Elektrobit

EB GUIDE Script reference card

FEATURE	DESCRIPTION	EXAMPLE

NAMESPACES	You have to prefix model elements when referring to them.
	The following prefixes exist: **dp:** for global properties, **ev:** for events, **v:** for local variables, **f:** for functions
		dp:x = 100; // set a global property
		fire ev:back(); // fire an event
		f:trace_string("hello world"); // call a function

ACCESSING GLOBAL PROPERTIES	Write a global property by placing it at the left side of an assignment. Read a global property by using it anywhere else in an expression. The redirect reference (**=>**) is a special form of global property assignment.	dp:x = 5; // writing to x
		dp:x = dp:y + dp:z; // reading y and z
		length dp:aList; // read the length of a list property
		dp:refX => dp:x; // redirect

SENDING EVENTS	Syntax:
	fire ev:<identifier>**(**<parameter-list>**);**
	
	Events can be fired after a timeout.
	This delayed event can be canceled with the **cancel_fire** expression.
	
	Syntax:
	fire_delayed <timeout>, **ev:**<identifier>**(**<parameter-list>**);**
	cancel_fire ev:<identifier>**;**
		fire ev:back();
		fire ev:mouseClick(10, 20);
		
		fire_delayed 3000, ev:back(); // send the event "back" in 3 seconds.
		cancel_fire ev:back; // cancel the event

REACTING ON EVENTS	To react on events, use **match_event**. This is a special form of the **if-then-else** statement. **If** and **else** branch must have the same type. If used at the right side of an assignment, the **else** branch is mandatory.
	
	Syntax:
	match_event v:<identifier> = **ev:**<identifier>
	in <sequence>
	else <sequence>
		match_event v:event = ev:back in {
		    f:trace_string("back event received");
		}
		
		v:this.x = match_event v:event = ev:back in 10 else 0;

ACCESSING EVENT PARAMETERS	The **in** expression of a **match_event** has access to the event parameters.
	Use the dot notation to access event parameters.
		match_event v:event = ev:mouseClick in {
		    v:this.x = v:event.x;
		    v:this.y = v:event.y;
		}

ACCESSING PRIVATE WIDGET PROPERTIES	If a script is part of a widget (widget actions, input reactions), it has access to the properties of that widget. A special local variable called **v:this** is available referring to the current widget. Use the dot notation to address private properties.	v:this.text = "hello world";
		v:this.x = 10;

NAVIGATING THE WIDGET TREE	If a script is part of a widget, it has access to the properties of other widgets. Use the widget tree navigation operator: **->**. To access the parent widget, use the identifier: **^**.	v:this->^->caption.text = "Play";
		  // goto parent, goto caption, property text
		v:this->^.x-= 1; // goto parent, property x

STRING FORMATTING	The **+** operator concatenates strings. For more string conversion functions, refer to the documentation.	v:this.text = "current speed: " + f:int2string(dp:speed) + "km/h";

EB GUIDE Script reference card

FEATURE	DESCRIPTION	EXAMPLE
CONSTANTS	String constants may be written without quotes. Color constants are RGBA quadruples.	"hello world" // string constant Napoleon // string constant 5 // integer constant color:0,235,0,255 // EB green
ARITHMETIC, LOGIC AND ASSIGNMENT OPERATORS	Addition and string concatenation: **+**, subtraction: **-**, multiplication: *, division: **/**, modulo: **%**, greater-than: **>**, less-than: **<**, greater-or-equal: **>=**, less-or-equal: **<=**, equal: **==**, not-equal: **!=**, and: **&&**, or: **||**, not: **!**, assignment: **=**, assign-increment: **+=**, assign-decrement: **-=**	dp:myString = "Hello" + "World"; dp:count += 1; // increment one

SEQUENCING	A sequence is either a single expression or a series of expressions enclosed in curly braces. The last expression in a sequence is the value of the sequence.	if(dp:something)
		  dp:x = 5; // single expression
		if(dp:other) {
		  dp:x = 5; // sequence enclosed
		  dp:y = 10; // in curly braces
		}

LOCAL VARIABLES	Use **let** bindings to introduce local variables. It is not allowed to use uninitialized variables.
	let bindings may be nested.
	
	Syntax:
	let v:<identifier> **=** <expression>**;**
	  v:<identifier2> **=** <expression>**;**
	  ...
	  in <sequence>
		let v:x = 42;
		  v:text = "hello world";
		in {
		  v:this.x = v:x;
		  v:this.text = v:text;
		}

WHILE LOOP	The **while** loop consists of two expressions: the condition and the body. The body is repeatedly evaluated until the condition yields false.
	
	Syntax:
	while(<expression> **)** <sequence>
		dp:i = 0;
		while(dp:i <= 10) {
		  dp:sum += i;
		  dp:i += 1;
		}

IF-THEN-ELSE	**If-then-else** behaves like the ternary conditional operator in C and Java. If it is used at the right side of an assignment, the **else** branch is mandatory and both branches must have the same type.
	
	Syntax:
	if(<expression> **)** <sequence> **else** <sequence>
		if(dp:buttonClicked) {
		  v:this.x = dp:x;
		}
		else {
		  v:this.x = 0;
		}
		
		v:this.x = if(dp:buttonClicked) dp:x else 0;

COMMENTS	C style block comments and C++ style line comments are allowed.	/* this is a C style block comment */ // this is a C++ style line comment
RETURN VALUE	The last expression in a script is the return value. To force a return value of type void, use **unit** or **{}**.	dp:x + 2; // returns datapool entry x plus 2