

FEATURE	DESCRIPTION	EXAMPLE
<ul style="list-style-type: none"> Namespaces 	You have to prefix model elements when referring to them. The following prefixes exist: dp : for datapool items, ev : for events, v : for local variables, f : for functions	<pre>dp:x = 100; // set a datapool item fire ev:back(); // fire an event f:trace_string("hello world"); // call a function</pre>
<ul style="list-style-type: none"> Accessing datapool items 	Write a datapool item by placing it at the left side of an assignment. Read a datapool item by using it anywhere else in an expression. The redirect-link (=>) is a special form of datapool item assignment.	<pre>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 datapool item dp:refX => dp:x; // redirect link</pre>
<ul style="list-style-type: none"> Sending events 	Syntax: fire ev :<identifier>(<parameter-list>); Events can be sent 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>;	<pre>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</pre>
<ul style="list-style-type: none"> 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 always 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>	<pre>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;</pre>
<ul style="list-style-type: none"> Accessing event parameters 	The in expression of a match_event has access to the event parameters. Use the dot notation to access event parameters.	<pre>match_event v:event = ev:mouseClick in { v:this.x = v:event.x; v:this.y = v:event.y; }</pre>
<ul style="list-style-type: none"> Accessing 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 widget properties.	<pre>v:this.text = "hello world"; v:this.x = 10;</pre>
<ul style="list-style-type: none"> 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: ^ .	<pre>v:this->^>caption.text = "Play"; // goto parent, goto caption, property text v:this->^>.x = 1; // goto parent, property x</pre>
<ul style="list-style-type: none"> String formatting 	The + operator concatenates strings. For more string conversion functions, please refer to the documentation.	<pre>v:this.text = "current speed: " + f:int2string(dp:speed) + "km/h";</pre>
<ul style="list-style-type: none"> String comparison 	To compare two strings with case sensitivity, use the equality operators == or != . To compare two strings without case sensitivity, use the equality operator =Aa= .	<pre>"name" == "NAME" // false "name" != "NAME" // true "name" =Aa= "NAME" // true</pre>
<ul style="list-style-type: none"> Changing language 	To change the language of all datapool items of an EB GUIDE model, use language . This operation is performed asynchronously. Syntax: f:language (!:<identifier>)	<pre>f:language(!:Standard) // changes language to the standard language f:language(!:German) // changes language to German</pre>

FEATURE	DESCRIPTION	EXAMPLE
<ul style="list-style-type: none"> Changing skin 	<p>To change the skin of all datapool items of an EB GUIDE model, use skin. This operation is performed asynchronously.</p> <p>Syntax: f:skin(s:<identifier>)</p>	<pre>f:skin(s:Standard) // changes to the standard skin f:skin(s: "myskin") // changes to a user-defined skin</pre>
<ul style="list-style-type: none"> Constants 	<p>String constants may be written without quotes. Color constants are RGBA quadruples.</p>	<pre>"hello world" // string constant Napoleon // string constant 5 // integer constant color:0,235,0,255 // EB green</pre>
<ul style="list-style-type: none"> Arithmetic, logic and assignment operators 	<p>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: -=</p>	<pre>dp.myString = "Hello" + "World"; dp.count += 1; // increment one</pre>
<ul style="list-style-type: none"> Sequencing 	<p>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.</p>	<pre>if(dp.something) dp:x = 5; // single expression if(dp.other) { dp:x = 5; // sequence enclosed dp:y = 10; // in curly braces }</pre>
<ul style="list-style-type: none"> Local variables 	<p>Use let-bindings to introduce local variables. It is not allowed to use uninitialized variables. let-bindings may be nested.</p> <p>Syntax: let v:<identifier> = <expression>; v:<identifier2> = <expression>; ... in <sequence></p>	<pre>let v:x = 42; v:text = "hello world"; in { v:this.x = v:x; v:this.text = v:text; }</pre>
<ul style="list-style-type: none"> While loop 	<p>The while loop consists of two expressions: the condition and the body. The body is repeatedly evaluated until the condition yields false.</p> <p>Syntax: while(<expression>) <sequence></p>	<pre>dp:i = 0; while(dp:i <= 10) { dp:sum += i; dp:i += 1; }</pre>
<ul style="list-style-type: none"> If-then-else 	<p>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.</p> <p>Syntax: if(<expression>) <sequence> else <sequence></p>	<pre>if(dp:buttonClicked) { v:this.x = dp:x; } else { v:this.x = 0; } v:this.x = if(dp:buttonClicked) dp:x else 0;</pre>
<ul style="list-style-type: none"> Comments 	<p>C style block comments and C++ style line comments are allowed.</p>	<pre>/* this is a C style block comment */ // this is a C++ style line comment</pre>
<ul style="list-style-type: none"> Return value 	<p>The last expression in a script is the return value. To force a return value of type void, use unit or {}</p>	<pre>dp:x + 2; // returns datapool item x plus 2</pre>