JavaScript Specifics

Using Base Type Objects

When using the JavaScript adapter, E2E Bridge Base Types (excluding **Any**) will automatically be converted to JavaScript types. If a conversion is not possible, an exception will be thrown (see section Exception Handling).

Use the following syntax to create a JavaScript object:

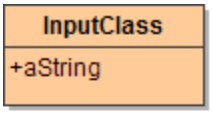
```
var myString = aString;
```

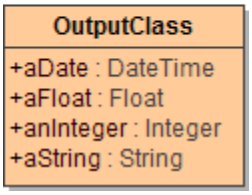
The table shows the E2E Bridge base types and the corresponding JavaScript types after conversion using the JavaScript adapter:

E2E Base Type	JavaScript Type	Additional Information
Array	**Array**	Nested/multidimensional arrays are not supported. Assigning arrays between JavaScript and the Bridge works both ways, but mentioned limitations are still applicable.
Blob		The support of type **Blob** is limited: Blobs cannot be created, but assigned with another blob. Blobs have a **toString()** method, which can be called internally to convert blob to string.
Boolean	**boolean**	
DateTime	**Date**	
Float	**number**	
Integer	**number**	
String	**string**	

Example: Conversion of E2E Bridge simple types to JavaScript types

The basis for the conversion example are the following classes:





The input parameters contain E2E Bridge base types, the output provides the corresponding JavaScript types:

Input parameter name	Input parameter value	JavaScript Code	Output parameter name	Output parameter value

aString	`{aString: "Hello Bridge!"}`	`var myString = aString + anObject.aString;`
		
		`output = {`
		`"aString": myString,`
		
		`"anInteger": anInteger,`
		`"aFloat": aFloat,`
		`"aDate": aDateTime`
		`};`
			output	`{aString: "HelloBridge! Hello world!", aDate: "2017-02-28", anInteger: "3", aFloat: "2.1"}`

anInteger	`{anInteger: "3"}`			
aFloat	`{aFloat: "2.1"}`			
aDateTime	`{aDateTime: "2017-02-28"}`			
anObject	`{aString: "Hello world!"}`			

Using Objects of Complex Type

With the JavaScript adapter, you can create objects of complex type by calling its constructor which is already included in JavaScript. The constructor is invoked with the operator `new`. The constructor name is created by simply replacing the dots of the full object type (with all packages) with underscores. The full object types prefix is deleted.

Example:

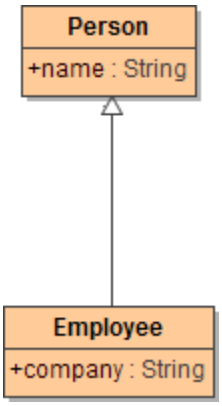
Full Object Type	urn:Services.MyTypes.CustomerAddress
Constructor Name	Services_MyTypes_CustomerAddress
Constructor Call	`var myObject = new Services_MyTypes_CustomerAddress();`

Please note: Using underscores in your full object types names may result in conflicts with the constructors name. For example, the constructors name of the two full object types *urn:Services.MyTypes.CustomerAddress* and *urn:Services_MyTypes.CustomerAddress* would be the same. In such cases the behavior is undefined.

JavaScript is dynamic while the action script used in the Bridge is well-regulated. When mixing Bridge objects of complex type with JavaScript objects, the following rules apply:

- Objects of complex type can be treated like any other JavaScript object.
- If JavaScript objects are assigned to Bridge complex types, only the fields defined in the complex type meta data are set.
- As JavaScript is dynamic, additional fields to objects of complex type can be set using the dot notation, but those fields will vanish when the script finishes. They exist only in this instance of the JavaScript adapter and cannot be returned to the Bridge.
- Polymorphism is not implemented. The assignment of different object types will be treated like an assignment between an object of complex type and a JavaScript object. The relation between parent and child classes is lost in the JavaScript adapter.

Example: Loss of parent-child relation



Input parameter name	Input parameter value	JavaScript Code	Output parameter name	Output parameter value
person	`{name: "John Doe"}`	`person = employee;`	**person**	`{name: "James Smith"}`
employee	`{name: "James Smith", company: "E2E"}`			

Parsing JSON

`JSON.parse` is a JavaScript construct you can use within the JavaScript adapter.

Syntax	`out = JSON.parse(json);`	
Semantics	Convert a JSON string into an object. The operation returns an object. An exception will be thrown if **json** is no valid JSON string.	
Substitutables	out	An object.
	json	Any JSON string.
Example	`myObject = JSON.parse(jsonString);`	

Exception Handling

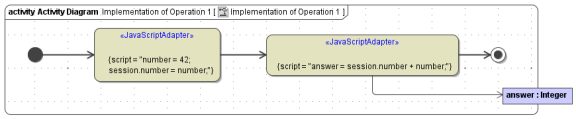
Thrown exceptions will be propagate d outside the adapter as standard Bridge exceptions. All JavaScript exceptions will be reported to the Bridge as **SCRIPTSM/1**. They can be caught using the JavaScript statement **try{} catch(){}**. You can throw a user-defined exception using the `throw` statement.

Data Exchange between JavaScript Adapter Calls

The global variable **session** can be used to store JavaScript objects between adapter calls. The variable exists for the duration of the session (one request to the Bridge). Its purpose is to store some state between adapter calls.

*Figure: Storing JavaScript objects in variable **session***

session.number is propagated to the second adapter call, but **number** is not. The value of **answer** will be 42.



Using Maps

As in Action Script (see section Map Operations), you can also use maps in JavaScript. You can build /create a map, add new elements to a map, retrieve and remove items from a map and get map entries. Maps can also be used as input/output parameters to the JavaScript Adapter.

Operation	Action Script	JavaScript	Additional Information
Creating a new map	`create myMap;`	`var myMap = new Base_Components_Basic_Behavior_Map_Map();`	
Adding a new element to a map	`set anObject = myMap.setMapValue ("my key", "my value");`	`myMap.setEntry('my key', 'my value');`	All simple types except **blob** can be used as key. All values except arrays and anonymous objects are allowed. (see section setMapValue() Operation)
Retrieving an item from a map	`set anObject = myMap.getMapValue ("my key");`	`var value = myMap.getEntry ('my key');`	(see section getMapValue() Operation)
Removing an item from a map	`set deletedObject = myMap. removeMapValue("my key");`	`var removed = myMap. removeEntry('my key');`	(see section removeMapValue() Operation)
Getting map entries	`set anArray = myMap. getMapEntries();`	`var mapEntries = myMap. getEntries(); var entryValue = mapEntries[0].value;`	(see section getMapEntries() Operation)

Building a map	`set myMap = myArray.buildMap ("name");`	**Building a map with myArray as input parameter**
		`var myMap = new Base_Components_Basic_Behavior_Map_Map (myArray, "name");`
		
		Building a map with creation of myArray
		`var r1 = new Base_Components_Basic_Behavior_MIME_MIMEHeaderField(); r1.name = 'key1'; r1.value = 'value1'; var r2 = new Base_Components_Basic_Behavior_MIME_MIMEHeaderField(); r2.name = 'key2'; r2.value = 'value2'; var myArray = new Array(); myArray.push(r1); myArray.push(r2);`
		
		`var myMap = new Base_Components_Basic_Behavior_Map_Map (myArray, "name");`
			(see section buildMap() Operation)