the class if required more object-oriented features.

```
interface MyStringDictionaryWithMembers {
  [key: string]: string;
  m1: string;
  m2: string; // Fixed!
}

class MyMap implements MyStringDictionaryWithMembers {
  constructor() {
    this.m1 = "";
    this.m2 = "";
  }
  [key: string]: string;
  m1: string;
  m2: string; // Fixed!
};

let myMap = new MyMap();
myMap.m1 = "M1Value";
myMap.m2 = "M2Value";
myMap["key1"] = "val1";
myMap["key2"] = "val2";
console.log(myMap);
```

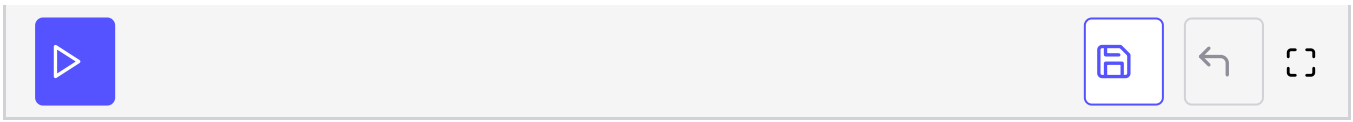


Different types

Different types

A way to work around this restriction would be to have the index be a union of many types. This would allow all union types to be available as member types.

```
interface MyStringDictionaryWithMembers2 {
    [key: string]: string | number;
    m1: string;
    m2: number;
}
```

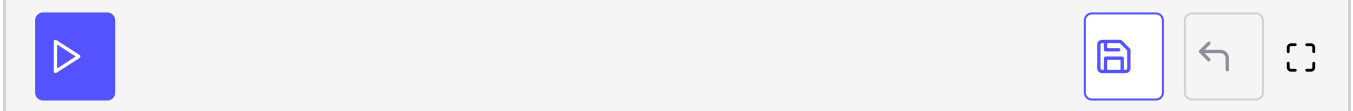


Implementing an interface with index signatures

You may wonder how you can implement an interface with members of an existing class. The same rules apply: you can only use members of the type defined by the index signature. It is not possible to create a union with a type of member of the class. The code below does not compile until the `| number` and `m2: number` are removed.

```
interface MyStringDictionaryWithMembers {
  [key: string]: string;
  m1: string;
}

class MyClass implements MyStringDictionaryWithMembers {
  [key: string]: string | number;
  m1: string;
  m2: number;
}
```



To add the `m2: number` to an existing class, the only possibility is to create a new type that will intersect the new type. The intersection is similar to bringing two different types together instead of one inheriting the other's properties. The problem is that even if the type is compilable, creating the object will fail TypeScript's validation since the value is not assignable to the

type of the index signature. Hence, we are at an impasse.

```
interface MyStringDictionaryWithMembers {
  [key: string]: string;
  m1: string;
}

interface MySecondInterface {
  m2: number;
}

type Intersect = MyStringDictionaryWithMembers & MySecondInterface;

// let m: Intersect = {
//   m1: "m1 value",
//   m2: 2,
//   ["AString"]: "AnotherString"
// };
```