# JavaScript Mapping for Dictionaries

Previous Next

Here is the definition of our EmployeeMap once more:

**Slice**

```
dictionary<int, Employee> EmployeeMap;
```

In the JavaScript mapping, dictionaries using a JavaScript built-in type as the key type are mapped to the JavaScript Map type. This is true for all Slice built-in types except `long`:

**JavaScript**

```
let em = new Map();

let e = new Employee();
e.number = 31;
e.firstName = "James";
e.lastName = "Gosling";

em.set(e.number, e);
```

For cases where the key type does not correspond with a JavaScript built-in type, the dictionary is mapped to HashMap. This is true for Slice dictionaries where the key type is `long` or a Slice structure that qualifies as a legal dictionary key:

**Slice**

```
dictionary<long, Employee> EmployeeMap;
```

In these cases an extra constructor is generated that initializes the HashMap with the desired comparison operator.

**JavaScript**

```
let em = new EmployeeMap();

let e = new Employee();
e.number = new Ice.Long(31);
e.firstName = "James";
e.lastName = "Gosling";

em.set(e.number, e);
```

HashMap supports the same API as the standard JavaScript Map object. It provides the following additional properties and functions:

- `HashMap(keyComparator, valueComparator)`
  This version of the constructor accepts optional comparator functions that the map uses to compare keys and values for equality. If you instantiate a map directly using `new Ice.HashMap()` without specifying comparator functions, the default comparators use the `===` operator to compare keys and values. As an example, the following map compares its keys and values using `equals` methods:

  **JavaScript**

  ```
  function compareEquals(a, b)
  {
      return a.equals(b);
  }
  var m = new Ice.HashMap(compareEquals, compareEquals);
  ```

The `valueComparator` function is only used when comparing two maps for equality.

> The type-specific constructor generated for a Slice dictionary supplies comparator functions appropriate for its key and value types.

- `equals(other, valueComparator)`
  Returns true if this map compares equal to the given map, false otherwise. You can optionally supply a function for `valueComparator` that the map uses when comparing values; this function takes precedence over the comparator supplied to the map's constructor.
- `clone()`
  Returns a shallow copy of the map.

Legal key types for `HashMap` include JavaScript's primitive types along with `null`, `NaN`, and any object that defines a `hashCode` method. The generated code for a Slice structure that qualifies as a legal dictionary key type includes a `hashCode` method. Suppose we define another dictionary type:

**Slice**

```
dictionary<Employee, string> EmployeeDeptMap;
```

The Slice compiler generates a constructor function equivalent to the following code:

**JavaScript**

```
class EmployeeDeptMap extends Ice.HashMap
{
    constructor(h)
    {
        let keyComparator = ...;
        let valueComparator = ...;
        super(h || keyComparator, valueComparator);
    }
}
```

Since the key is a user-defined structure type, the map requires a comparator that properly compares keys. Instantiating a map using `new EmployeeDeptMap` automatically sets the comparators, whereas calling `new Ice.HashMap` in this case would require you to supply your own comparators.

> Slice dictionaries that map to a HashMap must be instantiated using the generated constructor.

Notes

- Attempting to use the `map[key] = value` syntax to add an element to the map will not have the desired effect; you must use the `set` function instead.

Back to Top ^

## See Also

- Dictionaries
- JavaScript Mapping for Enumerations
- JavaScript Mapping for Structures
- JavaScript Mapping for Sequences

Previous Next