Array Operations

The following operations in this section are array operations. They are used to manipulate and access arrays:

Operation	Description
`[ ]`	The bracket operator can be used to get and set array elements.
`append Statement`	appends elements to arrays.
`apply Statement`	applies E2E Action Language operations to array elements.
`buildMap()`	builds a map from an array.
`concatArrays()`	concatenates the elements of one or more arrays to a target array.
`count()`	counts the number of elements in an array.
`getMapEntries()`	gets all map entries to an array (see chapter Map Operations).
`join()`	concatenates all strings of an array of strings and separates them a separator.
`reduce Statement`	reduces array valued types to scalars.
`select Statement`	filters array items by evaluating a boolean expression for each array element using an SQL like syntax.
`sort Statement`	sorts array elements in defined order.

All Any type operations listed on Any Type Operations also apply to arrays.

The following pages describe the above operations in more detail. More information about arrays can be found in section Base Types.

Arrays containing array elements of type **Array** are not supported by the Bridge.

On this Page:

Creating Arrays

You can create arrays using the `create` statement (see action script example below):

```
create anArray;
append "Hello World!" to anArray;
```

However, most of the time the E2E Runtime will create the array implicitly on appending the first item. There is one exception to this rule, though: Arrays that contain array elements having a complex type with multiplicity.
Let's assume you have an array of complex type **ArrayElement** and this complex type has a property `subArray` with multiplicity 0..*.

- What you can do, if `subArray` is `NULL`:

  ```
  set array1[0].subArray = anotherArray;
  ```

  The reference `subArray` is changed to point to `anotherArray`.
- What you can't do, if `subArray` is `NULL`:

```
append "something" to array1[0].subArray;
```

In this case (get statement on the right side of a statement), the Runtime will throw a get error for `array1[0].subArray`.

Some Array How-tos

Find below a list of some useful code snippets in array context:

Task	Code Snippet	Descriptions
empty array	`set array = select each from array where false;`	A new array is created containing all references to elements of `array` that correspond to the condition (which are none). The reference of `array` is changed to point to the new array. `array` is empty now. You can append items using the append statement.
	`set array = NULL;`	The reference of `array` is changed to point to nowhere. In the UML model, you will have now a non-existent array. You can append items using the append Statement statement, the array will implicitly be created, then.
copy all content from array1 to array2	`set array2 = select each from array1 where true;`	A new array is created containing all references to elements of `array1` that correspond to the condition (which are all). The reference `array2` is changed to point to the new array. In the UML model you will now see two arrays `array1` and `array2` that contain the same element references. A change to `array1` itself will not change `array2` (e.g. appending elements), but a change to an element of `array1` will change that very same element in `array2`.
	`set array2 = array1.copy();`	A new array is created that contains a true deep copy of all elements of `array1`. The reference `array2` is changed to point to the new array. In the UML model you will now see two arrays `array1` and `array2` that are completely independent. A change to `array1` in any way will not change `array2`.
append all content from array1 to array2	`set array2 = array2.concatArrays (array1)`	A new array is created containing all references to elements of `array1` and `array2`. The reference `array2` is changed to point to the new array. In the UML model you will now see an array `array2` containing all element references of `array1` and `array2`. A change to `array1` itself will not change `array2` (e.g. appending elements), but a change to an element of `array1` will change that very same element in `array2`.