Objects

Each service uses object nodes to pass data not only to and from clients, but also internally. Inside the service, you will want to read / change data which is stored in such data items. In order to do this, you must first understand that there are four different kinds of objects, as listed in the table below.

Object Type	Graphical Representation	Description
Base Types	outputElement : String	The data is not a user defined data structure but a simple String type named `outputElement`.
Array of Base Types	outputArray : Array {arrayElement = String}	The data is an array of elements called `outputArray`. The type of the array elements is set via the Tagged Value `arrayElement`. The output array consists of 0 to n String elements.
Complex Type	inputItem : DataItem	The input/output is a data structure (we call them "data items"). Refer to Basic Structural Modeling for more information on data structures.
Complex Type Array	outputArray : Array {arrayElement = DataItem}	The input/output is an array of data items named o `utputArray`. The output array consists of 0 to n DataItem elements.

You need to employ different techniques to access the types of data items depicted above. Please understand that accessing a "base type" element is not the same as accessing the "foo" attribute of the third element of the "bar" array.

The following examples show how to access all types of data structures in action scripts.

Example File (Builder project Basic Modeling/Data):

	<your example path>\Basic Modeling\Data\uml\objectNavigation.xml

On this Page:

Related Pages:

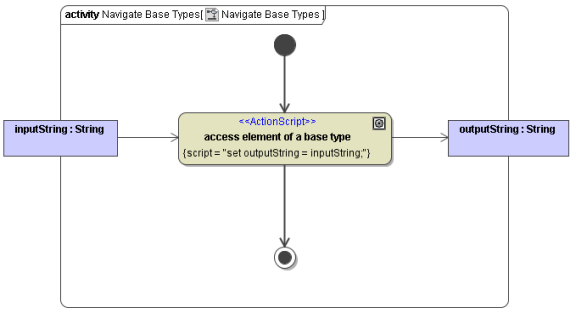
- Basic Structural Modeling
- Array Operations
  - Get Array Element Operator [ ]

Handling Base Type Objects

Input and output are Base Types. Assignments are made by referencing the names of the base types.

The `set` statement is defined in the action script of the action.

Figure: Access of Base Types



Handling Arrays of Base Type Objects

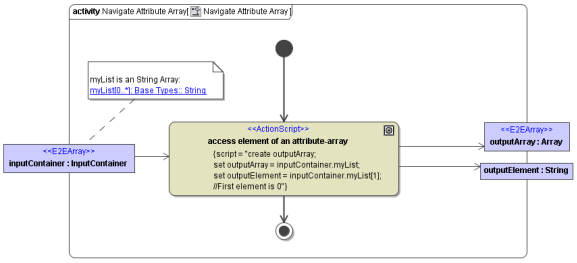
The figure *Access of Base Type Arrays* below shows how to access elements of arrays of base types. It also displays how to access the array itself.
InputContainer is a data item that contains an array of type String. This array is named **`myList`** and can have 0 to n entries.

> Array indices start at **0**.

There are two output items: `outputElement` which is a base type object and `outputArray` which is of type array. To specify which type the array elements of `outputArray` have, the tagged value `arrayElement` must be used.
To reference the whole `myList` array to the output **String** array, the following statement is used:

```
set outputArray = inputContainer.myList;
```

Figure: Access of Base Type Arrays



To access a single `myList` element the `[ ]` notation is used:

```
set outputElement = inputContainer.myList[0];
```

This statement copies the first element of **`myList`** into **`outputElement`**.

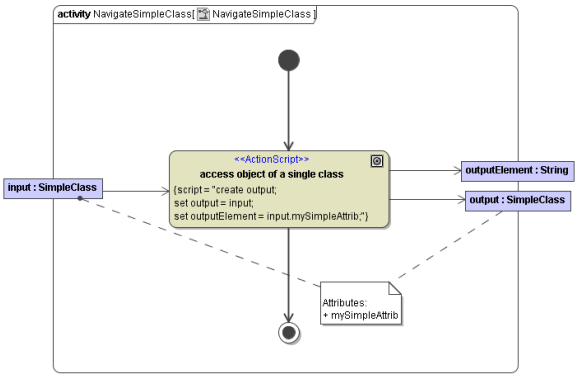
> Objects are strong typed. This means that you cannot copy an integer to a string. In the example above, **`outputElement`** must be of the same type as **`myList[0]`** .

If you want to work with an input parameter as a variable to define witch element of the array should be copied to the `outputElement`, take a look at **Get Array Element Operator [ ]** .

Handling Objects of Complex Type

The figure below shows an example of accessing a data item as a whole part and then accessing a single attribute of it.
To access a single attribute the notation `<objectName>.<attributeName>` is used.

Figure: Accessing single item elements



Handling Arrays of Complex Type Objects

To access the array itself, the object name is used.
To access a single element, in this example the data item `singleClass`, use the syntax: `arrayName [index] (set singleClass=inputArray[3])`
Use `objectName[index].AttributeName` to access a single attribute of the specified element.

Figure: Accessing elements of a data item array