reduce

Syntax	`reduce anArray using <expression with element and nextElement>` `    if single use <expression with element>`	
Semantics	`reduce` allows you to reduce an array having elements of base or complex types to a scalar value by recursively applying an expression to each array element and its next element. `if single use` is a mandatory extension and allows to define the action for an array containing only one single element. If the array exists, but has no array elements, an exception is raised: "The `first item of Reduce is NULL`".	
Substitutables	`anArray`	Can be any variable or object attribute having the type **Array**.
	`<expression with element and next element>`	This expression must use the currently evaluated array element and its next neighbor and evaluates to a scalar having the same type as the result value.
	`<expression with element>`	The expression after `if single use` uses the sole array element and evaluates to a scalar having the same type as the result value.

Examples	`// concatenating all strings in a list separated by a space`
	`set value = reduce myList using concat(element, " ", nextElement)`
	`            if single use element;`
	
	`// reducing complex types, calculation a number of characters`
	`set outLength = reduce inArray`
	`                using element.stringAttr.stringLength()`
	`                      + nextElement.stringAttr.stringLength()`
	`                if single use element.stringAttr.stringLength();`
	
	`// calculating the sum of all list elements`
	`set sum = reduce myNumbers using element + nextElement`
	`          if single use element;`
	
	`// concatenating blobs`
	`set reducedBlob = reduce blobs using concat(element, nextElement)`
	`                  if single use element;`
	
	`// if single use`
	`set reduceString = reduce stringArray`
	`                    using concat("many elements: ", element, " ", nextElement)`
	`                    if single use concat("only one element: ", element);`	

On this Page:

Example File (Builder project E2E Action Language/Array):

<your example path>\E2E Action Language\Array\uml\arrayReduce.xml

Reducing Arrays with Elements of Base Type

For example, in an array of strings, you can reduce the list to a simple string consisting of a concatenation of all single elements. Suppose you have an array of **Strings** and you wish to copy all of them to a single **String**, separated by a space. The solution is to use operation `reduce` in combination with using `concat(element, " ", nextElement)`. Note that the second parameter of the `concat` operation is the string that should be used as separator between any given array element and its next neighbor. In this case, this is a space string literal.

The example below will concatenate all elements of a string array together with "space slash space" as a separator between any two elements.

```
set value = reduce myList using concat(element, " / ", nextElement) if
single use element;
```

`element` and `nextElement` are keywords and allow you to use relative references (as opposed to normal, absolute indexes) to single elements of an array (see Get Array Element Operator [ ]).

Based on the type of the array elements, you may use different operations. For example, elements of base type **Integer** can be computed (addition, subtraction, division, or multiplication).

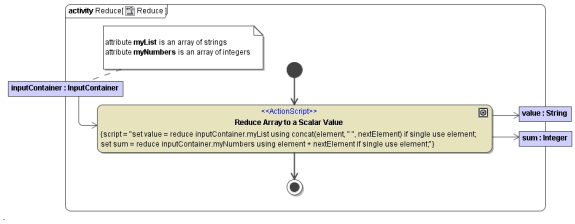
```
set sum = reduce myNumbers using element + nextElement if single use
element;
```

Reduce Algorithm

The `reduce` algorithm works as follows. Suppose you want to reduce the array **`myNumbers`** having the numbers **`1`**, **`2`**, and **`3`**. In the first iteration, the first occurrence of the element expression is initialized (here: take the first integer **`1`**), then the next element is evaluated and added (`+ nextElement`). The result of the first iteration is **`3`**. In the second iteration, element refers to the intermediate result of the first iteration. Then, `nextElement` is added again, resulting in value **`6`**. This value is finally assigned to integer **`sum`**.

The following activity diagram shows how to reduce strings and integers to a scalar.

Figure: Reducing Arrays with Elements of Base Type

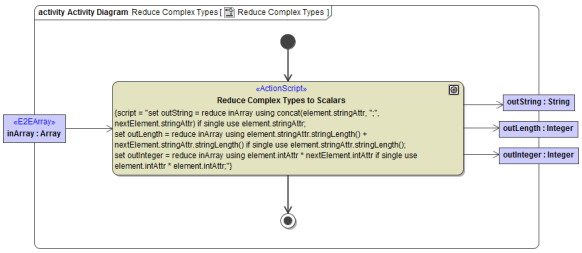


Reducing Arrays with Elements of Complex Type

The same algorithm applies to arrays with elements of complex types as described in the previous chapter. Since each `element` or `nextElement` refers to an object of complex type, you can access the attributes of the object like any other instance (e.g. `element.name`).
In the example below, array **inArray** contains objects of complex type **ReduceStructure**. Now, you wish to copy attribute **strAttr** of each array element to a single string separated by a semicolon using the operation reduce in combination with using `concat(element, ";", nextElement)`.

```
set outString = reduce inArray using concat(element.stringAttr, ";",
nextElement.stringAttr) if single use element.stringAttr;
```

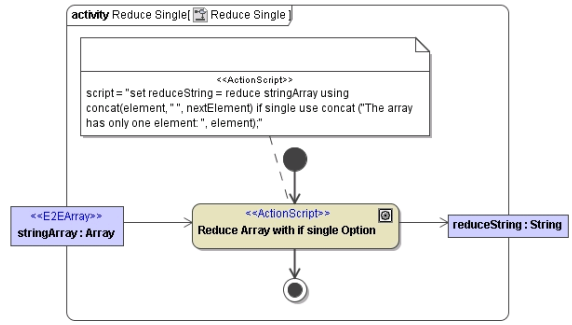
Figure: Reducing Arrays with Elements of Complex Type



Single Array Elements

If an array contains only one single element, you can define what will be assigned to the scalar with the mandatory extension `if single use` in the `reduce` statement.

Figure: Reducing an Array Having a Single Element



If an array has only one single element, the statement following `if single use` will be evaluated. In the example above, element refers to the only string in the array. If the string is **`Hello World!`**, the string **`The array has only one element: Hello World!`** will be assigned to the output string **`reduceString`**.

In the following example, the square of an integer contained in the array will be returned, if the integer is the only element in the array. Otherwise, all integers of the array will be multiplied with each other and assigned to **`outInteger`**.

```
set outInteger = reduce inArray using element.intAttr * nextElement.
intAttr if single use element.intAttr * element.intAttr;
```