Creating Objects of Base and Complex Types

The most basic features of each language are creating instances of classes and setting values.

- With the `set` assignment statement, it is possible to create **objects of base types** like **String**, **Integer**, **Float**, **Boolean**, and **DateTime**
- whereas **objects of complex types** need to be instantiated (created) first with the `create` statement - that is creating an instance of the corresponding class. Once the instance of the class (the object) has been created, it can store data in its attributes.

On this Page:

Related Pages:

Basics of the Action Script Language:

Creating Base Types with the set Assignment Statement

The following examples shows how to use the set assignment statement to create base type objects.

Syntax	`set anObject = aValue;`	
Semantics	Assigns a value to `anObject`.	
Substitutables	`anObject`	Can be an object node or an attribute, or an association end.
	`aValue`	Can be a literal, a object node of base type, or an action script operation or expression returning a base type.

Examples	`set aString = "Hello World!";`
	`set anInteger = 12345;`
	`set currentDate = currentDateTime();`	

The example in the figure below shows how to create base type objects like strings, integers, etc. Base type objects need not to be instantiated, they are created by using the `set` assignment statement to directly assign a value to the variable.
All variables are drawn as object nodes with all objects being of base type.

Creating Arrays by Appending Items

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

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

Most of the time the xUML 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`.

Creating Objects of Complex Type

The following example shows how to use the `create` statement to create objects of complex type.

> Scalar base type objects are never created using the `create` statement (see Creating Base Types with the set Assignment Statement above).

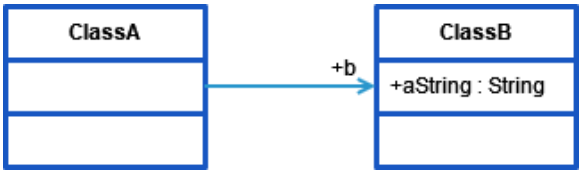
Syntax	`create anObject;`	
Semantics	Creates an object of complex type. The object reference is stored in `anObject`. Initial values defined on the class attributes will be set.	
Substitutables	`anObject`	Can be any valid object name.
Examples	`create simpleObject;`	

In the following cases, objects need to be created with the `create` statement:

- You want to explicitly create an instance of a class.
- Result objects of an iteration over an action script need to be created.

In the following cases, the `create` statement is not necessary:

- Adapters may create objects of any type. Those objects are implicitly created by the adapter and do not need to be created explicitly, e.g. the result set of an SQL query.
- When having related classes, the instantiation of intermediate objects is not mandatory.
  In the example below, **ClassA** has an attribute **b**, which is of type **ClassB** (see association). **ClassB** in turn has a string attribute named **aString**.



When creating an instance of the top-level class **ClassA**, the lower classes are instantiated implicitly, so the action script below would be fine.

```
create objectOfClassA;
set objectOfClassA.b.aString ="Hello World!";
```