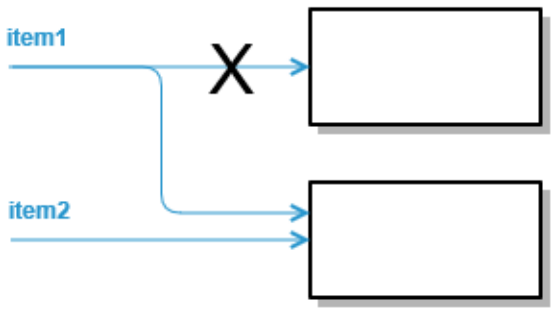
Object References

On this Page:

Object References

The E2E Bridge uses references for all objects, for objects of Base Type (like **String**, **Integer**, ...) as well as for objects of complex type. If you have two objects of same type, for example `item1` and `item2`, you can write `set item1 = item2`.



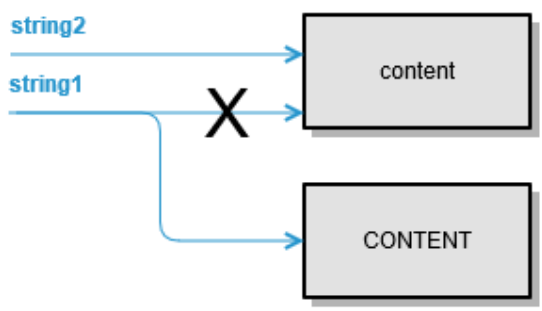
`item1` gets assigned the same reference as `item2` and now they are both pointing to the same object. This means, if you later on modify any attribute values of `item1`, object `item2` will also have these changed values.

Using Base Type Methods

What happens, if you use a Base Type method, like e.g. `toUpper()` for type **String**, to change an object. You can write

```
set string1 = string2;
set string1 = string1.toUpper();
```

Will the second statement also change `string2`? No, and this is why:



`string1.toUpper()` will create a new object containing the result of the function call and `string1` will point to the new object.

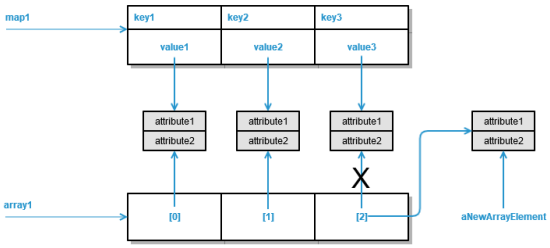
Arrays

The same applies to more advanced types like arrays and maps. If you have two arrays `array1` and `array2`, you can write `set array 1 = array2`.

`array1` is now referencing the same elements as `array2` and adding elements or changes to elements of one of them will change the other array, too.

Arrays and Maps

If you have an array `array1` and execute a `buildMap()` operation on this array, you will have the following situation:

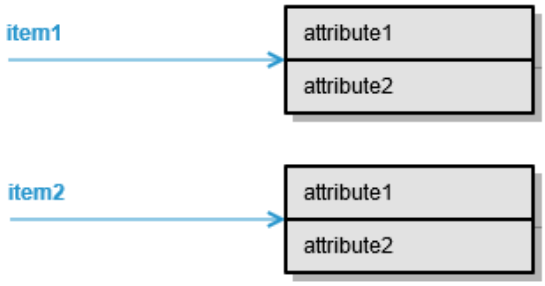


The map values are referencing the array elements. This has the following effects:

- Changing the content of an existing array element will **change** the corresponding map value content, e.g.
  `set array1[0].attribute2 = ...`
- Changing the reference to an existing array element will **not change** the corresponding map value, e.g.
  `set array1[2] = aNewArrayElement;`
- Changing an existing map value content will **change** the corresponding array element content, e.g.
  `newValue = map1.getMapValue(key1);`
  `set newValue.attribute2 = ...`
- Appending elements to the array will **not affect** the map and vice versa.

Independent Objects

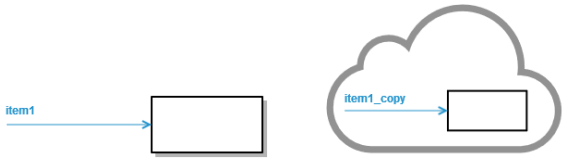
If you want truly independent objects, use the `copy()` operation. Contrary to referencing objects, if you write `set item2 = item1.`**`copy()`**, changes in the state of `item1` will not effect `item2.`



This will also duplicate used memory space, however.

Memory Adapter

The Memory adapter is an exception to this in such a way that all objects stored to memory are duplicated (stored as a copy), and also all objects that are retrieved from memory are stored in a copy.



Changes to `item1` will not change `item1_copy` and vice versa.