Scripting Manual

VBS4 24.1.1

Bohemia Interactive SIMULATIONS

Documentation Legal Notice

This Documentation, including any embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by Bohemia Interactive Simulations (BISim) at any time. This Documentation and its contents are proprietary information of BISim, also protected by copyright, and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of BISim.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all BISim copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to BISim that all copies and partial copies of the Documentation have been returned to BISim or destroyed.

BISim has made every reasonable effort to ensure the accuracy of all the information contained in the Documentation. However, product specifications are subject to change without notice, and BISim makes no representations or warranties regarding the accuracy, completeness, or suitability of information contained in the Documentation. To the maximum extent permitted by law, BISim disclaims any and all liability for any loss, damage (direct or indirect) or other consequence which may arise from the use of or reliance upon any information contained in the Documentation.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

Customer Support

The Bohemia Interactive Simulations Support page can be found at:

- http://www.bisimulations.com/support

For any type of assistance with Bohemia Interactive Simulations products, use the following support email and we will respond to your query with urgency.

- support@bisimulations.com

Our website contains a range of media and handouts relating to Bohemia Interactive Simulations products:

- http://www.bisimulations.com/

The BISim Wiki is the primary resource on VBS4 scripting:

https://sqf.bisimulations.com/display/SQF/VBS+Scripting+Reference

PhysX

VBS4 uses the PhysX physics engine. For more information on PhysX visit the Nvidia site.

https://gameworksdocs.nvidia.com/simulation.html

Contents

1. Scripting Overview

During mission editing, you can come across situations where actions or features you would like to have in your mission cannot be included using VBS Editor in the Prepare Mode. For example, when you want to create custom UIs and / or custom and more complex AI behaviors. The solution is to take advantage of the simulation engine ability to use a more advanced feature known as scripting.

The VBS4 scripting language is called SQF. It allows you to control and influence objects and the simulation environment with over 2,000 commands. The ability to create objects and perform tasks not available in VBS Editor allows trainers to enhance the training.

The simulation engine controls the environment and how objects react in the environment. A scripting command is an instruction that tells the VBS4 simulation engine what to do. SQF is a proprietary language similar to C++.

This manual is divided into three main parts, followed by tutorials and a scripting command and function reference. The three main parts are:

- Basic Scripting (on the next page)
- Intermediate Scripting (on page 40)
- Advanced Scripting (on page 152)

To demonstrate different aspects of VBS scripting, from basic to advanced, various demo missions have been prepared:

- Scripting Tutorials (on page 210)

The manual is concluded with scripting several scripting references topics, and information about the Bohemia Interactive Simulations Wiki:

- Scripting Reference (on page 259)

2. Basic Scripting

A basic script can be added to any object in VBS4. Before starting to script in VBS4, you have to become familiar with SQF script syntax, and how it its related to anything you would like to program through scripting. Next, all the possible places - and their differences - where a basic script can be created / inserted are described. Nearly all scripting commands in VBS4 (including the most basic ones which are Operators (on page 28)) use variables / parameters, which rely on data types. When you start scripting, you encounter many errors - that is normal. This chapter also covers the most common errors a beginner script developer runs into. The chapter is concluded by describing where longer (having more lines of code) and more complex scripting takes place.

- Basic Script Syntax (on the next page) - The general structure of all SQF scripts.
- Using Basic Scripts (on page 15) - All the places where a basic script can be created / inserted.
- Basic Data Types (on page 20) - An introduction to what data types are and the essential data types used in SQF.
- Basic Variables / Parameters (on page 24) - To pass data around in VBS scripting, variables (or parameters) are used. Each variable / parameter has its own data type.
- Operators (on page 28) - To work with variables, operators are often needed (for example, arithmetic operators). Operators are the most basic commands in SQF.

The chapter is concluded with common scripting errors which may appear in an incorrectly written script, and `init.sqf` - See:

- Error Messages (on page 33) - The common error messages that appear in SQF scripting.
- Init.sqf (on page 37) - The place where longer and more complex scripts can be inserted for execution on mission initialization, prior to mission start.
- Exercises (on page 38) - Five simple exercises that repeat the concepts explained in the Basic Scripting chapter.

2.1 Basic Script Syntax

In order for a script to work, the command and correct parameters must be used in the correct order.

A script statement can have three syntax variations - the order of script elements - recognized by VBS4:

Syntax Variation	Description
COMMAND	The command takes no parameters. **EXAMPLE** In this example, allUnits (https://sqf.bisimulations.com/display/SQF/allUnits) is a command that returns all the units in a mission: `allUnits;`

COMMAND parameter	The command takes one parameter, which can also be an Array of parameters.
	
	EXAMPLE
	In this example, the following happens:
	1. The alive (https://sqf.bisimulations.com/display/SQF/alive) command takes the name of a unit (`unit1`) as a parameter and checks if that unit is alive. In this script statement, the unit parameter is also a variable name (see Basic Variables / Parameters (on page 24)).
	2. The createDialog (https://sqf.bisimulations.com/display/SQF/createDialog) command takes the name of a dialog as a string (see String (on page 21)) to create it.
	
	`alive unit1;`
	`createDialog "mySampleDialog";`

Syntax Variation	Description
parameter1 **COMMAND** parameter2	The command acts as an operator between parameter1 and parameter2 (for example, an arithmetic operator such as +, -, /, * is also considered a command), parameter1 and / or parameter2 can be also Arrays of parameters.

NOTE

If parameter2 is an Array of parameters, some parameters inside the Array can be optional.

EXAMPLE

In this example, the following happens:

1. The plus (+) operator is also a command (see Operators (on page 28)) that takes two parameters, which are also variables, called `number1` and `number2`, and performs a summation.
2. The allowDamage (https://sqf.bisimulations.com/display/SQF/allowDamage) command takes a unit (`unit1`) as first parameter and a Boolean value of `false` as second parameter, to disable damage for that unit.
3. The moveInCargo (https://sqf.bisimulations.com/display/SQF/moveInCargo) command takes a unit (`soldier1`) as first parameter and a vehicle (`truck1`) as second parameter, and instantaneously relocates the unit into the vehicle cargo space.

```
number1 + number2;
unit1 allowDamage false;
soldier1 moveInCargo truck1;
```

NOTE

Each script statement must end with a semicolon (;) for the simulation engine to compile and execute it.

Any syntax variation has two constituents:

- **COMMAND** – A specific VBS4 word and a command that does something in the simulation. Command names reflect what they do in the simulation (for example, move a unit in a cargo position - moveInCargo (https://sqf.bisimulations.com/display/SQF/moveInCargo), set a certain amount of fuel in a vehicle - setFuel (https://sqf.bisimulations.com/display/SQF/setFuel)).

- **Variables / Parameters (parameter1, parameter2, ...)** – A single constituent needed for a scripting command to work. A variable can be a parameter for a command. Without the correct parameters, a command does not work. Each variable has its own data type (Array (on page 23), String (on page 21), Number (on page 21), and so on).

In any syntax variation, there are two general rules about using spaces:

Rule	Examples

Spaces are needed before and after scripting commands	`// correct`
	`unit1 moveInGunner vic1;`
	`// incorrect, needs a space before the command`
	`unit1moveInGunner vic1;`
	
	`// correct`
	`player sideChat "working";`
	`/* incorrect, using two commands,`
	`   needs a space in between each one`
	`*/`
	`playersideChat "working";`
	
	`// correct`
	`vic1 setDir 270;`
	`// incorrect, needs a space after the command`
	`vic1 setDir270;`

Spaces are optional before and after special characters
• Math: + - * / = < >
• Punctuation: , ; ""
• Brackets: [ ] { } ()
	Each of the examples below are acceptable:
	`var1 = direction  vic1*speed  vic1;`
	`this addMagazine "ak47_mag";`
	`if (player in vic1) then {};`

2.2 Using Basic Scripts

A basic script can be created directly in the Editor (Prepare Mode). Each object in the simulation can have script statements associated with it. Basic scripts are inserted in the **Object Properties** dialog (see description below) in the Editor.

The **Object Properties** dialog (which can have varying fields based on the type of simulation object) enables scripts in the following places:

- Initialization Statements in an Object (on the next page)
- Condition to Complete and Code on Completion in a Waypoint (on the next page)
- Condition, On Activation and On Deactivation in a Trigger (on page 17)

2.2.1 Initialization Statements in an Object

The **Initialization Statements** field of every object is the most common place to insert a basic script. Any script inserted in that field executes before the mission starts. The field can use this (on page 27).

EXAMPLE

The mission starts with `soldier1` already wounded:

Image-1: The keyword 'this' referring to 'soldier1'

2.2.2 Condition to Complete and Code on Completion in a Waypoint

Each waypoint has both **Condition to Complete** and **Code on Completion** fields to handle the current and next waypoint.

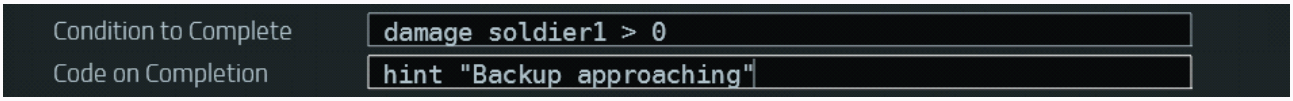
If the condition in the **Condition to Complete** field is fulfilled, the waypoint is considered to be completed.

NOTE

If the waypoint is not completed, moving to the next waypoint is not allowed.

EXAMPLE

If `soldier1` is wounded, a unit assigned with the conditioned first waypoint advances to the second waypoint next to `soldier1`:



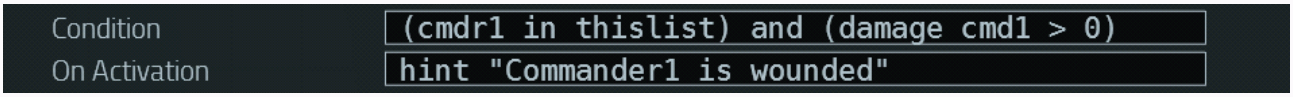
The **Code on Completion** field of a waypoint is activated when the waypoint is reached. In this example, after the condition of the first waypoint is fulfilled, the unit begins to approach the second waypoint next to `soldier1` and the message "`Backup approaching`" is displayed using the `hint` command.

2.2.3 Condition, On Activation and On Deactivation in a Trigger

Each trigger has the **Condition**, **On Activation** and **On Deactivation** fields which control what happens when the trigger is activated and deactivated, respectively.

EXAMPLE

Apart from the **Activation** and **Activation Type** fields, the **Condition** field controls under which additional conditions the trigger activates. Here is a trigger that activates by displaying the message "`Commander 1 is wounded`" only when a wounded BLUFOR soldier with the name `cmdr1` steps into the trigger area:



On Activation in triggers is the field where some script statements can be executed when the trigger is activated. For example, in the previous example the message "`Commander 1 is wounded`" is displayed using the `hint` command.

When a trigger is deactivated (after it was activated), the **On Deactivation** field can be used to execute some script statements.

NOTE

On Deactivation only exists in a trigger. There is no similar field in a waypoint.

EXAMPLE

When the trigger is activated, the message "`Wounded Commander 1 arrived`" is displayed, and when the trigger is deactivated, the message "`Wounded Commander 1 left`" is displayed.

Condition	`(cmdr1 in thislist) and (damage cmd1 > 0)`
On Activation	`hint "Wounded Commander1 arrived"`
On Deactivation	`hint "Wounded Commander1 left"`

2.2.4 Combining and Executing Multiple Basic Scripting Commands

Scripting commands can be combined and / or executed one after another in any of the fields described previously. Each command has to be separated by a semicolon (;).

EXAMPLE

Multiple Basic Script Commands: In the **Initialization Statements** of unit `soldier1`, make the unit wounded and display the message that the unit is wounded:

Initialization Statements	`this setDamage 0.9; hint "Soldier1 is wounded"`

Combining Basic Script Commands: Also in **Initialization Statements**, one command returns a result (`_pos`) which is used in another command:

Initialization Statements	`_pos = position this; hint str _pos`

2.3 Scripting Essentials

This chapter outlines the essentials aspects required for the creation of a basic script. To have a working basic script, knowledge of data types, variables and operators are needed.

- Basic Data Types (on the next page) - A variable / parameter must be of a certain data type (Array (on page 23), String (on page 21), Number (on page 21), and so on).
- Basic Variables / Parameters (on page 24) - A variable is needed for a scripting command to work. When a variable is passed to a command, it can also be called a parameter. Without the correct parameters, a command does not work.
- Operators (on page 28) - An operator (such as an arithmetic operator) can have one or two variables in SQF.

2.3.1 Basic Data Types

There are several data types that can be used with scripting commands. The correct data type must be used or the simulation engine stops executing and reports an error.

The following basic data types are described:

- Boolean (below)
- Number (on the next page)
- String (on the next page)
- Object (on the next page)
- Side (on page 22)
- Group (on page 22)
- Array (on page 23)

For more data types, see Advanced Data Types (on page 153).

2.3.1.1 Boolean

A Boolean can be a parameter for a command or a question to the simulation engine that returns `TRUE` or `FALSE`. Usually used in condition lines of Triggers, Waypoints, or control structures.

Command where one of the parameters is of the Boolean data type:

```
Unit1 allowDamage FALSE;
```

Syntax	
Syntax:	object **allowDamage** allow
Parameters:	• object: Object - The object. • allow: Boolean - If false, then unit / object cannot be injured / damaged, or killed / destroyed.
Return Value:	Nothing

Command whose return value is of the Boolean data type:

```
Unit1 in truck1;
```

Syntax	
Syntax:	person **in** vehicle
Parameters:	• person: Object - Person to check. • vehicle: Object - Vehicle to check.
Return Value:	Boolean - Returns true, if the person is in the vehicle, and false otherwise.

2.3.1.2 Number

Any real number (for example, -1, 0, 1, 10.123).

Command where one of the parameters is of the Number data type:

```
Unit1 setDir 270;
```

Syntax	
Syntax:	object **setDir** heading
Parameters:	• object: Object - Affected object. • heading: Number - Object heading angle. Angles are measured in degrees clockwise from north. The accepted heading range is from 0 to 360. Negative angles represent a counter-clockwise angle and the angle can be of any size.
Return Value:	Nothing

2.3.1.3 String

Specific text enclosed by "quotes" or 'apostrophes'. Usually used with messages displayed on the screen. When scripts use markers, the `"marker name"` is a string.

EXAMPLE

`"marker_name"`, `"VBS2_AKM"`, `"my message to be displayed"`

Command where one of the parameters is of the String data type:

```
Unit1 addWeapon "VBS2_AKM";
```

Syntax	
Syntax:	unit **addWeapon** weapon
Parameters:	• unit: Object - The unit to add the weapon to. • weapon: String - The class name of the weapon to add.
Return Value:	Nothing

2.3.1.4 Object

Name property of any object (unit, vehicle, trigger, building, and so on) added in the Editor.

EXAMPLE

`unit1`, `truck1`, `bldg1`

Command that uses parameters of the Object data type:

```
unit1 moveInCargo truck1;
```

Syntax	
Syntax:	unit **moveInCargo** vehicle
Parameters:	• unit: Object - Unit to move. • vehicle: Object - Vehicle to move to.
Return Value:	Nothing

2.3.1.5 Side

The side an object belongs to. Most common are: `EAST`, `WEST`, `CIVILIAN` (these are not text strings).

Command where one of the parameters is of the Side data type:

```
Temp_grp = createGroup WEST;
```

Syntax	
Syntax:	**createGroup** side
Parameters:	• side: Side - Data Type - Can be: east, west, resistance, civilian or sideLogic.
Return Value:	Group - Created group. Is grpNull, if the maximum number of groups has been exceeded, or no center exists for that side. Groups that do not have any members (yet), return sideUnknown as their side.

2.3.1.6 Group

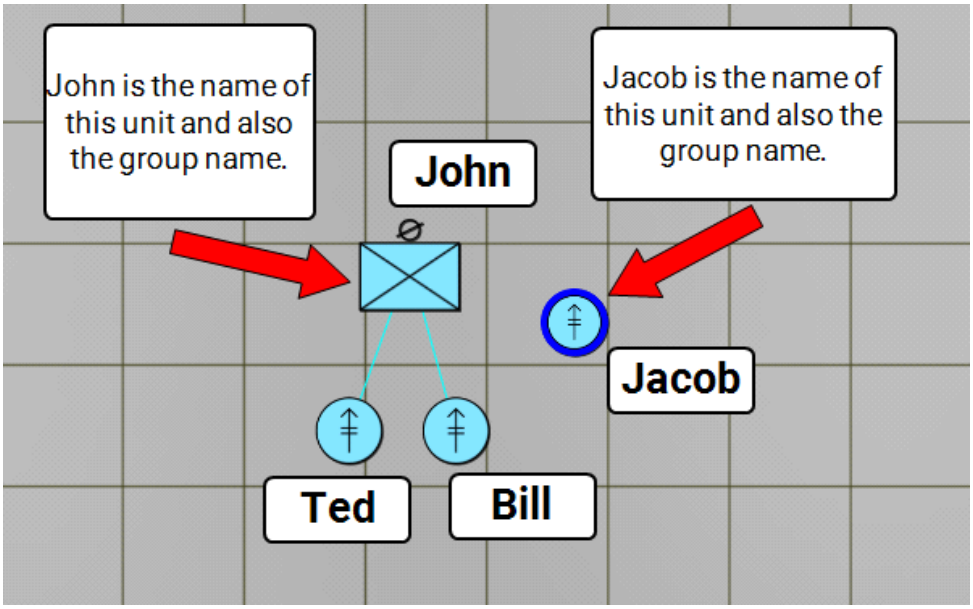
Defined in the simulation as a collection of several units or a single named unit.

Command whose return value is of the Group data type:

```
group player;
```

Syntax	
Syntax:	**group** unit
Parameters:	• unit: Object - Unit to check.
Return Value:	Group - Group the queried unit belongs to. For dead units grpNull is returned.

Image-2: Example of group names in VBS4



2.3.1.7 Array

List of items enclosed by square [brackets]. Each element is separated by a comma. Usually a list of items such as names, strings, or a position.

NOTE

An Array of Arrays is also possible.

EXAMPLE

- `[unit1, unit2, unit3]`
- `["VBS2_AU_AW50", "VBS2_AKM"]`
- `[["VBS2_AU_AW50", 3], ["VBS2_AKM", 2]]`

Command where the position parameter is based on the Array data type:

```
Unit1 setPos [10,10,0];
```

Syntax

Syntax:	object **setPos** pos
Parameters:	• object: Object - Affected object. • pos: Position-2 - Object position.
Return Value:	Nothing

2.3.2 Basic Variables / Parameters

A variable stores a value. The value can be a number, text, or data of a more complex type. When a variable is used with a script command, it is also a command parameter.

- Relation to Data Types (below)
- Declaration (below)
- Assignment (on the next page)
- Unassigning (on the next page)
- Local and Global Variables (on page 26)
- Basic Variables / Parameters (above)

For more complex aspects of variables / parameters, see Advanced Variables / Parameters (on page 157) .

2.3.2.1 Relation to Data Types

The list of valid types can be found in Basic Data Types (on page 20).

Unlike in programming languages such as C++ or Java, the type of the variable does not need to be explicitly stated when the variable is declared. Instead, the type is automatically inferred from the assigned value.

In contrast to languages such as JavaScript or PHP, there is no implicit type conversion. This means that if a script command expects a String and is given a Number instead, it does not automatically convert the number to a string but rather generates an error.

2.3.2.2 Declaration

Variables are automatically declared when assigned to for the first time.

It is also worth noting that since variables do not need to be declared, it is possible that your code may attempt to access undefined or undeclared variables. If this is the case, these variables are treated as nil. To avoid this, you can check if a variable is defined with isNil (https://sqf.bisimulations.com/display/SQF/isNil). Most script commands fail with an error silently when receiving an undefined variable. An undefined variable returns "any". However, there are exceptions where passing a "nil" causes the script command to perform some special behavior (usually resetting something to the default state).

Rules for variable name declaration:

- Only text characters and numbers (a-z, A-Z, 0-9).
- No special characters except underscores (_).
- Cannot start with a number or underscore.

- No spaces.
- No scripting commands and no duplicates.

2.3.2.3 Assignment

Variable assignment is achieved using the = operator.

EXAMPLE

```
myVar = 3;
```

Because variables are not strictly typed, any value can be assigned to any variable at any time.

EXAMPLE

```
myVar = 3;
// MyVar is: 3
hint ("MyVar is: " + str myVar);
myVar = myVar + 1;
// MyVar is: 4
hint ("MyVar is: " + str myVar);
myVar = "Hello world!";
// myVar is: "Hello world!"
hint ("MyVar is: " + str myVar);
/* Produce an error, since myVar is now a string,
   and string + scalar operation is not valid
*/
myVar = myVar + 1;
```

To avoid potential errors from variables being of unexpected type, the type of a variable can be queried using the typeName (https://sqf.bisimulations.com/display/SQF/typeName) command.

2.3.2.4 Unassigning

It may be desired at times to unassign or undefine a variable. To do this, you can assign the variable to nil.

```
myVar = nil;
```

2.3.2.5 Local and Global Variables

There are typically two types of variables: Global and Local.

Global variables can be accessed from any scope. For example, if declared within a script file (for example, `init.sqf`), a global variable will be visible to code in any script. This makes global variables risky to use because you can cause a name collision if you have two instances of the same script running, and both scripts are trying to use the same variables.

Local variables exist only within the scope (see Scope (on page 158)) where they are declared. For example, if declared within a script file (for example, `init.sqf`), local variables will only be visible to the code within that script.

TIP

Try to avoid global variables, and use local variables wherever possible.

Local variables are declared by prefixing them with an underscore (_).

EXAMPLE

```
// This global variable persists outside of this scope
myGlobalVar = 10;
/* This local variable can only be accessed within this script.
   When this script finishes executing, the variable will no longer exist
*/
_myLocalVar = 3;
```

For more information on where a global variable exists and can be accessed, see Namespaces (on page 157) and Scope (on page 158).

2.3.2.6 'this', '_this', and 'thislist'

The keywords `this`, `_this` and `thislist` are reserved for special variables in VBS4.

- **this**

  The keyword is used in the initialization of objects, and refers to the object itself. It is also used in Waypoints and Triggers, and designates the Waypoint or Trigger objective, respectively.

- **_this**

  The keyword is used for passing arguments to other scripts (for example, see Event Handlers (on page 61)).

- **thislist**

  The keyword is used in Waypoints and Triggers. In Triggers, it refers to all the objects which fulfill the condition of a Trigger. In Waypoints, it refers to all the objects that have completed a Waypoint.

2.3.3 Operators

Operators are the basic commands each programming language, including SQF, is built on. They provide the ability to perform basic mathematical and logical operations.

2.3.3.1 Terms

Term	Description
Operand	An operand is any value given to an operator. **EXAMPLE** The operands `1` and `2` can be used with the `+` operator.
Expression	An expression is any code that returns a value. **EXAMPLE** The expression `1+2` returns the value `3`.
Nullary Operator	Just the operator, without any operands, which makes it any engine-defined variable.
Unary Operator	A unary operator is an operator that requires only one operand. **EXAMPLE** `operator expression`
Binary Operator	A binary operator is an operator that requires two operands. **EXAMPLE** `expression operator expression`

2.3.3.2 Assignment Operators

Assignment operators are used to assign values to a variable. SQF provides only one assignment operator (=).

General syntax: `variable = expression`

EXAMPLE

```
a = b
```

Variable `a` is assigned with the value of variable `b`.

```
a = b*c
```

The expression is the multiplication of `b` and `c`, the result of which is assigned to `a`.

2.3.3.3 Arithmetic Operators

All operands of arithmetic operations must be Numbers. Arithmetic operations always return a Number.

Unary arithmetic operators:

Operator	Name	Description	Example
-	Negation	Negates an operand.	`-a`

Binary arithmetic operators:

Operator	Name	Description	Example
+	Addition	Adds two operands.	`a + b`
-	Subtraction	Subtracts `b` from `a`.	`a - b`
*	Multiplication	Multiplies two operands.	`a * b`
/	Division	Divides `a` by `b`.	`a / b`
%	Modulus	Returns the remainder of the division `a / b`.	`a % b`
mod	Modulus	The same as for `%`.	`a mod b`
^	Raise to the power of	Raises `a` to the power of `b`.	`a ^ b`

2.3.3.4 Logical Operators

Logical operators evaluate Boolean values. All operands of logical operations are Booleans. A logical operation always returns a Boolean.

Unary logical operators:

Operator	Name	Description	Example
!	Not	The not operator always returns the inverse Boolean value. If a Boolean `a` is true, `!a` returns false, and vice versa.	`!a`
not	Not	The same as for `!`.	`not a`

Binary logical operators:

Operator	Name	Description	Example
&&	And	Only returns true if both operands are true.	`a && b`
and	And	The same as for `&&`.	`a and b`
||	Or	Returns true if one or both operands are true.	`a || b`
or	Or	The same as for `||`.	`a or b`
<>	Xor	Returns true if exactly one operand is true.	`a <> b`
xor	Xor	The same as for `<>`.	`a xor b`

Combined logical operators:

Operator	Combination	Description
Xor	((a || b) && !(a && b))	The same as for `<>`.
Nor	!(a || b)	Returns true if none of both values is true.
Nand	!(a && b)	Returns true if not both values are true at the same time.

2.3.3.5 Comparison Operators

Comparison operators compare two values. Operands of comparisons may be of type Number (on page 21), Side (on page 22), String (on page 21), Object (on page 21), Group (on page 22), Structured Text (on page 155), Config (on page 153), Control / Display (on page 153). Comparisons always return a Boolean: true if the comparison matches, false if not.

Operator	Name	Description	Example
==	Equal	Returns true if both operands are equal.	`a == b`
!=	Not equal	Returns true if `a` is not equal to `b`.	`a != b`

Operator	Name	Description	Example
<	Less than	Returns true if `a` is less than `b`.	`a < b`
>	Bigger than	Returns true if `a` is bigger than `b`.	`a > b`
<=	Less or equal	Returns true if `a` is smaller or equal to `b`.	`a <= b`
>=	Greater or equal	Returns true if `a` is bigger or equal to `b`.	`a >= b`

2.3.3.6 Array Operators

SQF offers its own operators to handle arrays. All operands have to be of type Array. The return value of an array operation is an Array.

Unary array operators:

Operator	Name	Description	Example
+	Copy	Normally arrays are assigned by reference. That means, if you assign array `a` to array `b` and change `a` afterward, `b` also gets changed. Use the copy operator to avoid this otherwise useful feature.	`+myArray`

EXAMPLE

```
_arrayA = [1,2];
_arrayB = _arrayA;
_arrayA set [0,5];
```

`_arrayA` results in `[5,2]`.

`_arrayB` results in `[5,2]`.

```
_arrayA = [1,2];
_arrayB = +_arrayA;
_arrayA set [0,5];
```

`_arrayA` results in `[5,2]`.

`_arrayB` results in `[1,2]`.

Binary array operations:

Operator	Name	Description	Example
+	Concatenation	Adds the second array to the end of the first array.	`myArray1 + myArray2`

Operator	Name	Description	Example
-	Removal	Removes all elements of the second array from the first array.	`myArray1 - myArray2`

EXAMPLE

```
_arrayA = [1,2];
_arrayB = [3,2,4];
_arrayC = _arrayA + _arrayB;
```

`_arrayC` results in `[1,2,3,2,4]`.

```
_arrayA = [1,2,3,2,4];
_arrayB = [2,3];
_arrayC = _arrayA - _arrayB;
```

`_arrayC` results in `[1,4]`.

2.3.3.7 String Operators

SQF offers one single string operator to concatenate strings. Both operands must be Strings. The return value of a string operation is a String.

Operator	Name	Description	Example
+	Concatenation	Adds the second string to the end of the first string.	`myString1 + myString2`

EXAMPLE

```
_stringA = "Hello ";
_stringB = "World!";
_stringC = _stringA + _stringB;
```

`_stringC` results in `"Hello World!"`.

2.4 Error Messages

As scripts get longer and more complex, the probability of errors increases. Error messages occur when the simulation engine loads the line of code but is unable to interpret the command because of missing or incorrect parameters, or a wrong command, or function name.

Error messages can appear in:

- VBS4 (at the top of the screen)
- The RPT (report) file, located in:

  `%LOCALAPPDATA%\VBS4\VBS4.RPT`

The error message displays the basic information where the problem occurred - that is, the line of code that could not be recognized by the simulation engine.

> **NOTE**
>
> The position the error is reported at may not be entirely accurate.

A typical error resembles the following:

```
a = b*c
'...\__tmpPREVIEWsave.intro\test.sqf"
unit1 |#|setUnitPo "UP"

'


Error Missing ;
File
C:\Users\GunRunner\Documents\VBS4\Battlespaces\__tmpPreviewDir.Intro\Missions\__
tmpPreviewDir.Intro\test.sqf, line 1
```

- `'...\__tmpPREVIEWsave.intro\test.sqf"`

  The file where the error occurred.

- `unit1 |#|setUnitPo "UP";`

  The hash symbol (#) indicates the place in the code unrecognized by the simulation engine.

- `Error Missing ;`

  The error message.

- `File C:\Users\GunRunner\Documents\VBS4\Battlespaces\__tmpPreviewDir.Intro\Missions\__tmpPreviewDir.Intro\test.sqf, line 1`

  The script file location and line number of the place in the code unrecognized by the simulation engine.

2.4.1 Common Errors

Error	Description
Invalid number in expression	A mathematical error. The formula is incomplete. `_newVar = 2 + ;` **Image-3: Invalid number in expression error** 

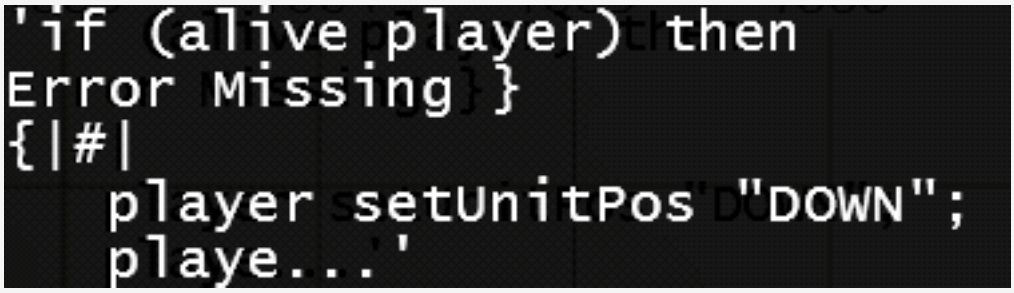
Generic error in expression	Using incorrect data type for numbers or incorrect data types when using mathematical functions.
	`_variable = "string" + 42;`
	`_variable = position player + "not an array";`
	`_vehicle setFuel "1";`
	Image-4: Generic error in expression error
	 

Type "something" expected "something else"	Data type used does not match the expected data type. `getHeight "soldier1";` **Image-5: Type "something" expected "something else" error** 
Zero Divisor	A mathematical error. The command is trying to divide a number by zero. `15/0` **Image-6: Zero Divisor error** 

Error	Description
Error missing }	This is one of the more difficult errors to fix. The simulation engine shows where the script stopped making sense. The actual error may be above or below the line number.

```
// Simple code
If (alive player) then
{
    player setUnitPos "DOWN";
    player sideChat "done";
```

Image-7: Error missing }

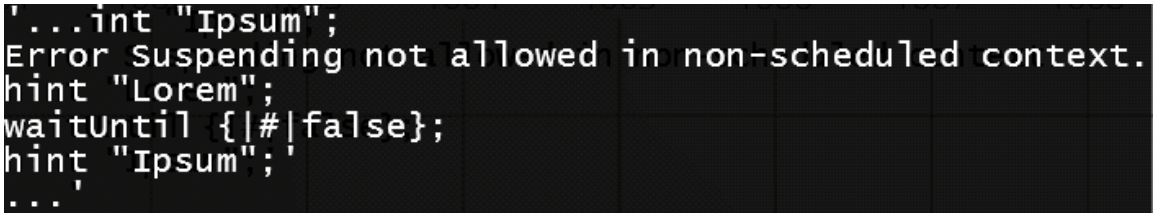


> **NOTE**
>
> The error is reported as being on line 3, this actually refers to the first element of the {}; pair. In this example, the missing } is on line 5. The } would also be followed with a ;.

So the correct script would be:

```
If (alive player) then
{
    player setUnitPos "DOWN";
}; // This was missing
player sideChat "done";
```

Error	Description

Error Suspending in Non-Scheduled Context	The commands sleep (https://sqf.bisimulations.com/display/SQF/sleep) and waitUntil (https://sqf.bisimulations.com/display/SQF/waitUntil) are not allowed in a non-scheduled environment (such as an event handler, Init line, Debug Console).
	`// Simple code`
	`hint "Lorem";`
	`sleep 1;`
	`hint "Ipsum";`
	`hint "Lorem";`
	`waitUntil {false};`
	`hint "Ipsum";`
	Image-8: Error Suspending in Non-Scheduled Context
	 

2.4.2 Tips for Resolving Errors

Here are a few tips on how to resolve errors:

- Be methodical and work on one section at a time.
- Save working copies as you go.
- Write the code in small increments, when possible, to make problem easier to find.
- Use the Script Debugger Plugin (on page 146) to debug your code.
- To check the value of a single variable use the command str (https://sqf.bisimulations.com/display/SQF/str). If nothing is displayed on the screen, the variable does not have a value (scalar error). It is likely the variable is misspelled somewhere.

```
player sideChat str _var1;
// Advanced scripting
hint format ["dir %1, dis %2", _var1, _var2];
player sideChat format ["dir %1, dis %2", _var1, _var2];
```

- Display messages when certain parts of the code are successful.

```
player sideChat "reached section 1";
```

2.5 Init.sqf

The `init.sqf` file allows the execution of script statements before the mission starts (when the mission is initialized). It is an alternative to the Initialization Statements in an Object (on page 16). If this file exists in the parent mission folder, when the scenario starts, the `init.sqf` file is executed by all computers connected to the server.

To create an example init.sqf, follow these steps:

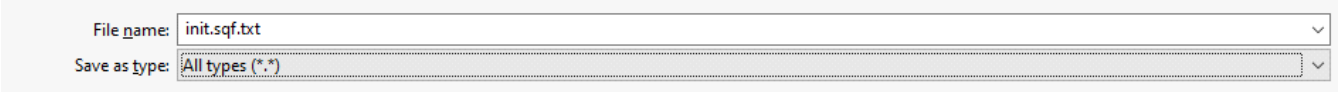
1. Create a mission with at least one playable unit and save it - the mission gets saved in the mission folder (for more information, see Battlespace Files and Folders in the Introduction to VBS4 Guide).
2. Open a text editor, type the following script:

```
player sideChat "my first external file";
```

3. Save the file as "**init.sqf**" in your mission folder.

> **NOTE**
> When saving the file, be sure to change the file type to "**All Files**". The simulation engine does not recognize the file if it has a different extension than `.sqf`.

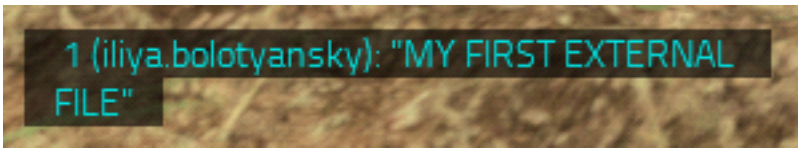
Image-9: Save the file as init.sqf



4. Start the mission.

The message is displayed.

Image-10: Message from init.sqf is displayed on the screen



2.6 Exercises

The following five exercises repeat the concepts explained in the Basic Scripting chapter. Each exercise has links to topics from the chapter (as well as the VBS4 Editor and Trainee Manuals) that can assist you in completing the exercise.

2.6.1 Exercise 1: Initialization Statement

Write a script statement for a unit / vehicle that outputs `"Hello World!"` on mission start.

For assistance, see: Initialization Statement in Using Basic Scripts (on page 15), the hint (https://sqf.bisimulations.com/display/SQF/hint) command.

Verification steps:

1. Create a player unit / vehicle.
2. Start the mission.

The message `"Hello World!"` is displayed.

2.6.2 Exercise 2: On Activation

Write a script statement for a unit / vehicle that outputs the name of the unit / vehicle when reaching a waypoint.

For assistance, see: Waypoints and Advance Order in the VBS Control AI Manual, On Activation in Using Basic Scripts (on page 15), the hint (https://sqf.bisimulations.com/display/SQF/hint) and player (https://sqf.bisimulations.com/display/SQF/player) commands.

Verification steps:

1. Create a player unit / vehicle.
2. Create an Advance Order waypoint and assign it to the unit / vehicle.

When the waypoint is reached, the name of the unit / vehicle is displayed.

2.6.3 Exercise 3: On Deactivation

Write a script statement for a unit / vehicle that outputs `"Inside trigger area"` / `"Outside trigger area"`, after entering / exiting a trigger area.

For assistance, see: Triggers in the VBS4 Editor Manual, On Deactivation in Using Basic Scripts (on page 15), the hint (https://sqf.bisimulations.com/display/SQF/hint) and player (https://sqf.bisimulations.com/display/SQF/player) commands.

Verification steps:

1. Create a player unit / vehicle.
2. Create a trigger.

3. Make the unit / vehicle enter the trigger area.
4. Make the unit / vehicle exit the trigger area.

The message `"Inside trigger area"` / `"Outside trigger area"` is displayed.

2.6.4 Exercise 4: On Condition

Write script statements that make a unit / vehicle wounded / damaged on mission start and output `"Commander wounded!"` after the unit / vehicle enters a trigger area.

For assistance, see: Triggers in the VBS4 Editor Manual, Initialization Statement in Using Basic Scripts (on page 15), On Condition in Using Basic Scripts (on page 15), Comparison Operators in Operators (on page 28), the hint (https://sqf.bisimulations.com/display/SQF/hint) and setDamage (https://sqf.bisimulations.com/display/SQF/setDamage) commands.

Verification steps:

1. Create a player unit / vehicle that is wounded / damaged on mission start.
2. Create a trigger.
3. Make the unit / vehicle enter the trigger area.

The message `"Commander wounded!"` is displayed.

2.6.5 Exercise 5: Init.sqf

Write a script that puts a BLUFOR unit in the cargo of a BLUFOR Chinook helicopter on mission start.

For assistance, see: Init.sqf (on page 37), the moveInCargo (https://sqf.bisimulations.com/display/SQF/moveInCargo) command, 3D World Actions (to verify that the BLUFOR unit is in the cargo) in the VBS4 Trainee Manual.

Verification steps:

1. Create a player unit.
2. Create a BLUFOR unit.
3. Create a BLUFOR Chinook helicopter.

Start the mission and observe that the BLUFOR unit is in the cargo of the Chinook helicopter.

> **NOTE**
>
> You can verify that the BLUFOR unit is in the cargo by approaching the Chinook helicopter, and selecting the **Open Ramp** 3D World Action (see 3D World Actions in the VBS4 Trainee Manual).

3. Intermediate Scripting

To create more complex scripts, external files are required. These external files are called script (or SQF) files. Scripts that are more than one line of code often have conditional handling, and can have repeated operations. This is where control structures come in.

To make the lives of script developers easier, VBS4 comes with its own Function Library that features many useful preexisting code elements called functions which enable developers to tackle complex problems in simulation behavior and not have to "reinvent the wheel".

Almost anything that happens in VBS4 is a simulation event. Event handlers enable you to attach your script to particular simulation events.

Intermediate scripting often produces more sophisticated and elusive errors. VBS4 comes equipped with special debugging tools that help script developers to overcome those errors.

- Basic External Files (on the next page) - How to create external script (or SQF) files.
- Control Structures (on page 43) - On conditions and loops (repeated operations) in scripts.
- Function Library (on page 60) - How the use the VBS4 Function Library.
- Event Handlers (on page 61) - The simulation event-handling system and how it is related to scripting.
- UI Event Handlers (on page 125) - Event handlers that are triggered, when interacting with the UI.
- VBS4 debugging tools:
  - Developer Console (on page 138) - A tool that allows you to test VBS4 commands and functions directly in the simulation.
  - Debug Console Plugin (on page 141) - A plugin that extends the functionality of the Developer Console.
  - Script Debugger Plugin (on page 146) - A plugin with a UI that is similar to common Integrated Development Environments (IDEs) with advanced debugging features.

3.1 Basic External Files

Apart from `init.sqf`, a user-defined external SQF files can be created to contain SQF code. This SQF code can call other external scripts located in other files.

External SQF files are use for the following reasons:

- A user-defined script is usually longer and more complex (containing several lines of code, control structures, conditions, and so on) than scripts specified directly in various Editor UI fields.
- To create scripts that are more modular, with the option to pass parameters.

3.1.1 Creating a User Defined SQF File

To create an example userdefined.sqf, follow these steps:

1. Create a mission with at least one playable unit and save it (the mission gets saved in the mission folder - for more information, see Battlespace Files and Folders in the Introduction to VBS4 Guide).
2. Open a text editor, type the following script:

   ```
   player sideChat "my first external file";
   ```

3. Save the file as "**userdefined.sqf**" in your mission folder.

   > **NOTE**
   >
   > When saving the file, be sure to change the file type to "**All Files**". The simulation engine does not recognize the file if it has a different extension than `.sqf`.

The next section describes how to execute the user defined SQF file.

3.1.2 Executing a User Defined SQF File

The safest command to execute an external file is call (https://sqf.bisimulations.com/display/SQF/call). The engine waits for the called script to finish executing and only then proceeds with the rest of the calling script. This command can be used in any place where the code is executed (for example, see Using Basic Scripts (on page 15), or one user defined script can call another user defined script).

To run a user defined SQF file, follow these steps:

1. Add one of the following SQF statements to the mission `init.sqf`:

```
_returned_value = call "file_name.sqf";
```

Or:

```
_returned_value = call "sub-folder/file_name.sqf";
```

- **_returned_value** - The result of the last command executed in the called code.

WARNING

If the external SQF file has no explicitly specified return value, an SQF error occurs.

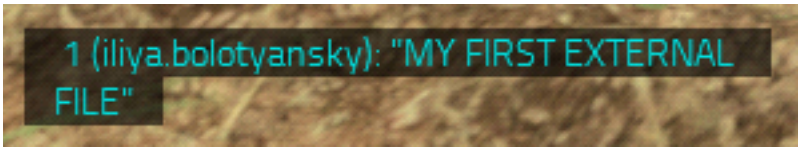
EXAMPLE

```
Script Code;
/* The last script statement can be a variable name,
   constant number, or preprocessor #define name
*/
Last Script Statement;
```

- "**file_name.sqf**" - The name of your script file in the mission folder. Sub-folders can also be created in the mission folder, and external script files can be placed there.

2. Start the mission.

The following message is displayed on the screen:



If you want the external-file script to execute in parallel with the calling script, see: execVM (https://sqf.bisimulations.com/display/SQF/execVM) or spawn (https://sqf.bisimulations.com/display/SQF/spawn).

3.2 Control Structures

Control Structures are statements that are used to control the execution flow of script statements. They are sequences of scripting code which help to control complex procedures. You can use control structures to define code which is only executed under certain conditions or which is executed repeatedly.

WARNING

The main purpose of the code fragments in this topic is to demonstrate the algorithmic workflow of each control structure, rather than provide a larger / specific SQF code context, which can be copied, pasted, and executed directly in VBS. Some code fragments contain placeholder text, marked with a yellow rectangle. To execute the code fragments in VBS, this text needs to be replaced with valid SQF code.

The placeholder text looks as follows:

```
Placeholder Text
```

3.2.1 Conditional Structures

Conditional structures control the conditions under which the code should be executed.

3.2.1.1 if-Statement

The if-statement encapsulates the code that is only executed if a certain condition is met:

```
if (CONDITION) then {
    STATEMENT;
    ...
};
```

- `CONDITION` is a Boolean statement or variable which returns either true or false. The code contained in the block that follows it is only executed if the condition is true. If the condition returns false, the code block is ignored.
- `STATEMENT` is one or more statements. They may be commands, assignments, control structures, and so on.

Example:

```
if (damage car > 0) then {
    hint "damaged";
};
```

In this example, the following happens:

- If the car damage is greater than zero (the condition is true), then `hint "damaged"` is executed.
- If the car damage equals zero (the car is intact, so the condition is false), then the code block is ignored.

3.2.1.2 Alternative Code (else)

With the `else` clause it is possible to encapsulate the code which is executed when the condition is not true.

```
if (CONDITION) then {
    STATEMENT_TRUE;
    ...
} else {
    STATEMENT_FALSE;
    ...
};
```

The second sequence of statements (`STATEMENT_FALSE`) are executed when `CONDITION` is false.

Example:

```
if (damage car > 0) then {
    hint "damaged";
} else {
    hint "intact";
};
```

In this example, the following happens:

- If the car is damaged (damage greater than zero), then the text `"damaged"` is displayed.
- If the car is undamaged (damage equals zero), then the `else` block is used and the text `"intact"` is displayed.

3.2.1.3 Alternative Code (elseif)

With the `elseif` clause it is possible to encapsulate multiple conditional code sections which are only executed when the respective condition is true.

Example:

```
if (damage player == 0) then {
    hint "no damage";
} elseif (damage player < 0.3) then {
    hint "damage is less than 0.3";
} elseif (damage player < 1) then {
    hint "player is not dead yet";
} else {
    hint "player is dead";
};
```

In this example, the following happens:

1. If the player is undamaged (player damage equals zero), then the text `"no damage"` is displayed.
2. If player damage is less than 0.3, then the text `"damage is less than 0.3"` is displayed.
3. If player damage is less than 1, then the text `"player is not dead yet"` is displayed.
4. For all other values of player damage, the text `"player is dead"` is displayed.

3.2.1.4 Conditional Assignments

If-then structures can also be used to assign conditional values to variables. The value of the last executed statement in the if-then block is the return value.

Example:

```
// _state contains true, if player is alive
_state = if (alive player) then {true} else {false};
// _state contains "alive", if player is alive
_state = if (alive player) then {"alive"} else {"dead"};
```

In this example, the following happens:

- If the player is alive, then the variable `_state` is assigned the value `"alive"`.
- If the player is dead, then the variable `_state` is assigned the value `"dead"`.

3.2.1.5 Nested if-Statements

Since the if-statement is itself a statement, nested if-statements can be also created.

Example:

```
_life_state = alive player;
_ammo_state = someAmmo player;
if (_life_state) then {
    if (_ammo_state) then {
        hint "The player is alive and has ammo!";
    } else {
        hint "The player is alive and out of ammo!";
    };
} else {
    if (_ammo_state) then {
        hint "The player is dead and has ammo!";
    } else {
        hint "The player is dead and out of ammo!";
    };
};
```

In this example, the following happens:

1. The variable `_life_state` is assigned the player life status (true if alive, false if dead).
2. If the player has any ammunition, the variable `_ammo_state` is set to true. Otherwise, the variable is set to false.
3. One of the following happens:
    - If the player is alive (`_life_state` is true), then:
        - If the player has any ammunition (`_ammo_state` is true), then the text `"The player is alive and has ammo!"` is displayed.
        - If the player has no ammunition (`_ammo_state` is false), then the text `"The player is alive and out of ammo!"` is displayed.
    - If the player is dead (`_life_state` is false), then:
        - If the player has any ammunition (`_ammo_state` is true), then the text `"The player is dead and has ammo!"` is displayed.
        - If the player has no ammunition (`_ammo_state` is false), then the text `"The player is dead and out of ammo!"` is displayed.

3.2.1.6 switch-Statement

To execute specific code, depending on the value contained in a variable (or returned by a statement), multiple nested `if` blocks could be used. However, the more values there are to compare, the longer and harder to read the code gets. To make this process easier to code and to read, the switch-statement was introduced:

```
switch (VARIABLE) do {
    case VALUE1: {
        STATEMENT1;
        ...
    };
    case VALUE2: {
        STATEMENT2;
        ...
    };
    ...
};
```

The structure compares `VARIABLE` against all given values (`VALUE1, VALUE2, ...`). If any of the values matches, the corresponding block of statements is executed.

Example:

```
_color = "red";
switch (_color) do {
    case "blue": {
        hint "What a nice color";
    };
    case "red": {
        hint "Don't you get aggressive?";
    };
}
```

In this example, the following happens:

1. The variable `_color` is assigned the value `"red"`.
2. The value of `_color` is evaluated:
    - In case the color is blue (`_color` is `"blue"`), then the text `"What a nice color"` is displayed.
    - In case the color is red (`_color` is `"red"`), then the text `"Don't you get aggressive?"` is displayed.

To catch values that are not explicitly defined in one of the case definitions, a default block can be used:

```
switch (VARIABLE) do {
    case VALUE1: {
        STATEMENT;
        ...
    };
    case VALUE2: {
        STATEMENT;
        ...
    };
    default {
       STATEMENT;
       ...
    };
};
```

NOTE

There is no colon after the `default` tag.

You can specify multiple values for the same statement.

Example:

```
switch (VARIABLE) do {
    case VALUE1;
    case VALUE2;
    case VALUE3:
    {
        STATEMENT1;
        ...
    };
    case VALUE4: {
        STATEMENT2;
        ...
    };
    default {
       STATEMENT3;
       ...
    };
};
```

In this example, the following happens:

1. Variable `VARIABLE` is evaluated.
2. If the value of `VARIABLE` matches `VALUE1`, `VALUE2`, or `VALUE3`, execute `STATEMENT1`.

3. If the value of `VARIABLE` matches `VALUE4`, execute `STATEMENT2`.
4. For all other `VARIABLE` values, execute `STATEMENT3`.

The switch-Statement can be used to assign different values to a variable. The value of the last executed statement in the switch block is the return value.

Example:

```
_color = switch (side player) do {
    case west: {"ColorGreen"};
    case east: {"ColorRed"};
};
hint _color;
```

In this example, the following happens:

1. The variable `_color` is assigned the return value of the switch-Statement, checks the player affiliation / side:
   - In case the player is on the west side, `_color` is assigned the value of `"ColorGreen"`.
   - In case the player is on the east side, `_color` is assigned the value of `"ColorRed"`.
2. The value of `_color` is displayed.

3.2.2 Loops

Loops are used to execute the same code block for a specific or infinite number of times.

3.2.2.1 for-Loop

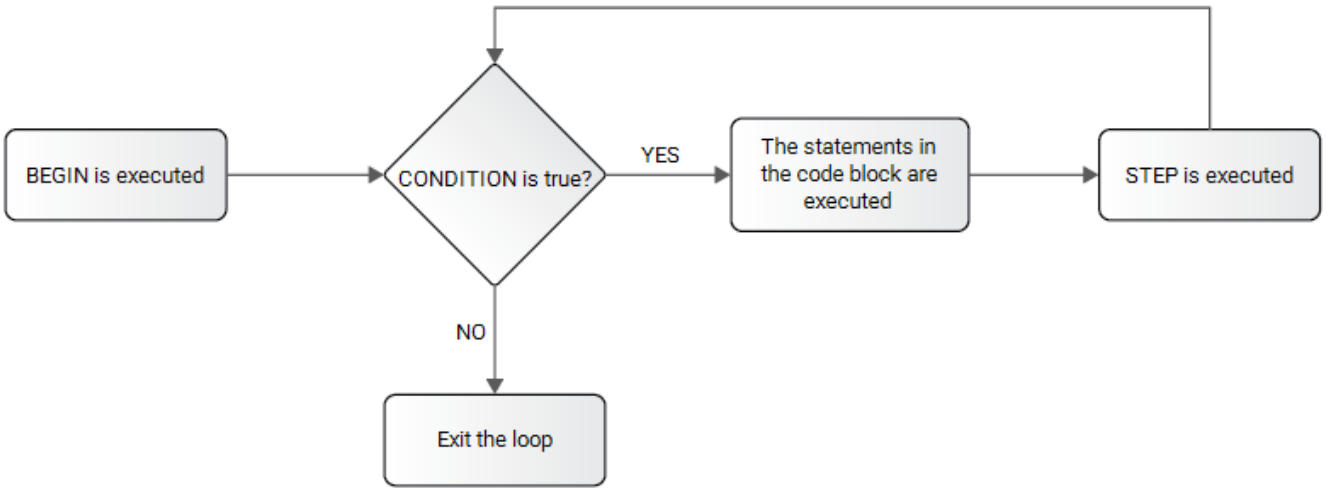
The for-loop repeats the same code block for a specific number of times.

The for-loop, whether it is executed through spawn (https://sqf.bisimulations.com/display/SQF/spawn) or execVM (https://sqf.bisimulations.com/display/SQF/execVM) or called directly, is blocking and atomic in execution. It does not terminate or get interrupted after a certain period of time or after a number of fixed iterations (unless there is a float overflow or other unexpected failure of the conditional statement). It differs from the while-loop, and has the potential to freeze VBS4.

```
for [{BEGIN}, {CONDITION}, {STEP}] do {
    STATEMENT;
    ...
};
```

- `BEGIN` is the number of statements executed before the loop starts.
- `CONDITION` is a Boolean condition evaluated before each loop.
- `STEP` is the number of statements executed after each loop.

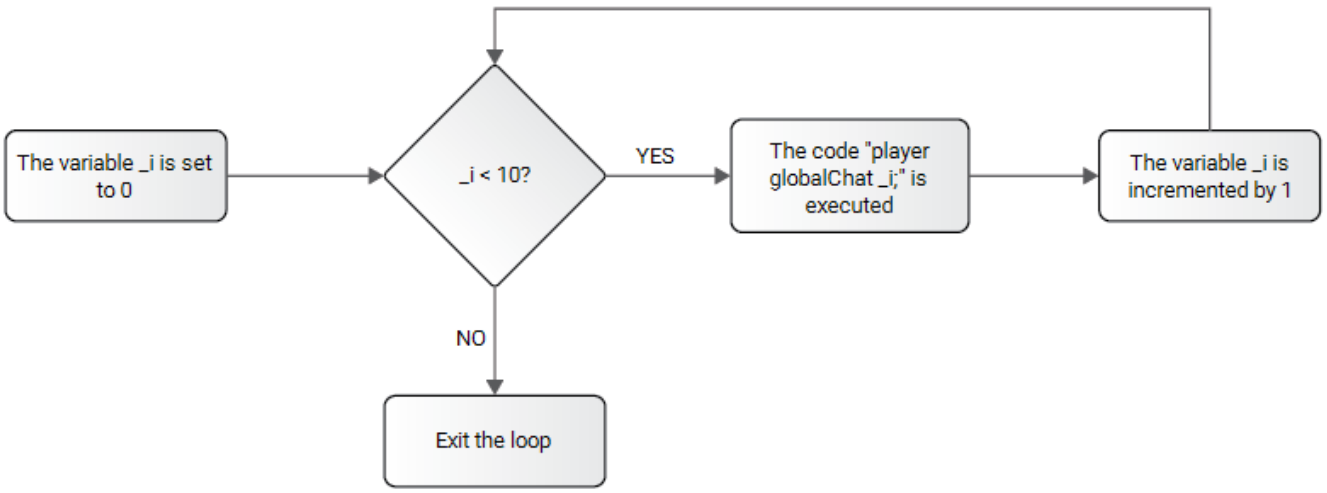
The loop executes in the following way:



> **NOTE**
>
> If `CONDITION` is false from the beginning, the code block is never executed.

Example:

```
// A loop repeating 10 times, displaying the numbers 0 to 9.
for [{_i=0}, {_i<10}, {_i=_i+1}] do {
    player globalChat format["%1",_i];
};
```



In this example, the following happens:

1. Enter the loop, and initialize the counter variable `_i` with 0.
2. One of the following happens:
   - If `_i` is not less than 10, exit the loop.
   - If `_i` is less than 10, a text with the value of `_i` is displayed.
3. Increment `_i` by 1.
4. Repeat step 2.

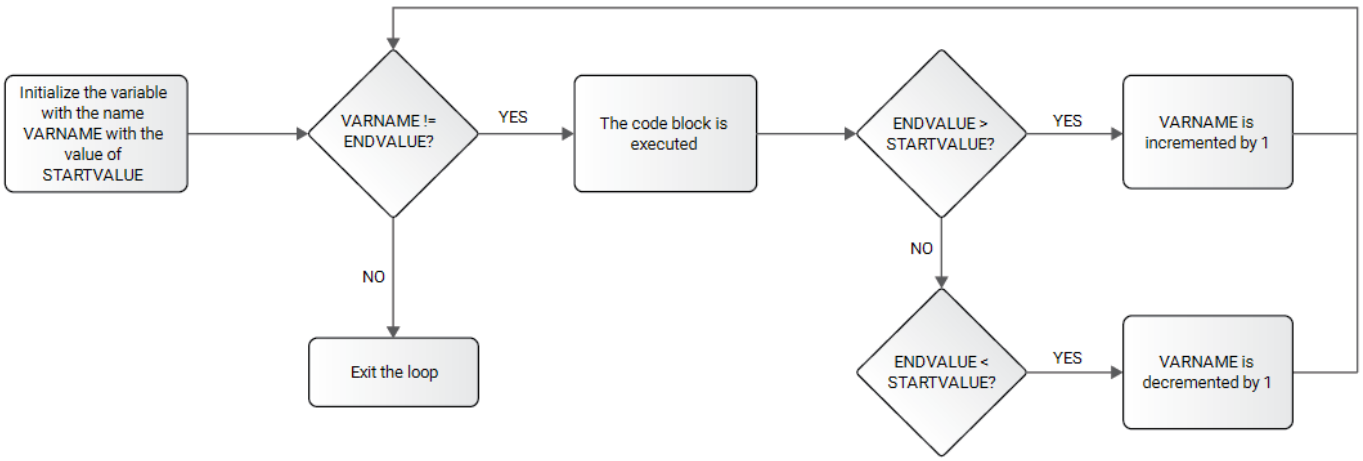
3.2.2.2 for-from-to-Loop

There exists an alternate syntax for the for-loop, which simplifies the last example above, and improves the performance of the loop as well.

```
for "VARNAME" from STARTVALUE to ENDVALUE do {
    STATEMENT;
    ...
};
```

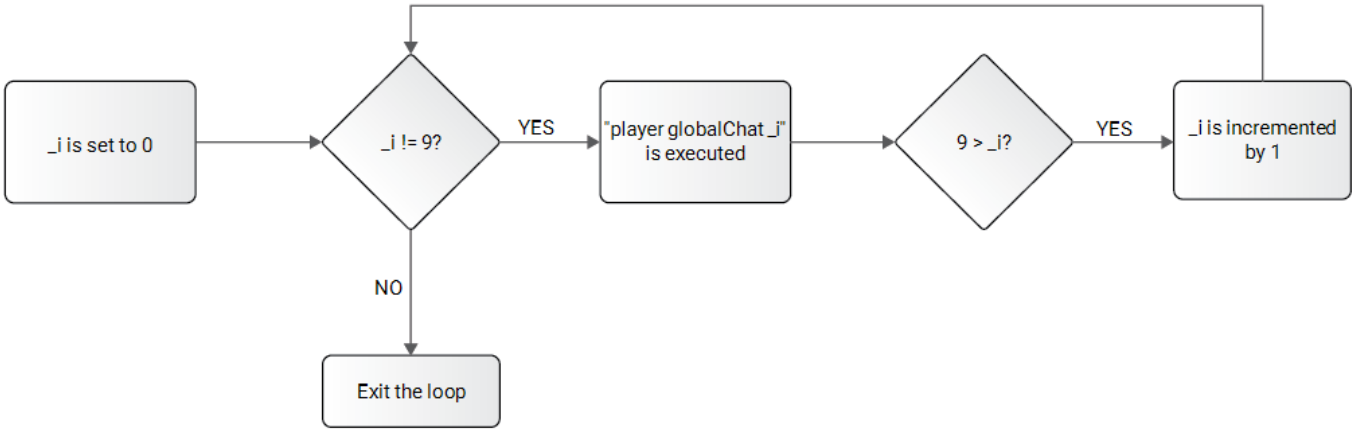
- `VARNAME` is any name given to the variable used to count the loop.
- `STARTVALUE` is a Number value given to the counter variable before the loop starts.
- `ENDVALUE` is a Number value until which the counter is incremented / decremented.

The loop executes in the following way:



Example:

```
// A loop that repeats 10 times
for "_i" from 0 to 9 do {
    player globalChat format["%1",_i];
};
```



In this example, the following happens:

1. Enter the loop, and initialize the counter variable `_i` with 0.
2. One of the following happens:
   - If `_i` is not less than 9, exit the loop.
   - If `_i` is less than 9, a text with the value of `_i` is displayed.
3. Increment `_i` by 1.
4. Repeat step 2.

WARNING

A backward loop (using a decrement) only works, if the decrement is specified as a custom step (see for-from-to-Loop with Custom Step (below)).

Example:

```
// Backward loop from 4 to 1
for "_i" from 4 to 1 step -1 do {
    hint format ["_i = %1", _i];
}};
```

In this example, the following happens:

1. Enter the loop, and initialize the counter variable `_i` with 4.
2. One of the following happens:
   - If `_i` is less than 1, exit the loop.
   - If `_i` is more than 1, a text with the value of `_i` is displayed.
3. Decrement `_i` by 1 (`step -1`).
4. Repeat step 2.

3.2.2.3 for-from-to-Loop with Custom Step

The default step to increment the variable in a for-from-to-loop is 1. You can set a custom step to have a different increment using this syntax:

```
for "VARNAME" from STARTVALUE to ENDVALUE step STEP do {
    STATEMENT;
    ...
};
```

`STEP` is a Number which defines the step by which the variable is incremented every loop.

Example:

```
// A loop repeating 5 times
for "_i" from 0 to 9 step 2 do {
    player globalChat format["%1",_i];
};
```

In this example, the following happens:

1. Enter the loop, and initialize the counter variable `_i` with 0.
2. One of the following happens:
   - If `_i` is not less than 9, exit the loop.
   - If `_i` is less than 9, a text with the value of `_i` is displayed.
3. Increment `_i` by 2.
4. Repeat step 2.

3.2.2.4 while-Loop

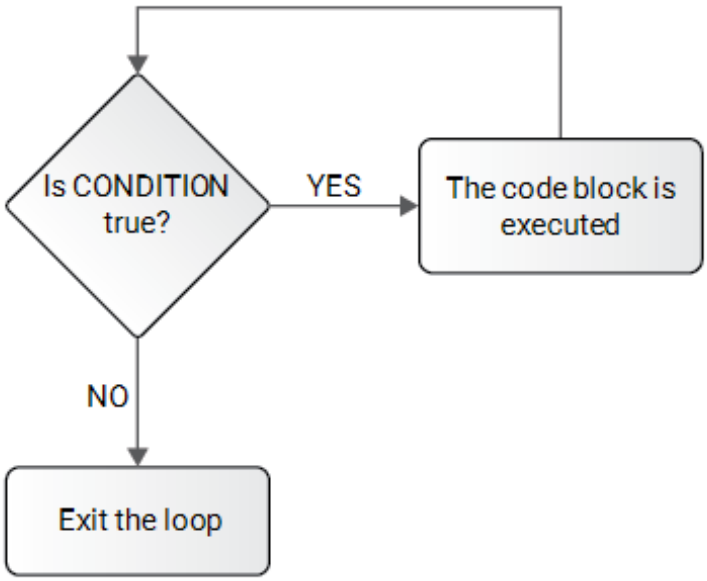
The while-loop repeats the execution of the same code block as long as a given Boolean condition is true.

```
while {CONDITION} do {
    STATEMENT;
    ...
};
```

NOTE

There are curly braces around `CONDITION`, unlike the round parentheses in an if-statement.

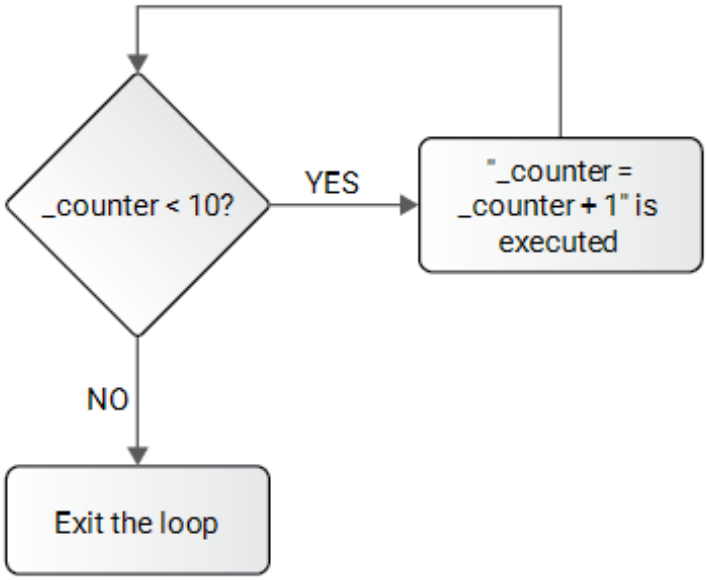
The loop executes in the following way:



A while-loop terminates without warning after 10,000 iterations if started from within a non-scheduled environment.

Example:

```
_counter = 0;
while {_counter < 10} do {
    _counter = _counter + 1;
};
player globalChat format["Counter: %1",_counter];
```



In this example, the following happens:

1. Set variable `_counter` to 0.
2. Enter the loop - one of the following happens:
    - If `_counter` is not less than 10, exit the loop.
    - If `_counter` is less than 10, increment it by 1.
3. The text `"Counter: 10"` is displayed.

3.2.2.5 waitUntil-Loop

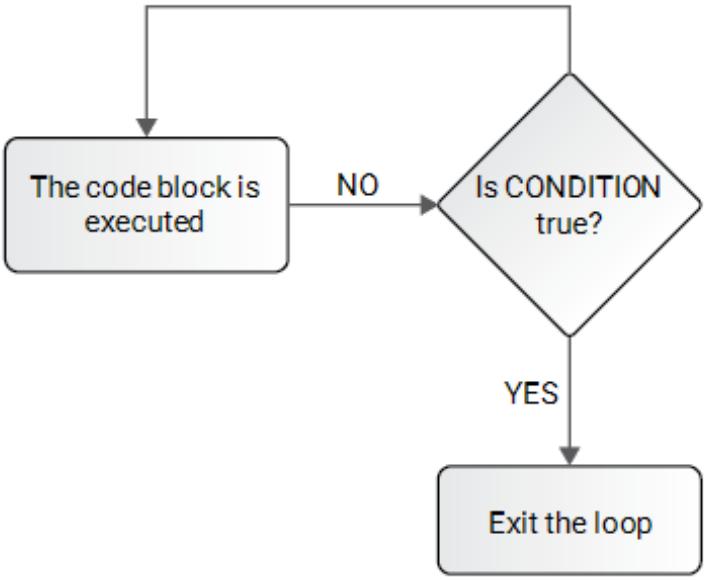
The waitUntil-loop repeats the execution of the same code block as long as a given Boolean condition is false.

> **NOTE**
>
> a waitUntil-loop is not the same as a while-loop. The former should only be used to block execution until the condition becomes true.

```
waitUntil {
    STATEMENT;
```

```
    ...
    CONDITION
};
```

The loop executes in the following way:

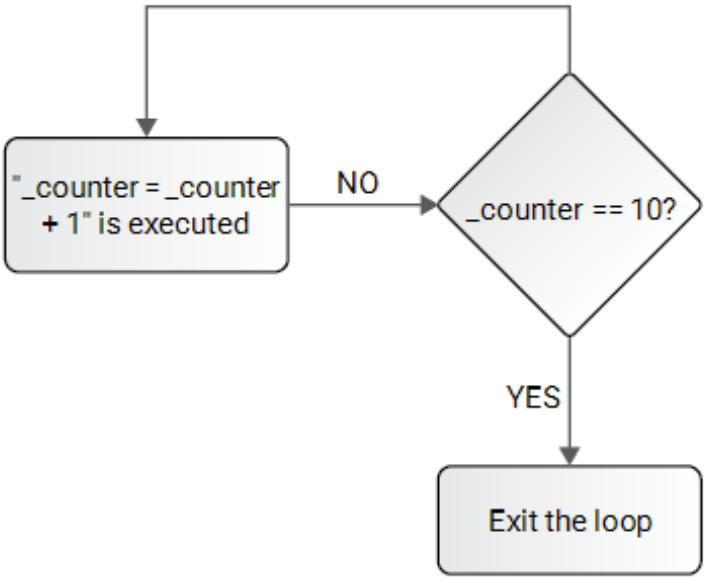


If `CONDITION` is always true, the loop quits after a single iteration of `STATEMENT`.

Because the test of the waitUntil expression takes place after the execution of the loop, a waitUntil loop executes one or more times. A waitUntil loop can be used only inside scheduled code. It is executed once per frame.

Example:

```
_counter = 0;
waitUntil {
    _counter = _counter + 1;
    _counter >= 10
};
player globalChat format["Counter: %1",_counter];
```



In this example, the following happens:

1. Set variable `_counter` to 0.
2. Enter the loop:
3. Increment `_counter` by 1.
4. If `_counter` is more than or equal to 10, exit the loop. If not, repeat 3.
5. The text `"Counter: 10"` is displayed.

3.2.2.6 forEach-Loop

To step through every element in an array, the forEach-loop can be used.

```
{
    STATEMENT;
    ...
} forEach ARRAY;
```

The code block is executed exactly (`count ARRAY`) times (the number of times which is the number of elements in the array).

The special variable `_x` can be used within the code block, which always references to the current item of the array. Another special variable inside a forEach-loop is `_forEachIndex`, which is accessible within the code block and represents the current item's array index.

Example:

```
_array = [unit1, unit2, unit3];
{
```

```
    _x setDamage 1;
} forEach _array;
```

In this example, the following happens:

1. In the first loop iteration, the statement `unit1 setDamage 1;` is executed.
2. In the second loop iteration, the statement `unit2 setDamage 1;` is executed.
3. In the third loop iteration, the statement `unit3 setDamage 1;` is executed.

While it is possible to nest forEach-loops, if the `_x` value of an outer loop is needed within an inner one, it has to be assigned to a local variable.

Example:

```
{
    _unit = _x;
    {
        player sidechat format["Unit '%1', weapon: '%2'",_unit,_x]
    } forEach (weapons _x);
} forEach allUnits;
```

In this example, the following happens:

For each of the units in array `allUnits`, do the following:

1. Set `_unit` to the currently iterated unit in array `allUnits`.
2. Iterate on all the weapons of the currently iterated unit, and for each of these weapons, display the text `"Unit '%1', weapon: '%2'"`, where `%1` is replaced by the currently iterated unit and `%2` by the currently iterated weapon of that unit.

3.2.2.7 count-Loop

The count-loop is used for the purposes of doing a conditional count.

```
{
    STATEMENT;
    ...
    EXPRESSION
} count ARRAY;
```

The code block is executed exactly (`count ARRAY`) times.

It is possible to use the special variable `_x` within the code block, which always references the currently iterated item in the array. It is mandatory for the code block to return defined Boolean value - true or false.

Example:

```
_array = [unit1, unit2, unit3];
{
    _x setDamage 1;
    false
} count _array;
```

In this example, the following happens:

1. In the first loop iteration, the statement `unit1 setDamage 1;` is executed.
2. In the second loop iteration, the statement `unit2 setDamage 1;` is executed.
3. In the third loop iteration, the statement `unit3 setDamage 1;` is executed.

While it is possible to nest count-loops, if the `_x` value of an outer loop is needed within an inner one, it has to be assigned to a local variable.

Example:

```
{
    _unit = _x;
    {
        player sidechat format["Unit '%1', weapon: '%2'",_unit,_x];
    } count (weapons _x);
} count allUnits;
```

For each of the units in array `allUnits`, do the following:

1. Set `_unit` to the currently iterated unit in array `allUnits`.
2. Iterate on all the weapons of the currently iterated unit, and for each of these weapons, display the text `"Unit '%1', weapon: '%2'"`, where `%1` is replaced by the currently iterated unit and `%2` by the currently iterated weapon of that unit.

3.2.3 Return Values

Control structures (with exception of `count`) always return the **last expression evaluated** within the structure.

Example:

```
_ret = if (CONDITION) then {VALUE A} else {VALUE B};
```

In this example, the following happens:

- If `CONDITION` is true, then `_ret` is set to `VALUE A`.
- If `CONDITION` is false, then `_ret` is set to `VALUE B`.

3.3 Function Library

The VBS4 Function Library adds a range of preexisting, ready-to-use functions that can be used in scripts. These are not scripting commands, but rather sub-routines created with existing script commands. For the full list of functions in the VBS4 Function Library, see Functions A-Z (https://sqf.bisimulations.com/display/SQF/Functions+A-Z).

Apart from the existing VBS4 Function Library, you can create your own functions. For more information, see Creating Functions (on page 176).

The next line shows a typical function call syntax:

```
_returnvalue = [parameters] call functionName;
```

Syntax Element	Description
`_returnvalue`	The value which the function returns.
`[parameters]`	An Array of parameters passed to the function. If a single parameter is passed, the square brackets can be omitted, but this generally depends on the function definition.
`functionName`	The name of the function.

EXAMPLE

A script calls function `fn_vbs_distance2D` from the VBS4 Function Library which calculates the distance (in meters) between two objects and returns that distance as its return value.

```
_dist = [player, s1] call fn_vbs_distance2D;

hint format["The distance between the player and unit 's1' is %1m",_dist];
```

A script calls function `fn_vbs_lastName` from the VBS4 Function Library which returns the last name of the unit.

```
_pLastName = player call fn_vbs_lastName;

hint format["The player's last name is %1",_pLastName];
```

3.4 Event Handlers

Almost everything that happens in the simulation can be seen as an event, and each simulation event can have a script associated with it. An event handler executes a piece of code when an event (for example, firing a weapon) occurs in VBS4. An event handler is attached to a simulation object and reacts to a specific event that happens to the object. Each event has parameters associated with it. These parameters are returned by the event handler through the _this (on page 27) variable.

To add event handler scripts, use any of the following commands:

- addEventHandler (https://sqf.bisimulations.com/display/SQF/addEventHandler) - Default event handlers, referred to as **Default EH** in the event handler **Type** description in this topic.

```
object addEventHandler ["EventName", {
  // Replace EventName with one of the event-handler names supported by VBS
  /* Add the code that is executed when the event occurs here (event handler
     parameters are accessed through the _this variable)
  */
}];
```

- addGlobalEventHandler (https://sqf.bisimulations.com/display/SQF/addGlobalEventHandler) - Global event handlers, referred to as **Global EH** in the event handler **Type** description in this topic.

```
object addGlobalEventHandler ["GlobalEventName", {
  /* Replace GlobalEventName with one of the global event-handler
     names supported by VBS
  */
  /* Add the code that is executed when the event occurs here (event handler
     parameters are accessed through the _this variable)
  */
}];
```

- addAAREventHandler (https://sqf.bisimulations.com/display/SQF/addAAREventHandler) - AAR event handlers, referred to as **AAR EH** in the event handler **Type** description in this topic.

```
object addAAREventHandler ["AAREventName", {
  /* Replace AAREventName with one of the AAR event-handler
     names supported by VBS
  */
  /* Add the code that is executed when the event occurs here (event handler
     parameters are accessed through the _this variable)
  */
}];
```

- fn_vbs_addSysEventHandler (https://sqf.bisimulations.com/display/SQF/fn_vbs_addSysEventHandler) - System event handlers, referred to as **System EH** in the event handler **Type** description in this topic.

  ```
  [type, code, parameters] call fn_vbs_addSysEventHandler
  ```

  The parameters are:

  - `type` - A string that specifies the type of the event handler to add. For a full list of events, see the System (on page 114) event handler category.
  - `code` - Code to execute when the event occurs. Each code has access to the following variables:
    - `_this` - Data passed by the event itself (contents vary depending on event type).
    - `_index` - Index of event handler being run (useful for removing the event).
    - `_data` - Extra data passed, when the event handler is added.
  - `parameters` - Data to pass to the `code`, using `_data`.

  NOTE

  You can define event handlers for custom events as system event handlers. For more information, see Event Handlers Configuration in the VBS Developer Reference.

EXAMPLE

When a specific vehicle fires a weapon, a message is displayed.

In this case, the event in question is called "`Fired`" (for instance, you can add the following code to the **Initialization Statements** of a vehicle called `vehicle1`):

```
vehicle1 addEventHandler ["Fired", {
    _vehicle1 = _this select 0;
    hint "Vehicle fired a weapon!";
}];
```

The first parameter (in position 0 of the `_this` array) is saved to `_vehicle1`.

The event-handling code is added to the `vehicle1` object. This code has its own scope and it does not recognize variables from outside of this scope. This is why variable `_vehicle1` is not the same as `vehicle1` to which the event handler is attached. For more information, see addEventHandler (https://sqf.bisimulations.com/display/SQF/addEventHandler).

Event handlers can be used both in scripts and configuration (for more information about using event handlers in configuration, see Event Handlers Configuration in the VBS Developer Reference).

The `_this` parameter of the event handler contains the event-handler **Return Values**.

The **Return Values** in each event handler can be used as a parameters list, with the params (https://sqf.bisimulations.com/display/SQF/params) command:

```
object addEventHandler ["EventName", {
   params ["parameter1", "parameter2", ...];
}];
```

EXAMPLE

For example, the Fired (on page 105) event handler has the following **Return Values**:

[unit, weapon, muzzle, mode, ammo, magazine, projectile]

Therefore, the params (https://sqf.bisimulations.com/display/SQF/params) command call inside addEventHandler (https://sqf.bisimulations.com/display/SQF/addEventHandler) is:

```
object addEventHandler ["Fired", {
   params [
     "_unit", "_weapon", "_muzzle", "_mode",
     "_ammo", "_magazine", "_projectile", "_gunner"
   ];
}];
```

The event handlers available in VBS4 are listed by the following categories:

- AAR (on the next page) - Event handlers that are activated during AAR.
- Collision / Damage (on page 67) - Event handlers that are activated when an object collides or gets damaged.
- Editor (on page 76) - Event handlers that are activated when objects are manipulated in the Editor (Execute Mode).
- Units (on page 79) - Event handlers that are activated when something happens to characters (for example, soldiers or civilians).
- Vehicles (on page 91) - Event handlers that are activated when something happens to vehicles.
- Weapons / Shots (on page 102) - Event handlers that are activated when weapons are manipulated or when a projectile (for example, a bullet or a missile) is shot.
- Multiplayer (on page 110) - Event handlers that are activated in a multiplayer session.
- UI Event Handlers (on page 125) - Event handlers used by the VBS4 UI.

- Camera (on page 112) - Event handlers associated with the camera operation.
- Communication (on page 112) - Event handlers used in communication between player units, players and AI units, and AI units.
- System (on page 114) - Event handlers used by the VBS system.
- Other (on page 118) - Other miscellaneous event handlers.

3.4.1 AAR

The following AAR event handlers are available:

- Edited (below)
- Load (on the next page)
- RecordedVariable (on the next page)
- RecordingStarted (on the next page)
- Saved (on page 66)

3.4.1.1 Edited

Triggered when the AAR data entries are edited.

There are three edit operations available:

- `Append` - Called when data entries from the appended AAR timeline are appended to the current AAR.
- `Delete` - Called when deleting a selected time range from AAR. Data entries that are located outside of the selected time slice are copied to the new AAR file, while data entries located between the start and end time of the selection are left out.
- `Trim` - Called when a time range gets trimmed from AAR. Data entries that lie within the selected time slice are copied to the new AAR files, while entries that extend outside the selected time slice are trimmed.

Type:

Parsed array, AAR EH

Returned Values:

["append", appendFrom, originPath, targetPath]

- appendFrom: String - Path to AAR files that have been appended.
- originPath: String - Where to take the AAR data from.
- targetPath: String - Where to put the modified AAR data to.

["delete" or "trim", startTime, endTime, originPath, targetPath]

- startTime: Number - Start time of the selection.
- endTime: Number - End time of the selection.
- originPath: String - Where to take the AAR data from.
- targetPath: String - Where to put the modified AAR data to.

3.4.1.2 Load

Triggered when an AAR recording is loaded in the Editor (Execute Mode).

Type:

Parsed array, AAR EH

Returned Values:

[path]

- path: String - The full path to the AAR recording.

3.4.1.3 RecordedVariable

Triggered when a variable value is stored during the recording using the AARRecordValue (https://sqf.bisimulations.com/display/SQF/AARRecordValue) command.

Type:

Parsed array, AAR EH

Returned Values:

[object, variable, value]

- object: Object - The object associated with the variable value.
- variable: String - The variable name.
- value: String - The variable value.

3.4.1.4 RecordingStarted

Triggered every time AAR recording starts.

Type:

N/A

Returned Values:

N/A

3.4.1.5 Saved

Triggered when an AAR recording is saved in the Editor (Execute Mode).

Type:

Parsed array, AAR EH

Returned Values:

[path]

- path: String - The full path to the AAR recording.

3.4.2 Collision / Damage

The following collision / damage event handlers event handlers are available:

3.4.2.1 BeforeKilled

Triggered just before a unit is killed.

Does not work with objects, and is only triggered after destruction, if used with vehicles.

Type:

Parsed array, Default EH

Returned Values:

[object]

- object: Object - Unit that is about to die.

3.4.2.2 CollisionEnd

Triggered when a collision between two objects (either PhysX or non-PhysX) ends.

Type:

Parsed array, Default EH

Returned Values:

When not using collision volumes, the following values are returned:

[object, []]

- object: Object - Object the event handler is attached to

> **NOTE**
> An empty array [] is returned. It can be discarded.

When using collision volumes, the following values are returned:

[object, collidingObj]

- object: Object - Object with the volume defined.
- collidingObj: Object - Optional other object, if the collision volume collided with another object, missing if collided with terrain.

3.4.2.3 CollisionStart

Triggered when a collision between two objects (either PhysX or non-PhysX) starts.

Type:

Parsed array, Default EH

Returned Values:

When not using collision volumes, the following values are returned:

[owner, collisions]

- owner: Object - Object the event handler is attached to.
- collisions: Array - Array of collisions that are occurring. Each element is:

  [collidingObj, [collidingObjComponents], [ownerComponents], [contactPointCoords]]

  - collidingObj: Object - Object that the owner collided with.
  - collidingObjComponents: Array - Array of component names (referring to named selections in the Geometry LOD) that the collision is affecting on the colliding object.
  - ownerComponents: Array - Array of component names (referring to named selections in the Geometry LOD) that the collision is affecting on the owner.
  - contactPointCoords: Array - An array of contact point coordinates (absolute position) in world space.

When using collision volumes, the following values are returned:

[object, collisionVolID, collidingObj, collidingObjVolID]

- object: Object - Object with the volume defined.
- collisionVolID: Number - Collision volume ID that triggered the event.
- collidingObj: Object - Optional other object, if the collision volume collided with another object, missing if collided with terrain.
- collidingObjVolID: Number - Optional other collision volume ID, if the collision volume collided with another one.

3.4.2.4 Dammaged

Triggered when a unit is damaged. Works with all vehicles not only people.

If simultaneous damage occurs (for example, because of a grenade), **Dammaged** might be triggered several times.

Type:

Parsed array, Global EH

Returned Values:

[unit, selectionName, damage]

- unit: Object - Object event handler is assigned to.
- selectionName: String - Name of the selection where the unit is damaged.
- damage: Number - Resulting level of damage.

3.4.2.5 DamagedHitPart

Triggered when an object is damaged. Is not triggered if damage is set using the setDamage (https://sqf.bisimulations.com/display/SQF/setDamage) script command. If simultaneous damage occurs (for example, from a grenade), **DamagedHitPart** is triggered several times.

Type:

Parsed array, Default EH

Returned Values:

[target, shooter, hitzone,totaldamage,hitdamage,structdamage,hitzonecenter,hitzoneradius,selections,surfaces]

- target: Object - Object that gets damaged.
- shooter: Object - Unit that inflicts the damage. If injured by a vehicle impact or a fall, the target itself is returned. With IEDs or artillery, either the target itself is returned, or the null object if further away from the explosion. In the case of explosives planted by someone (for example, satchel charges), either the planting unit is returned or the target unit itself, if further away.
- hitzone: String - Hit zone that is damaged.
- totaldamage: Number - Accumulated damage to hit zone.
- hitdamage: Number - Amount of damage hit zone receives from this hit.
- structdamage: Number - Damage increase to object itself, received by this hit.
- hitzonecenter: Position3D - Position of hit zone center in model space.
- hitzoneradius: Number - Radius of the hit zone.
- selections: Array of Strings - Names of selections the hit zone belongs to.
- surfaces: Array of Strings - Surface types of hit zone.

3.4.2.6 EpeContact

Triggered (every frame) while a PhysX object is in contact with another PhysX object.

Type:

Parsed array, Default EH

Returned Values:

[owner, object, ownsel, objsel, force]

- owner: Object - Object the event handler is attached to.
- object: Object - Object that is colliding with owner object (may be objNull, if a moving object touching the ground).

- ownsel: String - Selection name of owner object that is being touched (empty if no selections defined).
- objsel: String - Selection name of colliding object that is being touched (empty if no selections defined).
- force: Number - Amount of force (in N) of collision. Can be 0 during the contact release.

3.4.2.7 EpeContactStart / EpeContactEnd

Triggered when a contact between two PhysX objects is initiated (or ended).

Type:

Parsed array, Default EH

Returned Values:

[owner, object, ownsel, objsel, force]

- owner: Object - Object the event handler is attached to.
- object: Object - Object that is colliding with owner object (may be objNull is a moving object is touching the ground).
- ownsel: String - Selection name of owner object that is being touched (empty if no selections defined).
- objsel: String - Selection name of colliding object that is being touched (empty if no selections defined).
- force: Number - Amount of force (in N) of collision.

3.4.2.8 FreeFall

Triggered when a unit drops from a height of more than 20 meters (for example, falling off a building or after being thrown into the air by an explosion). Is not triggered on parachute ejections.

Type:

Parsed array, Default EH

Returned Values:

[unit]

- unit: Object - Object the event handler is assigned to.

3.4.2.9 HandleDamage

Triggered when the damage value of an object changes, and allows the adjustment of the damage value inside the event handler code section.

May fire multiple times, if multiple sections are affected, or if damage is caused by a direct or indirect hit (for example, using grenades).

The setDamage (https://sqf.bisimulations.com/display/SQF/setDamage) script command should not be used in the code section of this event handler, as it can cause a dead loop (with the engine endlessly resetting and reacting to the changes of damage). This event handler may be triggered multiple times, damage values set by one event may be overwritten by a later one, if these happen within the same frame. While full damage cannot be reversed, if only partial damage is applied, it can be overwritten by a later event (for example, if the car1 example applied a damage of only 0.9, a damage event to the "hull" could negate the originally applied damage).

Type:

Parsed array, Default EH

Returned Values:

[target, selection, damage, shooter, bullet]

- target: Object - Object event handler is attached to (can be unit, vehicle or object).
- selection: String - Name of selection that received damage. Empty string ("") for overall damage. "?" for unknown / unnamed section.
- damage: Number - Damage value inflicted by this hit.
- shooter: Object - Source of damage. If injured by a vehicle impact or a fall the target itself is returned. For explosions or vehicles hitting obstacles, the null object is returned.
- bullet: String - Class name of projectile that caused damage. Empty string ("") if damage is caused by different means (for example, a fall or impact).

The code section of this event handler can contain a return value which specifies the damage that should be applied to the target. If no return value is given, no damage is applied (same effect as passing a '0').

To use the original damage that would have been caused by the hit, `_this select 2` should be returned.

EXAMPLE

```
unit1 addEventHandler ["HandleDamage","hint 'ouch'; _this select 2"]
```

A return value of '1' destroys the attached object. The following example does this whenever the vehicle engine is hit, any other hit does not cause damage:

```
car1 addEventHandler ["HandleDamage","if ('engine' in _this) then {1} else {0}"]
```

3.4.2.10 HandleHeal

Triggered when a unit is healed by another unit. Does not fire when healing in an ambulance or tent.

If the code returns nothing (or false), the normal healing process is activated. If it returns true, **HandleHeal** continues to be triggered until either the target health is set to full (using the setDamage (https://sqf.bisimulations.com/display/SQF/setDamage) script command) or the AISFinishHeal (https://sqf.bisimulations.com/display/SQF/AISFinishHeal) script command is executed.

Type:

Parsed array, Default EH

Returned Values:

[target, healer, isMedic]

- target: Unit - Unit event handler is attached to.
- healer: String - Unit that is healing the target.
- isMedic: Boolean - Is true if healer is a medic.

3.4.2.11 Hit

Triggered when a unit is hit / damaged. Not always triggered when a unit is killed by a hit.

Most of the time, only the Killed (on the next page) event handler is triggered when a unit dies from a hit.

The **Hit** event handler does not necessarily get triggered if only minor damage occurs (for example, firing a bullet at a tank), even though the damage increases.

Type:

Parsed array, Local EH

Returned Values:

[unit, causedBy, damage]

- unit: Object - Object event handler is assigned to.
- causedBy: Object - Object that caused the damage. Contains the unit itself in the case of collisions.
- damage: Number - Level of damage caused by the hit.

3.4.2.12 HitPart

Triggered when an object it is added to is hit by a weapon.

Returns the position and component that is hit on the object.

Type:

Parsed array, Default EH

Returned Values:

[target, shooter, bullet, position, velocity, selection, ammo, direction, radius, surface, direct]

- target: Object - Object that is fired at.
- shooter: Object - Unit that fires the shot.
- bullet: Object - Object that is fired.
- position: Position3D - Position the bullet impacts.
- velocity: Vector - 3D speed at which bullet impacts.
- selection: Array - Array of Strings with the named selections of the object that is hit.
- ammo: Array - Ammunition info:

  [hit value, indirect hit value, indirect hit range, explosive damage, ammo class name]

  OR, if there is no shot object:

  [impulse value on object collided with,0,0,0]

  - direction: Vector3D - Vector that is orthogonal (perpendicular) to the surface struck. For example, if a wall is hit, the vector would be pointing out from the wall at a 90 degree angle.
  - radius: Number - Radius (size) of component hit.
  - surface: String - Surface type struck.
  - direct: Boolean - True if object is hit directly, false if it is hit by indirect / splash damage.

3.4.2.13 Killed

Triggered when a unit is killed.

Type:

Parsed array, Local EH

Returned Values:

[unit, killer]

- unit: Object - Object event handler is assigned to.
- killer: Object - Object that killed the unit. Contains the unit itself in case of collisions.

3.4.2.14 turretDriveConnected

Triggered when contact between a PhysX object and a configured vehicle turret is initiated.

Type:

Parsed array, Default EH

Returned Values:

[owner, turret, force]

- owner: Object - Object the event handler is attached to.
- turret: Array - Path to the turret involved in the collision.
- force: Number - Amount of force (in N) of collision.

3.4.2.15 turretDriveDisconnected

Triggered when contact between a PhysX object and a configured vehicle turret is ended.

Type:

Parsed array, Default EH

Returned Values:

[owner, turret, force]

- owner: Object - Object the event handler is attached to.
- turret: Array - Path to the turret involved in the collision.
- force: Number - Amount of force (in N) of collision.

3.4.2.16 WheelCollisionEnd

This event occurs when wheel collision with the target object ends (also see WheelCollisionStart (on the next page)).

- The event handler only executes for raycast wheels (for example, it does not work for **M119 - 105mm Howitzer**).

Type:

Parsed array, Default EH

Returned Values:

[vehicle, selection, trgObject, hitPoint]

- vehicle: Object - The vehicle.
- selection: String - Wheel selection name (for example, "wheel_1_1_damper").
- trgObject: Object - Trigger object (the one that is hit by the wheel).
- hitPoint: String - Hit point on the object that is hit by the wheel (or empty string if there is no hit point defined for the hit component).

3.4.2.17 WheelCollisionStart

This event occurs when wheel collision with the target object starts (also see WheelCollisionEnd (on the previous page)).

- The event handler only executes for raycast wheels (for example, it does not work for **M119 - 105mm Howitzer**).
- Some events may be skipped if the vehicle drives fast over small objects with the event handler turned on.
- The events get reported for each wheel separately, but it may be reported only once for each collision object and not for each collision object component that the wheel touches.

Type:

Parsed array, Default EH

Returned Values:

[vehicle, selection, trgObject, hitPoint]

- vehicle: Object - The vehicle.
- selection: String - Wheel selection name (for example, "wheel_1_1_damper").
- trgObject: Object - Trigger object (the one that is hit by the wheel).
- hitPoint: String - Hit point on the object that is hit by the wheel (or empty string if there is no hit point defined for the hit component).

3.4.3 Editor

The following Editor event handlers are available:

- Delete (on the next page)
- Deleted (on the next page)
- GroupChangedRTE (on the next page)
- Local (on page 78)
- onDragStart (on page 78)

- Paste (on the next page)
- PlayerChangedRTE (on page 79)

3.4.3.1 Delete

Triggered when an object is deleted (also see Deleted (below)).

> **NOTE**
> This event handler can only be added to scripts using the addEventHandler (https://sqf.bisimulations.com/display/SQF/addEventHandler) script command.

Type:

Parsed array, Default EH

Returned Values:

[object]

- object: Object - Deleted unit.

3.4.3.2 Deleted

Triggered just before an object is deleted.

Examples of what could delete an object include the deleteVehicle (https://sqf.bisimulations.com/display/SQF/deleteVehicle) script command, the Editor (Execute Mode), or through VBS Gateway.

If used in a `config.cpp`, an error occurs during the packing of the content. This event handler should be used within a `config.cpp`.

Type:

Parsed array, Default EH

Returned Values:

[object]

- object: Object - Object that was deleted.

3.4.3.3 GroupChangedRTE

Added to a SIDE, not an object. Triggered only if the Editor (Execute Mode) is open, when a change occurs to a specific group or subgroup status.

Also triggered on creation of a new group, or deletion of an existing group.

Type:

Parsed array, Default EH

Returned Values:

[group]

- group: Group - Group for which the change occurs.

3.4.3.4 Local

Checks if a given unit is local in multi-player games.

Type:

Parsed array, Default EH

Returned Values:

[object]

- object: Object - Local object in multi-player games.

3.4.3.5 onDragStart

Triggered when starting to drag an object.

Type:

Parsed array, Default EH

Returned Values:

[object, unit]

- object: Object - The object being dragged.
- unit: Object - The unit dragging the object.

3.4.3.6 Paste

Triggered when an object has been copied and pasted in the Editor.

If a vehicle is copied, the **Paste** event handler is triggered once for the vehicle, and once for every crew member.

Type:

Parsed array, Default EH

Returned Values:

[original, copy]

- original: Object - Object that is copied.
- copy: Object - Object that is created.

3.4.3.7 PlayerChangedRTE

Added to a SIDE, not an object. Can also be used on units if unit ownership is changed by the player.

Type:

Parsed array, Default EH

Returned Values:

[unit, mode]

- unit: Object - Unit / vehicle the event handler is assigned to.
- mode: String - Action that caused the change, the options are:
  - "CONNECT": Player occupied unit using MP JIP (Join-in-progress).
  - "DISCONNECT": Player disconnected from MP session.
  - "SELECTEDPLAYER": Player switched into a unit.
  - "UNSELECTEDPLAYER": Player switched out of a unit.

When switching unit (for example, using selectPlayer (https://sqf.bisimulations.com/display/SQF/selectPlayer)) both events always fire — first "UNSELECTEDPLAYER" on the unit that is being left, and then "SELECTEDPLAYER" on the unit that is being occupied.

3.4.4 Units

The following units event handlers are available:

3.4.4.1 AnimChanged (for a unit)

Triggered every time a new unit animation starts.

Type:

Parsed array, Global EH

Returned Values:

[unit, anim, currentPhase, targetPhase]

- unit: Object - Object the event handler is assigned to.
- anim: String - Name of the animation that started.
- currentPhase: Number - Current animation phase.
- targetPhase: Number - Desired animation phase.

3.4.4.2 AnimDone (for a unit)

Triggered every time a new unit animation finishes.

Type:

Parsed array, Global EH

Returned Values:

[unit, anim]

- unit: Object - Object event handler is assigned to.
- anim: String - Name of the animation that has finished.

3.4.4.3 AnimStateChanged

Triggered when a unit animation state changes. Unlike AnimChanged (for a unit) (on the previous page) and AnimDone (for a unit) (on the previous page), it is triggered for all animation states in a sequence.

Type:

Parsed array, Default EH

Returned Values:

[unit, anim]

- unit: Object - Object the event handler is assigned to.
- anim: String - Name of the animation that has started.

3.4.4.4 BeforeCreated

This event handler is added to unit configuration and is triggered just before a unit is created in a scenario. The return value is the type of unit to create as a string.

This event handler can only be used in the `config.cpp`.

Type:

Parsed array, Default EH

Returned Values:

[name, type, player, client ID, playable, position, azimuth]

- name - Variable name, if set (for example, "unit1"). "unit_0" when no variable name is set.
- type: - For example, "VBS2_AU_Army_IFVCrew_W_F88c".
- player - true / false.
- client ID - Unique ID of the player client. Some value for non-players (for example, -1).
- playable - true / false.
- position - ASL.
- azimuth - degrees.

3.4.4.5 CoverReached

Triggered when a unit in combat mode reaches a cover position.

> **FEATURE NOTICE**
> Temporarily disabled for AI. See One AI in the VBS4 Release Notes.

Type:

Parsed array, Default EH

Returned Values:

[unit, coverType, coverPosition, coverEntryPosition]

- unit: Object - Object the event handler is assigned to.
- coverType: String - Type of object used as cover, the options are:
  - 0: None (no cover).
  - 1: RightCornerHigh (high vertical edge, such as a building edge - use standing + leaning).
  - 2: RightCornerMiddle (middle vertical edge, such as a fence edge - use kneel + leaning).
  - 3: RightCornerLow (low vertical edge - use prone + sideways movement).
  - 4: LeftCornerHigh (high vertical edge, such as a building edge - use standing + leaning).
  - 5: LeftCornerMiddle (middle vertical edge, such as a fence edge - use kneel + leaning).
  - 6: LeftCornerLow (low vertical edge - use prone + sideways movement).
  - 7: EdgeMiddle (horizontal edge, such as a fence - use kneel + stand).
  - 8: EdgeLow (low horizontal edge, such as a fence - use prone + kneel).
  - 9: Wall (wall - total protection from one side, no fire over the obstacle).
- coverPosition: PositionASL - Position of cover (at about eye level).
- coverEntryPosition: PositionASL - Approximate position from which cover position is entered. Roughly one meter away from vertical cover (for example, walls), similar to **coverPosition** for ground cover (such as grass).

3.4.4.6 Danger

Triggered when an AI unit encounters a danger indicator.

> **FEATURE NOTICE**
>
> Temporarily disabled for AI. See One AI in the VBS4 Release Notes.

Type:

Parsed array, Default EH

Returned Values:

[unit, dangerType, dangerPosition, until, cause, script]

- unit: Object - Object the event handler is assigned to
- dangerType: String - The cause of the danger, the options are:
  - "ENEMYDETECTED" - The first enemy detected. Triggers only at very close distances if unit is in "NO FIRE" combat mode.
  - "FIRE" - Bullet impact noticed, only triggered if the unit combat mode is SAFE or AWARE.
  - "HIT" - Unit is hit.
  - "ENEMYNEAR" - Enemy very close, < 10m.
  - "EXPLOSION" - Explosion detected, caused by bombs or bullet impacts.
  - "DEADBODYGROUP" - Dead soldier from same group found.
  - "DEADBODY" - Other dead person found.
  - "SCREAM" - Hit soldier screaming - non-functional.
  - "CANFIRE" - New opportunity to fire on an enemy target.
- dangerPosition: PositionASL - Position of object that triggers the danger event.
- until: Number - Duration (seconds) of how long the danger is expected to persist.
- cause: Object - Object that triggers the danger event.

  For "DEADBODY" events, returns the killer, but only if the killer is an enemy within view distance (actual visibility does not matter). If killed by either friendlies or using scripting, or out of range, objNull is returned.
- script: Boolean - If true, the event was triggered by command setDanger (https://sqf.bisimulations.com/display/SQF/setDanger).

3.4.4.7 GestureChanged

Triggered when a new gesture is started or completed.

Type:

Parsed array, Default EH

Returned Values:

[unit, gesture]

- unit: Object - Object the event handler is assigned to.
- gesture: String - Name of the gesture that was started. When completed, event returns "<none>".

3.4.4.8 GestureDone

Triggered when a gesture is completed.

Type:

Parsed array, Default EH

Returned Values:

[unit, gesture]

- unit: Object - Object the event handler is assigned to.
- gesture: String - Name of the gesture that was completed.

3.4.4.9 GroupChanged

Added to a SIDE, not an object. Triggered when a change occurs to a specific group or subgroup's status.

Also triggered on creation of new group, or deletion of an existing group.

Type:

Parsed array, Default EH

Returned Values:

[group]

- group: Group - Group for which the change occurs.

3.4.4.10 HandleIdentity

Triggered when a new identity is created for a unit.

Type:

Parsed array, Default EH

Returned Values:

[unit]

- unit: Object - Unit the event handler is attached to.

3.4.4.11 LoadOutChanged

Triggered whenever the inventory is opened and closed to change the loadout.

Type:

Parsed array, Default EH

Returned Values:

[object]

- object: Object - Object that has its equipment changed.

3.4.4.12 Moved

A unit / vehicle / object has moved (either by itself, or it was dragged in the Editor).

If a unit is inside a vehicle, and that vehicle moves, then only the event for the vehicle is triggered.

Type:

Parsed array, Default EH

Returned Values:

[unit, oldPos, oldDir, oldUp, oldDirS, newPos, newDir, newUp, newDirS]

- unit: Object - Entity that is moved.
- oldPos: PositionASL - Old position.
- oldDir: Vector3D - Old vectorDir.
- oldUp: Vector3D - Old vectorUp.
- oldDirS: Vector3D - Old direction aside (perpendicular to the y and z-axis, pointing along the x-axis to the right).
- newPos: PositionASL - New position.
- newDir: Vector3D - New vectorDir.
- newUp: Vector3D - New vectorUp.
- newDirS: Vector3D - New direction aside.

3.4.4.13 NameChanged

Added to a SIDE, not an object. A unit name is assigned using setUnitName (https://sqf.bisimulations.com/display/SQF/setUnitName) script commands.

This event is triggered even if the new name is the same as the old one.

Type:

Parsed array, Default EH

Returned Values:

[unit, newName, oldName]

- unit: Object - Unit whose name is changed.
- newName: String - Unit new name.
- oldName: String - Unit old name.

3.4.4.14 PathPlanFailed

Triggered when a unit / vehicle is not able to reach a assigned destination (assignment can be done using a waypoint, move commands, or group order).

> **FEATURE NOTICE**
> Temporarily disabled for AI. See One AI in the VBS4 Release Notes.

Type:

Parsed array, Default EH

Returned Values:

[unit, destination, retry]

- unit: Object - Unit / vehicle the event handler is assigned to.
- destination: Position3D - Position that cannot be reached.
- retry: Boolean - If true the unit retries reaching the destination (normally it retries twice before giving up, at which point false is returned).

3.4.4.15 PlayerChanged

Added to a SIDE, not an object. Can also be used on units if unit ownership is changed by the player.

Type:

Parsed array, Default EH

Returned Values:

[unit, mode]

- unit: Object - Unit / vehicle the event handler is assigned to.
- mode: String - Action that caused the change, the options are:
  - "CONNECT": Player occupied unit using MP JIP (Join-in-progress).
  - "DISCONNECT": Player disconnected from MP session.
  - "SELECTEDPLAYER": Player switched into a unit.
  - "UNSELECTEDPLAYER": Player switched out of a unit.

When switching unit (for example, using selectPlayer (https://sqf.bisimulations.com/display/SQF/selectPlayer)) both events always fire — first "UNSELECTEDPLAYER" on the unit that is being left, and then "SELECTEDPLAYER" on the unit that is being occupied.

3.4.4.16 PositionChangedMan

Triggered when a person either enters, exits, or changes positions inside a vehicle. Also see PositionChanged (on page 99), for a vehicle-based event handler.

Type:

Parsed array, Default EH

Returned Values:

[person, vehicle, positionFromArray, positionToArray]

- person: Object - Unit the event handler was attached to.
- vehicle: Object - Vehicle that was entered / exited / moved in.
- positionFromArray / positionToArray: Array - Indicates the position the unit was in, and the position they moved to.

  Format of both arrays is: **[type, turretPath or cargoIndex]**

  If vehicle is entered as the driver, then the from / to arrays are reversed, returns [1,0],[0,0], instead of [0,0],[1,0]
  - type: Number - 0 = not in vehicle; 1 = driver; 2 = cargo position, 3 = turret.
  - turretPath: Array - Path to turret.
  - cargoIndex: Number - Cargo slot index.

3.4.4.17 ResetFatigue

Triggered when unit fatigue level is reset using resetFatigue (https://sqf.bisimulations.com/display/SQF/resetFatigue) script command.

Type:

Parsed array, Default EH

Returned Values:

[unit, fatigue]

- unit: Object - The unit whose fatigue level is reset.
- fatigue: Number - The fatigue level before it is reset.

3.4.4.18 Respawn

Triggered when an object respawns.

Type:

Parsed array, Default EH

Returned Values:

[new unit, old unit]

- new unit: Object - The player new unit, after respawning.
- old unit: Object - The player old unit, before respawning.

3.4.4.19 Revived

Triggered after a unit is revived.

Type:

Parsed array, Default EH

Returned Values:

[man]

- man: Unit - Unit that is revived.

3.4.4.20 Suppressed

Triggered each time a unit is 'suppressed' by incoming rounds.

> **FEATURE NOTICE**
> Temporarily disabled for AI. See One AI in the VBS4 Release Notes.

Type:

Parsed array, Default EH

Returned Values:

[unit, who fired, type, distance, position]

- unit: Object - The unit that is suppressed.
- who fired: Object - Object that fired the suppressing round.
- type: String - Class name of the round that caused suppression.
- distance: Number - Nearest distance that round came to the suppressed unit.
- position: Position3D - AGL position where the round gets closest to the suppressed unit.

3.4.4.21 UPRPlaybackEnd

Triggered when the playback of a UPR recoding ends.

Type:

Parsed array, Default EH

Returned Values:

[unit, stopped]

- unit: Unit - AI unit to which the event handler is added.
- stopped: Boolean - If true, then the playback is stopped by the unit itself (for example, because of enemy contact), if false, then the playback is completed or stopped using the UPRStopPlayback (https://sqf.bisimulations.com/display/SQF/UPRStopPlayback) script command.

3.4.4.22 UPRRecordEnd

Triggered when a UPR recording ends (the recording is finished or stopped, using the UPRSaveRecording (https://sqf.bisimulations.com/display/SQF/UPRSaveRecording) script command, or is aborted for other reasons - for example, the unit dies or enters a vehicle).

Type:

Parsed array, Default EH

Returned Values:

[unit, filename, stopped]

- unit: Unit - Unit to which the event handler is added.
- filename: String - Name of the UPR file.
- stopped: Boolean - If true, playback was finished or stopped using the UPRSaveRecording (https://sqf.bisimulations.com/display/SQF/UPRSaveRecording) script command, if false, the recording is aborted for other reasons (for example, the unit dies or enters a vehicle).

3.4.4.23 WaypointComplete

This event must be added to a Group, not an individual unit or vehicle. It is triggered when the group completes a waypoint.

> **NOTE**
> After the waypoint is completed, it is deleted.

Type:

Parsed array, Default EH

Returned Values:

[group, index, waypoint]

- group: Group - Group that triggered event was added to.
- index: Number - Index of waypoint that was completed.
- waypoint: Object - Completed waypoint.

3.4.5 Vehicles

The following vehicles event handlers are available:

3.4.5.1 AfterGetInMan / AfterGetOutMan

Triggered after a person gets in or out of a vehicle.

Type:

Parsed array, Default EH

Returned Values:

[person, vehicle, position, unit]

- person: Object - The person that gets into or out of a vehicle.
- vehicle: Object - Vehicle the event handler is assigned to.
- position: String - Can be "driver", "gunner" or "cargo".
- unit: Object - Unit that leaves or enters a vehicle.

3.4.5.2 AnimChanged (for a vehicle)

Triggered every time a new vehicle animation starts.

Type:

Parsed array, Global EH

Returned Values:

[unit, anim, currentPhase, targetPhase]

- unit: Object - Object the event handler is assigned to.
- anim: String - Name of the animation that started.
- currentPhase: Number - Current animation phase.
- targetPhase: Number - Desired animation phase.

3.4.5.3 AnimDone (for a vehicle)

Triggered every time a new vehicle animation finishes.

Type:

Parsed array, Global EH

Returned Values:

[unit, anim]

- unit: Object - Object event handler is assigned to.
- anim: String - Name of the animation that has finished.

3.4.5.4 AfterGetIn / AfterGetOut

Triggered after a unit gets in or out of a vehicle.

Type:

Parsed array, Default EH

Returned Values:

[vehicle, position, unit]

- vehicle: Object - Vehicle the event handler is assigned to.
- position: String - Vehicle position. Can be "driver", "gunner" or "cargo".
- unit: Object - Unit that leaves or enters the vehicle.

3.4.5.5 BeforeGetIn / BeforeGetOut

Triggered just before a unit gets into or out of a vehicle.

Fires after the unit has entered / left the vehicle (it is the same as the AfterGetIn / AfterGetOut (above)).

Type:

Parsed array, Global EH

Returned Values:

[vehicle, position, unit]

- vehicle: Object - Vehicle the event handler is assigned to.
- position: String - Can be "driver", "gunner" or "cargo".
- unit: Object - Unit that leaves or enters the vehicle.

3.4.5.6 BeforeGetInMan / BeforeGetOutMan

Triggered just before a unit gets in or out of a vehicle.

Fires after the enters / leaves the vehicle (it is the same as the AfterGetIn / AfterGetOut (above)).

Type:

Parsed array, Default EH

Returned Values:

[person, vehicle, position, unit]

- vehicle: Object - Vehicle the event handler is assigned to.
- position: String - Vehicle position. Can be "driver", "gunner", or "cargo".
- unit: Object - Unit that leaves or enters the vehicle.

3.4.5.7 CargoChanged

Triggered whenever a weapon or magazine is added / removed from a vehicle cargo space.

Could occur as a result of script commands (such as addWeaponCargo (https://sqf.bisimulations.com/display/SQF/addWeaponCargo) or clearMagazineCargo (https://sqf.bisimulations.com/display/SQF/clearMagazineCargo)), or it could happen as a result of in-game actions such as an AI soldier taking out gear from a vehicle.

The event handler has the following considerations:

1. The magazines of an infantry unit are in their equipped space (does not trigger the event handler); when the magazines are moved in a vehicle, they are in the vehicle cargo space (triggers the event handler).
2. When a weapon is moved from the primary slot of an infantry unit to their backpack, it is considered as moving the weapon from the equipped space into the cargo space of the infantry unit (triggers the event handler).
3. For infantry units, magazine space is not considered cargo space, even if the infantry unit has a backpack (moving a magazine to an infantry unit does not trigger the event handler).

Type:

Parsed array, Default EH

Returned Values:

[object]

- object: Object - Object that has its cargo changed.

3.4.5.8 Created

Similar to Init (on page 120), but is called after Init (on page 120), and after side and license plate have been applied to a vehicle.

Type:

Parsed array, Default EH

Returned Values:

[unit]

- unit: Object - Object event handler is assigned to

3.4.5.9 Engine

Triggered when the engine of a vehicle is turned on / off.

Type:

Parsed array, Global EH

Returned Values:

[vehicle, engineState]

- vehicle: Object - Vehicle the event handler is assigned to.
- engineState: Boolean - True when the engine is turned on, false when turned off.

3.4.5.10 Fuel

Triggered when a unit fuel status changes between completely empty / not empty (only useful if the event handler is assigned to a vehicle).

Type:

Parsed array, Global EH

Returned Values:

[vehicle, fuelState]

- vehicle: Object - Vehicle the event handler is assigned to.
- fuelState: Boolean - 0 when no fuel, 1 when the fuel tank is full.

3.4.5.11 Gear

Triggered when a unit lowers / retracts landing gear (only useful if the event handler is assigned to a member of the class `Plane`).

Type:

Parsed array, Global EH

Returned Values:

[vehicle, gearState]

- vehicle: Object - Vehicle the event handler is assigned to.
- gearState: Boolean - True when the gear is lowered, false when retracted.

3.4.5.12 GetIn

Triggered when a unit enters the object (only useful when the event handler is assigned to a vehicle).

Does not trigger upon a change of positions within the same vehicle. Also not triggered by the `moveInX` script command (for example, moveInDriver (https://sqf.bisimulations.com/display/SQF/moveInDriver)).

Type:

Parsed array, Global EH

Returned Values:

[vehicle, position, unit]

- vehicle: Object - Vehicle the event handler is assigned to.
- position: String - Can be either "driver", "gunner", "commander" or "cargo".
- unit: Object - Unit that entered the vehicle.

3.4.5.13 GetOut

Triggered when a unit gets out of a vehicle (only useful when the event handler is assigned to a vehicle).

It does not trigger upon a change of positions within the same vehicle.

Type:

Parsed array, Global EH

Returned Values:

[vehicle, position, unit]

- vehicle: Object - Vehicle the event handler is assigned to.
- position: String - Can be either "driver", "gunner", "commander", or "cargo".
- unit: Object - Unit that enters the vehicle.

3.4.5.14 GetInMan

Triggered when a unit enters a vehicle. Does not trigger upon a change of positions within the same vehicle.

> **NOTE**
> This event is attached to a person, not a vehicle like the GetIn (on the previous page) event.

Type:

Parsed array, Global EH

Returned Values:

[vehicle, position, unit]

- vehicle: Object - Vehicle the event handler is assigned to.
- position: String - Can be either "driver", "gunner", "commander" , or "cargo".
- unit: Object - Unit that entered the vehicle.

3.4.5.15 GetOutMan

Triggered when a unit gets out of a vehicle. It does not trigger upon a change of positions within the same vehicle.

> **NOTE**
>
> This event is attached to a person, not a vehicle like the GetOut (on the previous page) event.

Type:

Parsed array, Global EH

Returned Values:

[vehicle, position, unit]

- vehicle: Object - Vehicle the event handler is assigned to.
- position: String - Can be either "driver", "gunner", "commander", or "cargo".
- unit: Object - Unit that gets out of the vehicle.

3.4.5.16 LandedStopped

Usually, triggered when an AI pilot would get out. Not executed for players.

Type:

Parsed array, Local EH

Returned Values:

[plane, airportID]

- plane: Object - Object event handler is assigned to.
- airportID: Number - ID of the airport (-1 for anything else).

3.4.5.17 LandedTouchDown

Triggered when a plane (AI or player) touches the ground.

> **FEATURE NOTICE**
>
> Temporarily disabled for AI. See One AI in the VBS4 Release Notes.

Type:

Parsed array, Local EH

Returned Values:

[plane, airportID]

- plane: Object - Object event handler is assigned to.
- airportID: Number - ID of the airport (-1 for anything else).

3.4.5.18 LaserFired

Triggered when a vehicle laser range finder is invoked.

The returned values are relative to the vehicle.

If the event-handler addEventHandler (https://sqf.bisimulations.com/display/SQF/addEventHandler) command returns true (for example, `tank addEventHandler["LaserFired","hint 'fired'; true"]`), lasing is interrupted (and the distance displayed appears in red).

If a range finder is returning an illegal range (for example, too far or too close, as indicated by an "`ERR`" message), the event handler returns the last lased distance, in addition to the new directions.

Depending on the vehicle or weapon configuration, multiple returns are provided.

Type:

Parsed array, Default EH

Returned Values:

[distance, elevation, turn, vehicle, turret, position, direction, distances]

- distance: Number - Current distance (or previous one, in case of error).
- elevation: Number - Required weapon elevation (in degrees) to hit the specified location.
- turn: Number - Required weapon turning angle (in degrees) to hit the specified location.
- vehicle: Object - Object that fired the laser range finder.
- turret: Array - Path to turret that fired the laser range finder.
- position: Vector3D - Starting position of the beam.
- direction: Vector3D - Laser beam direction.
- distances: Array - All ranges (Numbers) collected in event.

3.4.5.19 LoadOutChanged

Triggered whenever a weapon or magazine is added / removed from a unit or vehicle.

This could occur as a result of script commands (such as addWeapon (https://sqf.bisimulations.com/display/SQF/addWeapon) or removeMagazine (https://sqf.bisimulations.com/display/SQF/removeMagazine)), or it could happen as a result of in-game actions such as reloading magazines, dropping weapons, for example.

Type:

Parsed array, Default EH

Returned Values:

[object]

- object: Object - Object that has its equipment changed.

3.4.5.20 PathPlanFailed

Triggered when a unit / vehicle is not able to reach a assigned destination (assignment can be done using a waypoint, move commands, or group order).

> **FEATURE NOTICE**
> Temporarily disabled for AI. See One AI in the VBS4 Release Notes.

Type:

Parsed array, Default EH

Returned Values:

[unit, destination, retry]

- unit: Object - Unit / vehicle the event handler is assigned to.
- destination: Position3D - Position that cannot be reached.
- retry: Boolean - If true, the unit retries reaching the destination (normally, it retries twice before giving up, at which point false is returned).

3.4.5.21 PositionChanged

Added to a vehicle. Triggered when a person either enters, exits, or moves position inside the vehicle.

Type:

Parsed array, Default EH

Returned Values:

[vehicle, person, positionFromArray, positionToArray]

- vehicle: Object - Vehicle that the event handler is attached to.
- person: Object - Unit that entered /exited / moved in the vehicle.

- positionFromArray / positionToArray: Array - Indicates the position the unit was in, and the position they moved to.

  Format of both arrays is: **[type, turretPath or cargoIndex]**

  - type: Number - 0 = not in vehicle; 1 = driver; 2 = cargo position, 3 = turret.
  - turretPath: Array - Path to turret.
  - cargoIndex: Number - Cargo slot index.

3.4.5.22 TurnIn

Triggered when a unit 'turns in' in a vehicle turret position.

Type:

Parsed array, Default EH

Returned Values:

[vehicle, unit, turretpath]

- vehicle: Object - Vehicle the unit is in.
- unit: Object - Unit that is turning in.
- turretpath: Array - [] for driver.

3.4.5.23 TurnOut

Triggered when a unit 'turns out' in a vehicle turret position.

Type:

Parsed array, Default EH

Returned Values:

[vehicle, unit, turretpath]

- vehicle: Object - Vehicle the unit is in.
- unit: Object - Unit that is turning out.
- turretpath: Array - [] for driver.

3.4.5.24 URNonChanged

Triggered through VBS Gateway, when a vehicle URN is changed.

Type:

Parsed array, Global EH

Returned Values:

[identifier]

- identifier: String - LVC identifier.

3.4.5.25 VehicleCreated

Triggered when a vehicle is created using either the Editor, SQF, or VBS Simulation SDK.

Type:

Parsed array, Global EH

Returned Values:

[vehicle, isSlowVehicle]

- vehicle: Object - Created vehicle object.
- isSlowVehicle: Boolean - Indicates whether the object is not expected to move often (for example, in case of a static object).

3.4.6 Weapons / Shots

The following weapons / shots event handlers are available:

- AmmoCreate (below)
- AmmoDelete (on the next page)
- AmmoExplode (on the next page)
- AmmoHit (on page 104)
- ChangedWeapon (on page 105)
- Explosions (on page 105)
- Fired (on page 105)
- FiredNear (on page 106)
- IncomingFire (on page 107)
- IncomingLaser (on page 107)
- IncomingMissile (on page 108)
- IncomingProjectile (on page 108)
- LaserFired (on page 109)
- SelectWeapon (on page 110)

3.4.6.1 AmmoCreate

Called whenever any `CfgAmmo` object of is created.

Has the following properties:

- Called on all clients and servers
- Configuration based only
- Not recorded by the AAR

Type:

Parsed array, Default EH

Returned Values:

[bullet, shooter, position, velocity, selection, ammo, direction, exploded]

- bullet: Object - Object that is fired.
- shooter: Object - Unit that fires the shot.
- target: Object - Always NULL.
- position: Position3D - Position of the fired object.

- velocity: Vector - 3D speed, speed of fired object.
- selection: Array - Always empty.
- ammo: Array - Ammunition info:

  [hit value, indirect hit value, indirect hit range, explosive damage, ammo class name]

  - direction: Vector3D - Always [0,0,0].
  - exploded: Boolean - Always 'false'.

3.4.6.2 AmmoDelete

Triggered when a shot is deleted using the deleteVehicle (https://sqf.bisimulations.com/display/SQF/deleteVehicle) script command.

Type:

Parsed array, Default EH

Returned Values:

[bullet, shooter, target, position, velocity, selection, ammo, direction, exploded]

- bullet: Object - Object (bullet, shell) that is fired.
- shooter: Object - Unit that fires the shot.
- target: Object - Object that is hit (always objNull).
- position: Position3D - Position of the bullet when it is deleted.
- velocity: Vector3D - 3D speed of the bullet when it is deleted.
- selection: Array - Array of named selections that are hit (always empty: []).
- ammo: Array - Ammunition info:

  [hit value, indirect hit value, indirect hit range, explosive damage, ammo class name]

  - direction: Vector3D - Vector to surface struck (always [0,0,0]).
  - exploded: Boolean - Whether ammunition exploded, or was destroyed (always 'true').

3.4.6.3 AmmoExplode

Only triggered when ammunition explodes or terminally impacts something.

When creating an addon, this event can be configured in class `CfgAmmo`.

Type:

Parsed array, Default EH

Returned Values:

[bullet, shooter, position, velocity, selection, ammo, direction, exploded]

- bullet: Object - Object that is fired.
- shooter: Object - Unit that fires the shot.
- target: Object - Object that is hit, or objNull, if the ground is hit.
- position: Position3D - Position the bullet impacted in model coordinates (if an object was hit), or position ASL if the ground is hit.
- velocity: Vector - 3D impact speed of the bullet.
- selection: Array - Array of Strings with the named selections of the object that are hit.
- ammo: Array - Ammunition info:

  [hit value, indirect hit value, indirect hit range, explosive damage, ammo class name]

  - direction: Vector3D - Vector that is orthogonal (perpendicular) to the surface struck. For example, if a wall is hit, the vector points out of the wall at a 90 degree angle.
  - exploded: Boolean - True if the ammunition explodes / is destroyed, false if it ricochets.

3.4.6.4 AmmoHit

Triggered when ammunition hits something, even if it ricochets.

When creating an addon, this event can be configured in class `CfgAmmo`.

Type:

Parsed array, Default EH

Returned Values:

[bullet, shooter, position, velocity, selection, ammo, direction, exploded]

- bullet: Object - Object that is fired.
- shooter: Object - Unit that fires the shot.
- target: Object - Object that is hit, or objNull, if the ground is hit.
- position: Position3D - Position the bullet impacts in model coordinates (if an object is hit), or position ASL if the ground is hit.
- velocity: Vector - 3D impact speed of the bullet.
- selection: Array - Array of Strings with the named selections of the object that were hit.
- ammo: Array - Ammunition info:

  [hit value, indirect hit value, indirect hit range, explosive damage, ammo class name]

  - direction: Vector3D - Vector that is orthogonal (perpendicular) to the surface struck. For example, if a wall is hit, the vector points out of the wall at a 90 degree angle.
  - exploded: Boolean - True if the ammunition explodes / is destroyed, false if it ricochets.

3.4.6.5 ChangedWeapon

Added to the configuration of a weapon (not added using the addEventHandler (https://sqf.bisimulations.com/display/SQF/addEventHandler) script command).

Triggered whenever this weapon is added or removed from a unit.

Type:

Parsed array, Default EH

Returned Values:

[owner, weapon, added]

- owner: Object - Object that has the weapon added or removed.
- weapon: String - Class name of the weapon that is added or removed.
- added: Boolean - True if the weapon is added to the owner, false if it is removed.

3.4.6.6 Explosions

Triggered when an explosion happens in the mission (only works with player units).

Type:

Parsed array, Default EH

Returned Values:

[positionX, positionY, positionZ, indirectHit, indirectHitRange, energyFactor]

- positionX/Y/Z: Number - Impact coordinates (above sea level).
- indirectHit: String - Ammunition configuration value.
- indirectHitRange: String - Ammunition configuration value.
- energyFactor: String - Unknown.

3.4.6.7 Fired

Triggered when a unit fires a weapon.

This event handler is triggered if a unit fires from a vehicle. For such cases, an event handler has to be attached to that particular vehicle.

Type:

Parsed array, Global EH

Returned Values:

[unit, weapon, muzzle, mode, ammo, magazine, projectile]

- unit: Object - Object event handler is assigned to.
- weapon: String - Fired weapon.
- muzzle: String - Muzzle which is used.
- mode: String - Current mode of the fired weapon.
- ammo: String - Ammunition used.
- magazine: String - Magazine which is used.
- projectile: Object - Object of the projectile that is shot.

3.4.6.8 FiredNear

Triggered when a weapon is fired nearby. Does not respond to explosions (IEDs or artillery) or thrown projectiles.

Effective range depends on the environment. In an open area, this event handler may fire up to a distance of 80m, in an urban area it may be less than 40m. The caliber and ammo type do not affect the trigger range (for example, a silenced 9mm pistol may have the same range as a 50cal machine gun).

This event handler continues to fire, even after the unit it is attached to is dead.

Type:

Parsed array, Default EH

Returned Values:

[unit, shooter, distance, weapon, muzzle, mode, ammo]

- unit: Object - Object the event handler is assigned to.
- shooter: Object - Unit / vehicle that fired the weapon.
- distance: Number - Distance to shooter.
- weapon: String - Fired weapon.
- muzzle: String - Muzzle which is used.
- mode: String - Current mode of the fired weapon.
- ammo: String - Ammunition used.

3.4.6.9 IncomingFire

Triggered when a bullet is fired at the associated object.

> **NOTE**
>
> The Event Handler is triggered when a bullet (or other projectile) gets into the 100m range from a unit.

Type:

Parsed array, Default EH

Returned Values:

[unit, shooter, ammo, distance, position, projectile]

- unit: Object - Object the event handler is assigned to.
- shooter: Object - Object that fired the weapon.
- ammo: String - Ammunition type that is fired on the unit.
- distance: Number - Nearest distance the bullet travels to hit the unit.
- position: Number - Nearest position the bullet travels to hit the unit.
- projectile: Object - Object of the projectile that is shot.

3.4.6.10 IncomingLaser

Triggers whenever an entity that has this event handler set is lased by a laser designator, LRF, or laser pointer. It also triggers when the lasing stops. The event handler can trigger multiple times, if it the lasing continues.

The following parameters are available:

- **CONTINUOUS** - If set to true, continuous lasing is repeatedly reported (potentially, each frame). If set to false, continuous lasing is reported only twice - at lasing start and lasing end.
- **PERIOD** - A single continuous lasing event is reported with a frequency set up by the period (in seconds). Can be used to reduce the frequency of calling a script.
- **TRIGGER_ANGLE_DELTA** - Continuous lasing, with CONTINUOUS set to false, is reported if the angle of the incoming laser moves by at least the delta value (in degrees), since the event was reported last time.

Type:

Parsed array, Local EH

Returned Values:

[lasedEntity, laserSource, laserType, laseEventID, endOfLaseEvent, laserDir, laserPos, laserConeAngle]

- lasedEntity: Object - Lased entity (that has the event handler set).
- laserSource: Object - Lasing entity.
- laserType: String - Laser type. Can be: "rangefinder", "designator", "pointer".
- laseEventID: Number - Lase event ID, to track continuous lasing and the end of the lasing event.
- endOfLaseEvent: Boolean - Set to true, if the event notifies of the lasing event end (a vehicle stops being lased by that specific laser).
- laserDir: Vector - Direction of the laser beam, from source to target.
- laserPos: Vector - Position of the laser origin, in world coordinates.
- laserConeAngle: Number - Angle of the laser cone for the conical laser beam.

3.4.6.11 IncomingMissile

Triggered when a guided missile is locked-on to a target or an unguided missile or rocket aimed by an AI at a target is fired.

Type:

Parsed array, Global EH

Returned Values:

[unit, ammo, whoFired]

- unit: Object - Object the event handler is assigned to.
- ammo: String - Ammunition type that is fired at the unit.
- whoFired: Object - Object that fires the weapon.

3.4.6.12 IncomingProjectile

Triggered when a projectile reaches a certain distance from the attached vehicle. The trigger distance can be set using the `parameters` parameter of addEventHandler (https://sqf.bisimulations.com/display/SQF/addEventHandler), and passing `TRIGGER_DISTANCE`. If not specified, the default trigger distance is 10m.

Type:

Parsed array, Global EH

Returned Values:

[vehicle, projectile, ammoType, distance, projectilePos, projectileVel, shooter, canDamage, shotOriginPos]

- vehicle: Object - Object the event handler is assigned to.
- projectile: Object - The object of the detected projectile.
- ammoType: String - The CfgAmmo class name of the detected projectile.
- distance: Number - The estimated distance that is reached by the projectile, after the next simulation step (not the current distance of the projectile from the target vehicle).
- projectilePos: Position - The future position of the projectile, at the moment it passes the border of the detection range. The position is always on the sphere, which is defined by the vehicle position and the detection radius at the intersection of the sphere surface with the trajectory of the projectile (with the exception of the projectile being fired from inside of the detection area).
- projectileVel: Array - The velocity of the projectile, when it is detected.
- shooter: Object - The unit or vehicle that fired the shot, or objNull if nobody fired it.
- canDamage: Boolean - Set to true, if the projectile can cause damage.
- shotOriginPos: Position - World position of the location, where the shot is spawned.

3.4.6.13 LaserFired

Triggered when a vehicle laser range finder is invoked. Returned values are relative to the vehicle.

If the event handler command returns true, lasing is interrupted (and distance displayed shows red).

If a range finder is returning an illegal range (too far or too close, as indicated by an "ERR" message), **LaserFired** returns the last lased distance, in addition to the new directions.

Type:

Parsed array, Default EH

Returned Values:

[lasedDistance, wantedAzimuth, wantedElevation, vehicle, turretPath, laserOrigin, laserDirection, lasedDistances]

- lasedDistance: Number - Main lased distance used for FCS.
- wantedAzimuth: Number - Azimuth that the turret needs to move into.
- wantedElevation: Number - Elevation that the turret needs to move into.
- vehicle: Object - Vehicle that lases.
- turretPath: Array of Number - The lasing turret path.

- laserOrigin: PositionASL2 - Position of the laser origin.
- laserDirection: Position3D - Direction of the lasing optics.
- lasedDistances: Array of Number - Echoes returned by the LRF in ascending order, only one value if LRF simulation is disabled or if the vehicle is not configured for it.

3.4.6.14 SelectWeapon

Triggered whenever a weapon is selected, or the muzzle changes, or is deselected.

Type:

Parsed array, Default EH

Returned Values:

[unit, weapon, muzzle, mode, selected]

- unit: Object - Object that is selected or deselected for the weapon (vehicle or soldier).
- weapon: String - Class name of the weapon that is selected or deselected.
- muzzle: String - Muzzle that is selected or deselected.
- mode: String - Mode of the weapon that is selected or deselected.
- selected: Boolean - True if weapon is selected or if mode or muzzle changes while weapon remains the same, false if the weapon is deselected.

3.4.7 Multiplayer

The following multiplayer event handlers are available:

- MPHit (below)
- MPKilled (on the next page)
- MPRespawn (on the next page)

3.4.7.1 MPHit

Triggered when a unit is hit / damaged. Same behavior as the regular Hit (on page 73) event handler, but the effect is global.

Type:

Parsed array, Default EH

Returned Values:

[target, shooter, damage]

- target: Object - Object that was injured / damaged.
- shooter: Object - Unit that inflicted the damage. If injured by a vehicle impact or a fall, the target itself is returned. For IEDs or artillery, the null object is returned. For explosives planted by someone (for example, satchel charges), that unit is returned.
- damage: Number - Level of damage caused by the hit.

3.4.7.2 MPKilled

Triggered when a unit is killed. Same behavior as the regular Killed (on page 74) event handler, but the effect is global.

Type:

Parsed array, Default EH

Returned Values:

[unit, killer]

- target: Object - Object that is killed.
- shooter: Object - The killer unit. If injured by a vehicle impact or fall, the target itself is returned. For IEDs or artillery, either the target itself is returned, or the null object, if further away from the explosion. For explosives planted by someone (for example, satchel charges), that unit is returned.

3.4.7.3 MPRespawn

Triggered when an object respawns. Same behavior as the regular Respawn (on page 88) event handler, but the effect is global.

Type:

Parsed array, Default EH

Returned Values:

[new unit, old unit]

- new unit: Object - The player new unit, after respawning.
- old unit: Object - The player old unit, before respawning.

3.4.8 Camera

The following camera event handlers are available:

- CameraOnChanged (below)

3.4.8.1 CameraOnChanged

Triggered when the player camera view changes, by either:

- Running switchCamera (https://sqf.bisimulations.com/display/SQF/switchCamera).
- Running selectPlayer (https://sqf.bisimulations.com/display/SQF/selectPlayer).
- Selecting **Switch to Unit** in the Editor context menu during scenario runtime.
- Selecting **Camera Views > Player Camera** in the Editor context menu during scenario runtime.
- Selecting **Camera Views > Bullet Camera** in the Editor context menu during scenario runtime.
- Exiting a vehicle.

Type:

Parsed array, Global EH

Returned Values:

[object]

- object: Object - The object that the camera is attached to.

3.4.9 Communication

The following communication event handlers are available:

- OnConversationStart (below)
- OnConversationEnd (on the next page)
- OnConversationSay (on the next page)

3.4.9.1 OnConversationStart

Triggered when the VBS conversation system is started (see conversationStart (https://sqf.bisimulations.com/display/SQF/conversationStart)).

Type:

Parsed array, Global EH

Returned Values:

[player, unit, conversationClass, initNode]

- player: Object - Player that initiates the conversation.
- unit: Object - Conversation partner (AI or another player).
- conversationClass: String - Conversation class being used.
- initNode: String - Initial conversation sub-class being used (can be empty for player - player conversations). If none is specified in conversationStart (https://sqf.bisimulations.com/display/SQF/conversationStart), then the name of the `initState` class is returned.

3.4.9.2 OnConversationEnd

Triggered when a VBS conversation system is ended (see conversationEnd (https://sqf.bisimulations.com/display/SQF/conversationEnd)), either using the conversation menu, conversationEnd (https://sqf.bisimulations.com/display/SQF/conversationEnd), or by pressing **Esc**.

Type:

Parsed array, Global EH

Returned Values:

[player, unit, endNode]

- player: Object - Player that initiates the conversation.
- unit: Object - Conversation partner (AI or another player).
- endNode: String - Last conversation sub-class used.

3.4.9.3 OnConversationSay

Triggered when a phrase is spoken using the VBS conversation system.

Type:

Parsed array, Global EH

Returned Values:

[player, unit, conversationClass, nodeClass, nodeText]

- player: Object - Unit that is speaking.
- unit: Object - Unit that is being spoken to.
- conversationClass: String - Conversation class used.
- nodeClass: String - Conversation sub-class used.
- nodeText: String - Full text (with substituted variables) used.

3.4.10 System

The following system event handlers are available:

> **NOTE**
>
> You can configure system event handlers using classes. For more information, see Class CfgSystemEventHandlers in Event Handlers Configuration in the VBS Developer Reference.

- editorLoad / editorUnload (below)
- EDITOR_CAMERA_MODE (below)
- inventoryLoad / inventoryUnload (on the next page)
- inventoryItemMoved (on the next page)
- beforeInventoryItemMoved (on page 116)
- VBS2_IED_PLACE_PICKUP_SYSEH (on page 117)

3.4.10.1 editorLoad / editorUnload

Triggered when the Editor opens / closes (loads / unloads).

Type:

Parsed array, System EH

Returned Values:

[editorMode, map]

- editorMode: String - Either "RTE", "OME", "C2", or "AAR".
- map: Control - Editor map control (used in many Editor scripting commands).

3.4.10.2 EDITOR_CAMERA_MODE

Triggered when switching between default, first-person, bullet, and nose cameras in the Editor.

Type:

Parsed array, System EH

Returned Values:

[cameraName]

- cameraName: String - Either "DEFAULT", "FIRSTPERSON", "GUN", or "NOSE".

3.4.10.3 inventoryLoad / inventoryUnload

Triggered when the inventory opens / closes (loads / unloads).

Type:

Parsed array, System EH

Returned Values:

[commonData]

- commonData: Array - An array that contains the following:
  - inventoryMode: String - Method in which the inventory dialog is used:
    - "PLAYER" - Opened by the player, to modify their own inventory or that of teammates.
    - "TEMPLATE" - Opened in the Editor, to create or modify a unit loadout template.
    - "EDITOR" - Opened in the Editor, to modify the inventory of an existing unit.
  - owner: Object - Unit whose inventory is being modified.
  - display: Display - Inventory dialog display.

3.4.10.4 inventoryItemMoved

Triggered when an item is moved using the inventory interface. Only runs on the computer that initiates the inventory move.

> **NOTE**
>
> Does not run when weapons or magazines are moved using other means, such as the AI picking them up, or picking them up using 3D World Actions (see 3D World Actions in the VBS4 Trainee Manual). Those can be reliably detected using the LoadOutChanged (on page 85) event handler.

Type:

Parsed array, System EH

Returned Values:

[commonData, oldOwner, newOwner, cfgType, itemID, stackSize]

- commonData: Array - An array that contains the following:
  - inventoryMode: String - Method in which the inventory dialog is used:
    - "PLAYER" - Opened by the player, to modify their own inventory or that of teammates.
    - "TEMPLATE" - Opened in the Editor, to create or modify a unit loadout template.
    - "EDITOR" - Opened in the Editor, to modify the inventory of an existing unit.
  - owner: Object - Unit whose inventory is being modified.
  - display: Display - Inventory dialog display.
- oldOwner: Object - Object that used to hold the item. For objects on the ground, objNull is used. For weapons / magazines on the ground, a weapon holder is used.
- newOwner: Object - Object that now has the item. For objects on the ground, objNull is used.
- cfgType: String - Category of item being moved. Can be: "CfgVehicles" (object), "CfgWeapons" (weapon), or "CfgMagazines" (magazine).
- itemID: String or Object - Item being moved. For weapons / magazines, the class name is provided (String). For objects, the actual object being moved is provided (Object).
- stackSize: Number - Number of items being moved (for weapons / magazines).

3.4.10.5 beforeInventoryItemMoved

Triggered right before an item is moved using the inventory interface. Only runs on the computer that initiates the inventory move.

> **NOTE**
>
> Does not run when weapons or magazines are moved using other means, such as the AI picking them up, or picking them up using 3D World Actions (see 3D World Actions in the VBS4 Trainee Manual). Those can be reliably detected using the LoadOutChanged (on page 85) event handler.

Type:

Parsed array, System EH

Returned Values:

[commonData, oldOwner, newOwner, cfgType, itemID, stackSize]

- commonData: Array - An array that contains the following:
  - inventoryMode: String - Method in which the inventory dialog is used:
    - "PLAYER" - Opened by the player, to modify their own inventory or that of teammates.
    - "TEMPLATE" - Opened in the Editor, to create or modify a unit loadout template.
    - "EDITOR" - Opened in the Editor, to modify the inventory of an existing unit.
  - owner: Object - Unit whose inventory is being modified.
  - display: Display - Inventory dialog display.
- oldOwner: Object - Object that used to hold the item. For objects on the ground, objNull is used. For weapons / magazines on the ground, a weapon holder is used.
- newOwner: Object - Object that now has the item. For objects on the ground, objNull is used.
- cfgType: String - Category of item being moved. Can be: "CfgVehicles" (object), "CfgWeapons" (weapon), or "CfgMagazines" (magazine).
- itemID: String or Object - Item being moved. For weapons / magazines, the class name is provided (String). For objects, the actual object being moved is provided (Object).
- stackSize: Number - Number of items being moved (for weapons / magazines).

Setting the third parameter of fn_vbs_addSysEventHandler (https://sqf.bisimulations.com/display/SQF/fn_vbs_addSysEventHandler) to true, when adding the event handler, allows to return a value from it - returning true lets the inventory move proceed, returning false or [false, "error message"] blocks the move, displaying the optional message in the inventory window (a generic error message is displayed otherwise). The first event handler that cancels the move is the one that shows the error message.

3.4.10.6 VBS2_IED_PLACE_PICKUP_SYSEH

Triggered when an IED is placed or picked up.

Type:

Parsed array, System EH

Returned Values:

[pickedUp, ied, unit]

- pickedUp: Boolean - Indicates if the IED is being picked up or not.
- ied: Object - The IED.
- unit: Object - Unit placing or picking up the IED.

3.4.11 Other

The following other event handlers are available:

3.4.11.1 Attached

Triggered when a child object is attached to a parent object.

When a child object is detached, the parent object returns objNull. When a child object is attached to another object, the parent returns the other object.

> **NOTE**
> Works in the same way as AttachTo (on the next page).

Type:

Parsed array, Global EH

Returned Values:

[object, parent, child, position]

- object: Object - Object that the event handler is attached to.
- parent: Object - Parent object.

- child: Object - Child object.
- position: Position3D - Position the object is attached to, in model coordinates.

3.4.11.2 AttachTo

Triggered whenever an object is attached to another object using the attachTo (https://sqf.bisimulations.com/display/SQF/attachTo) script command, or when an object is detached from another object.

Type:

Parsed array, Global EH

Returned Values:

[attached, stuckto, position]

- attached: Object - Object that is attached to another object (or objNull, if an object is detached).
- stuckto: Object - Object that has an object attached to it.
- position: Position3D - Position the object is attached to, in model coordinates.

3.4.11.3 ComponentAttached

Triggered for the base object when a Universal Component System (UCS) component is attached to it directly (parent object is the base object), or to a component attached to the base object (parent object is not the base object). Also triggered when the moveComponentToSlot (https://sqf.bisimulations.com/display/SQF/moveComponentToSlot) command executes.

Type:

Parsed array, Default EH

Returned Values:

[base, parent, component, slot, position]

- base: Object - Unit / vehicle that is the base object.
- parent: Object - Parent object.
- component: String - Component class name.
- slot: String - Slot to add the new component to.
- position: Number - Rail position index.

3.4.11.4 ComponentDetached

Triggered on the base object when a Universal Component System (UCS) component is detached from it directly (parent object is the base object) or from a component attached to the base object (parent object is not the base object). Also triggered when the moveComponentToSlot (https://sqf.bisimulations.com/display/SQF/moveComponentToSlot) command executes.

Type:

Parsed array, Default EH

Returned Values:

[base, parent, component, slot, position]

- base: Object - Unit / vehicle that is the base object.
- parent: Object - Parent object.
- component: String - Component class name.
- slot: String - Slot to remove the component from.
- position: Number - Rail position index.

3.4.11.5 FileChanged

Triggered when a file that matches the fileWatch (https://sqf.bisimulations.com/display/SQF/fileWatch) filter is modified.

Type:

Parsed array, Global EH

Returned Values:

[filename]

- filename: String - Full path to the modified file.

3.4.11.6 Init

Triggered on mission start or when a vehicle is created 'on the fly' using createVehicle (https://sqf.bisimulations.com/display/SQF/createVehicle) script command.

Type:

Parsed array, Global EH

Returned Values:

[unit]

- unit: Object - Object event handler is assigned to.

3.4.11.7 MissionEnd

Triggered on mission end.

> **NOTE**
>
> The missionNameSpace (https://sqf.bisimulations.com/display/SQF/missionNameSpace) scope still exists when the event is triggered.

Type:

Parsed array, Global EH

Returned Values:

[restart]

- restart: Boolean - Returns true, if the mission is about to be restarted.

3.4.11.8 inputAction

Triggered when any user input is detected.

Type:

Parsed array, Global EH

Returned Values:

[action, inputAmount]

- action: String - Action activated / deactivated.
- inputAmount: Number or Boolean - Amount of input. For analog input, a number between 0 - 1. For digital input, a Boolean value of true or false.

3.4.11.9 OnBTMessage

Global event handler to process messages from Behavior Trees, sent using the `SendMessage` Lua function (see `SendMessage` in the VBS Control Editor Manual) for the `Message.External` recipient (see the `recipient` parameter in `SendMessage`).

> **NOTE**
>
> The following considerations apply:
>
> - Messages sent to other recipients cannot be intercepted by this event handler.
> - This event handler can only be used with the VBS Control Editor.
> - It is also possible to use the VBS Simulation SDK `ControlAIListenerAPI::OnMessage` function (see the Simulation SDK Developer Reference).

Type:

Parsed array, Default EH

Returned Values:

[sender, subject, data]

- sender: Group or Unit - The source of the message.
- subject: String - The subject of the message.
- data: Array - The contents of the message (for the format of the message, see Using Variables in the VBS Control Editor Manual).

3.4.11.10 onMove

Triggered when an object is moved around in the inventory dialog (only in the dialog).

Type:

Parsed array, Default EH

Returned Values:

[newOwner, oldOwner, itemClass, notDropped, newEquipMode]

- newOwner: Object - The new owner of the moved object.
- oldOwner: Object - The old owner of the moved object.
- itemClass: String - `CfgVehicles` class name of the moved object.
- notDropped: Boolean - True if the object is being picked up, false otherwise.

- newEquipMode: Number - Equipment mode. Can be:
  - 0 - Ground
  - 1 - Inventory
  - 2 - Weapon slots
  - 3 - Drag
- itemID: Object - The object being moved.

3.4.11.11 onPickup

Triggered when:

- Picking up an object using the inventory or a user action.
- Adding an object in the Editor (Execute Mode).
- Moving an object from the drag slot to the inventory.

Type:

Parsed array, Default EH

Returned Values:

[object, unit]

- object: Object - The object being picked up.
- unit: Object - The unit picking up the object.

3.4.11.12 statement

Code that is executed when an action is activated. Some standard variables can be used in the evaluation. The variable `this` references the object to which an action is attached to. The variable `_this` refers to the caller.

Type:

Parsed array, Default EH

Returned Values:

[code]

- code: String - The code to execute.

3.4.11.13 VariableChanged

Triggered whenever a setVariable (https://sqf.bisimulations.com/display/SQF/setVariable) script command is run for the specified unit / object. It does not matter if the variable is actually changed, or if it is created or destroyed — whenever any object variable for the specified owner is set, the event is triggered.

Type:

Parsed array, Default EH

Returned Values:

[owner, variable, newval, oldval]

- owner: Object - Object the event handler is assigned to, and whose variable is changed.
- variable: String - Name of variable that is set.
- newval: Anything - New value of variable (can be null, if variable is destroyed).
- oldval: Anything - Old value of variable (can be null, if variable is just created).

3.4.11.14 VariableNameChanged

Added to a SIDE, not an object. An object variable name is assigned using the setVehicleVarName (https://sqf.bisimulations.com/display/SQF/setVehicleVarName) script command. This event fires even if the new name is the same as the old one.

Type:

Parsed array, Default EH

Returned Values:

[unit, newName, oldName]

- object: Object - Object whose variable name is changed.
- newName: String - New variable name.
- oldName: String - Old variable name.

3.5 UI Event Handlers

UI event handlers are used to monitor UI displays, dialogs, and controls used in VBS4. A custom code is executed, when a particular UI event is triggered.

This topic describes the following:

- Defining UI Event Handlers (below) - How to define your own UI event handlers.
- UI Event Reference (on page 128) - A list of UI events, for which event handlers can be created.

3.5.1 Defining UI Event Handlers

You can define a UI event handler in two ways:

- Using the class parameter definitions (in `Description.ext` or the addon `config.cpp`).
- Using scripting commands (ctrlAddEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlAddEventHandler)).

For UI event handlers to work for a specific UI element (display / dialog / control), the UI element must be enabled using ctrlEnable (https://sqf.bisimulations.com/display/SQF/ctrlEnable).

Regardless of how the UI event is defined, it can be removed using the following commands:

- ctrlRemoveEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlRemoveEventHandler)
- ctrlRemoveAllEventHandlers (https://sqf.bisimulations.com/display/SQF/ctrlRemoveAllEventHandlers)
- displayRemoveEventHandler (https://sqf.bisimulations.com/display/SQF/displayRemoveEventHandler)
- displayRemoveAllEventHandlers (https://sqf.bisimulations.com/display/SQF/displayRemoveAllEventHandlers).

For more information, see Custom User Interface (UI) (on page 190).

3.5.1.1 Parameters

UI event handlers (for example, a clicked control or a change in the mouse position) receive parameters, using the _this (on page 27) variable.

EXAMPLE

The UI event handler for `onMouseButtonDown` sends a chat message, using sideChat (https://sqf.bisimulations.com/display/SQF/sideChat), if the **RMB** is clicked over the control, as defined in the following command string:

```
onMouseButtonDown =
"if ((_this select 1)==2) then {player sidechat 'Right mouse button
clicked'}";
```

3.5.1.2 Command Strings

The command string that is passed to the event handler can contain one or more commands, separated by semicolons. Also, local variables can be used in command strings.

EXAMPLE

An event handler for `onMouseButtonDblClick` (**LMB** double-click) stores the passed text control in `_ctrl`, saves its original content in `tst_var`, and then clears the control using ctrlSetText (https://sqf.bisimulations.com/display/SQF/ctrlSetText):

```
onMouseButtonDblClick =
"_ctrl=_this select 0; tst_var=ctrlText _ctrl; _ctrl ctrlSetText ''";
```

3.5.1.3 Return Values

Once the commands in the command string are executed, the event handler returns a Boolean to the VBS4 engine, which represents the result of the last command in the string:

- If the return value is `true`, the event has completed and no further processing of the event is done.
- If the return value is `false`, then normal event processing continues.

The return value is important in two situations:

- Key-press event handlers (for `onKeyDown`). If a key-press event returns `true`, then this prevents the engine from interpreting the **Esc** key, and dialogs cannot be closed using a key-press event anymore. In order to prevent that from happening, force a `false` return value at the end (replace `doSomething` with your commands):

```
onKeyDown = "doSomething; false";
```

- Multiple (stacked) event handlers for the same event type. If multiple event handlers are defined for the same event type (using ctrlAddEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlAddEventHandler) or displayAddEventHandler (https://sqf.bisimulations.com/display/SQF/displayAddEventHandler)) and the command string returns `true`, then only the first event is executed. This can be circumvented by forcing a `false` return value at the end:

```
_ctrl ctrlAddEventHandler ["MouseButtonDown","systemChat 'EH #1'; false"];
_ctrl ctrlAddEventHandler ["MouseButtonDown","systemChat 'EH #2'; false"];
```

3.5.1.4 Class-Defined Event Handlers

Event handlers can be defined in display / dialog and control classes (in `description.ext` or the addon `config.cpp`). The event parameter value (string) is executed as a line of code.

EXAMPLE

An `onMouseDown` event handler:

```
onMouseDown = "hint str _this";
```

An `onMouseButtonDblClick` event handler in a dialog class definition:

```
class DlgInput {
    idd = 20000;
    movingEnable = true;
    class controls {
        class EDIT1 : RscEdit {
            idc = 20000;
            x = .2; y = .2; w = .2; h = .2;
            onMouseButtonDblClick = "(_this select 0) ctrlSetText ''";
        };
    };
};
```

For more information, see Configuring UI Elements in the VBS Developer Reference.

The Developer Reference is in the `\docs\` folder of the VBS Developer Suite installation.

3.5.1.5 Script-Defined Event Handlers

Event handlers can also be defined using the following scripting commands:

- ctrlSetEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlSetEventHandler)
- ctrlAddEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlAddEventHandler)
- displaySetEventHandler (https://sqf.bisimulations.com/display/SQF/displaySetEventHandler)
- displayAddEventHandler (https://sqf.bisimulations.com/display/SQF/displayAddEventHandler)

WARNING

When using UI event handlers in SQF, the `"on"` prefix must be omitted (for example, use `"ButtonDown"` instead of `"onButtonDown"`). The prefix is only needed, if the UI event handlers are defined in configuration classes (see Class-Defined Event Handlers (on the previous page)).

EXAMPLE

```
(findDisplay 46) displaySetEventHandler [
  "keyDown", "_this execVM 'eventScript.sqf'"
];
```

3.5.2 UI Event Reference

The following table contains a list of UI events and their return parameters, with applicable UI displays / dialogs and controls, which can be used in the event handler code:

NOTE

The applicable UI displays / dialogs and controls can be found in the UI Configuration section of the VBS Developer Reference in the `\docs\` folder of your VBS Developer Suite installation.

Event Name	Description	Return Parameters	Applicable Classes
onAdvTreeClick	Any item in a tree row is clicked (down and up). If the button is still held down after about 1 second, the event is still triggered.	Returns the control, row number (1-based), column number (0-based).	Tree

Event Name	Description	Return Parameters	Applicable Classes
onAdvTreeDblClick	Any item in a tree row is double-clicked.	Returns the control, row number (1-based), column number (0-based).	Tree
onAdvTreeMouseDown	The mouse button is currently pressed over any item in a tree row.	Returns the control, row number (1-based), column number (0-based).	Tree
onAdvTreeMouseUp	The mouse button is released over any item in a tree row.	Returns the control, row number (1-based), column number (0-based).	Tree
onAdvTreeToggle	A tree row collapsed or expanded (either using a user action or a scripting command).	Returns the control, row number (1-based), and whether the branch is currently expanded.	Tree
onButtonClick	The attached button action is performed.	Returns the control.	ActiveText, Button
onButtonDown	The **LMB** is pressed over the button area, or a key is pressed.	Returns the control.	ActiveText, Button
onCanDestroy	Queries the dialog, if it can be closed (used for validation of contained data).	Returns the control and exit code (0 - closed using closeDialog (https://sqf.bisimulations.com/display/SQF/closeDialog), 2 - closed using **Esc**).	Any display / dialog
onChar	Triggered when some readable character is recognized.	Returns the control and the character code.	ActiveText, Tree, Button, Combo, Display, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox
onCheckBoxesSelChanged	Check box selection changes.	Returns the control, the selected element index, and the current state.	Checkbox
onChildDestroyed	Child display is closed.	Returns the display (whose child display is closed) and exit code.	Display
onDestroy	Triggered when a control is destroyed.	Returns the control and exit code.	Any display / dialog

Event Name	Description	Return Parameters	Applicable Classes
onDraw	Triggered when the map is drawn (can occur more than once per second).	Returns the map control.	Map
onHTMLLink	Pressing and releasing an HTML link.	Returns the control and HREF.	HTML
onIMEChar	Triggered when an IME character is recognized (used in Korean and other eastern languages).	Returns the control and the character code.	Any control class
onIMEComposition	When partial IME character is recognized (used in Korean and other eastern languages).	Returns the control and the character code.	Any control class
onKeyDown	Pressing any key. Triggered before the `onKeyUp` event.	Returns the control, the Direct Input Keyboard (DIK) key codes and the states of Shift, Ctrl, and Alt. If this event is attached to a display, and its function does not return `false`, the **Esc** key becomes non-functional, and you are unable to close the dialog. This can be used to make the dialog closure dependent on a certain condition.	ActiveText, Tree, Button, Combo, Display, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox

EXAMPLE

```
onKeyDown = "if (somecondition ||
    (_this select 1)!=1)
    then {false}
    else {true}"
```

Event Name	Description	Return Parameters	Applicable Classes
onKeyUp	Releasing any key. Triggered after the `onKeyDown` event.	Returns the control, the DIK key codes, and the states of Shift, Ctrl, and Alt.	ActiveText, Tree, Button, Combo, Display, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox
onKillFocus	Input focus is no longer on the control, which no longer accepts keyboard input. A control only loses focus, if another one receives it - just leaving the control bounding box does not kill the focus.	Returns the control.	ActiveText, Tree, Button, Combo, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox
onLBDblClick	A list box row is double-clicked.	Returns the control and the selected element index.	Listbox
onLBDrag	Drag and drop operation started.	Returns the control and a nested array, containing [text, data, value], for all the selected elements.	Listbox
onLBDragging	Drag and drop operation in progress.	Returns the control currently hovered over, the current X and Y coordinates, the IDC of the source control, a nested array containing [text, data, value, index], for all the selected elements, and a -1 as the last value. In 2D list boxes, the element index is calculated by counting every cell in every line (for example, the first cell in line 3, each containing 2 cells, returns 4).	Listbox

Event Name	Description	Return Parameters	Applicable Classes
onLBDrop	Drag and drop operation finished.	Returns the control that the item is dropped onto, the current X and Y coordinates, the IDC of the source control, a nested array containing [text, data, value, index], for all the selected elements, and the index it is dropped onto (or -1, if dropped onto a control that is not a list box, or onto an empty space of a list box). In 2D list boxes, the element index is calculated by counting every cell in every line (for example, the first cell in line 3, each containing 2 cells, returns 4).	ActiveText, Button, Combo, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox
onLBSelChanged	Triggered when a selection in a list box or combo box has changed, the **LMB** has been released, and the new selection is fully made.	Returns the control and the selected element index (always the latest selection, even if multiple selections are enabled).	Combo, Listbox
onLoad	Display and all controls are created.	Returns the display.	Display
	NOTE findDisplay (https://sqf.bisimulations.com/display/SQF/findDisplay)is not able to find the newly created display, until after the `onLoad` event has finished processing. If you require access to the display in your `onLoad` event handler, use the handle passed in the `_this` parameter.		
onMenuSelected	An item in the context menu (used only in the new mission editor) is selected.	Returns the control and the command ID.	Menu
onMouseButtonClick	Pressing and releasing a mouse button.	Returns the control, the pressed button (0 - left, 1 - right, and so on), the X and Y coordinates, and the states of Shift, Ctrl, and Alt.	ActiveText, Tree, Button, Combo, Group, HTML, Listbox, Map, Slider, Edit, Toolbox

Event Name	Description	Return Parameters	Applicable Classes
onMouseButtonDblClick	Pressing and releasing a mouse button twice, within a very short time.	Returns the control, the pressed button (0 - left, 1 - right, and so on), the X and Y coordinates, and the states of Shift, Ctrl, and Alt.	ActiveText, Tree, Button, Combo, Group, HTML, Listbox, Map, Slider, Edit, Toolbox
onMouseButtonDown	Pressing a mouse button, followed by the `onMouseButtonUp` event.	Returns the control, the pressed button (0 - left, 1 - right, and so on), the X and Y coordinates, and the states of Shift, Ctrl, and Alt. For some displays (for example, 46 - editor) the value for X and Y is always [0.5, 0.5]. If a display covers a dialog, the display event handler is still triggered, even when the dialog is clicked.	ActiveText, Tree, Button, Combo, Display, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox
onMouseButtonUp	Releasing a mouse button, followed by the `onMouseButtonDown` event.	Returns the control, the pressed button (0 - left, 1 - right, and so on), the X and Y coordinates, and the states of Shift, Ctrl, and Alt. For some displays (for example, 46 - editor) the value for X and Y is always [0.5, 0.5]. If a display covers a dialog, the display event handler is still triggered, even when the dialog is clicked.	ActiveText, Tree, Button, Combo, Display, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox
onMouseEnter	The mouse pointer enters the control area.	Returns the control.	ActiveText, Button, Tree, Statebox, Toolbox

Event Name	Description	Return Parameters	Applicable Classes
onMouseExit	The mouse pointer exits the control area.	Returns the control.	ActiveText, Button, Tree, Statebox, Toolbox
onMouseHolding	Triggered continuously, while the mouse is not moving within a certain interval.	Returns the control, the X and Y coordinates, and a Boolean, indicating whether the mouse cursor is currently over a compatible control. For displays or non-compatible controls, the display, and X, Y mouse speed is returned.	ActiveText, Tree, Button, Combo, Display, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox
onMouseMoving	Triggered continuously, while moving the mouse within a certain interval.	Returns the control, the X and Y coordinates, and a Boolean, indicating whether the mouse cursor is currently over a compatible control. For displays or non-compatible controls, the display, and X, Y mouse speed is returned.	ActiveText, Tree, Button, Combo, Display, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox
onMouseZChanged	Triggered when the mouse wheel position changes.	Returns the control and the number of lines scrolled through (depends on the scroll speed and system settings). Values are positive for forward scrolls, and negative for backward scrolls.	ActiveText, Tree, Button, Combo, Display, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox

Event Name	Description	Return Parameters	Applicable Classes
onObjectMoved	Moving an object.	Returns the control and the offset on the X, Y, and Z axes.	Object
onPlanClick	The mouse button is pressed.	Returns the [control, x, y], with the coordinates being relative to the plan view.	Plan
onPlanDraw	Triggered each frame the plan view control is displayed.	Returns the control. Does not work, if assigned using ctrlSetEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlSetEventHandler).	Plan
onPlanOnUnitLeft	A unit leaves a section covered by the plan view.	Returns [vehicle, control, unit, section].	Plan
	NOTE Unlike the other events, this one does not return the control as the first element.		
onplanUnitEntered	A unit enters a section covered by the plan view.	Returns [vehicle, control, unit, section].	Plan
	NOTE Unlike the other events, this one does not return the control as the first element.		
onPlanUnitSelect	A unit, visible in the plan view, is clicked.	Returns [control, unit, selected]. The last parameter toggles between `true` and `false`.	Plan
onRearrange	The aspect ratio has changed.	Returns the display.	Display
onSetFocus	Input focus is on the control, to accept keyboard input.	Returns the control.	ActiveText, Tree, Button, Combo, Group, HTML, Listbox, Map, Statebox, Slider, Edit, Toolbox

Event Name	Description	Return Parameters	Applicable Classes
onSetVisible	The control visibility changes, using ctrlShow (https://sqf.bisimulations.com/display/SQF/ctrlShow).	Returns the control and the visibility status (1 - visible, 0 - hidden).	ActiveText, Tree, Button, Combo, Group, HTML, Listbox, Map, Static, Statebox, Slider, Edit, Toolbox
onSliderPosChanged	Changing the position of a slider.	Returns the control and the change.	Slider
onStateBoxToggle	A state box control is clicked.	Returns the control, current state, and previous state. If the command string returns `true`, then the state does not change.	Statebox
onTimer	Triggered after the time defined in the `timer` parameter of a control has passed, since the control has been opened.	Returns the control.	Any control class
onToolBoxSelChanged	Tool box selection changes.	Returns the control and the selected element index.	Toolbox
onTreeCollapsed	The tree folder structure has collapsed.	Returns the control.	Tree
onTreeDblClick	Double-clicking a tree.	Returns the control.	Tree
onTreeExpanded	The tree folder structure has expanded.	Returns the control.	Tree
onTreeLButtonDown	Pressing and releasing the **LMB** on a tree.	Returns the control.	Tree
onTreeMouseExit	The mouse cursor exits the tree control area.	Returns the control.	Tree
onTreeMouseHold	Triggered continuously, while the mouse is not moving within a certain interval.	Returns the control.	Tree
onTreeMouseMove	Triggered continuously, while the mouse is moving within a certain interval.	Returns the control.	Tree

Event Name	Description	Return Parameters	Applicable Classes
onTreeSelChanged	Changing the selection in a tree.	Returns the control.	Tree
onUnload	Display is closed, but no controls are destroyed yet.	Returns the display and exit code. The exit code is 2, if the dialog is closed by pressing **Esc**. Otherwise, it is the number that is passed using closeDialog (https://sqf.bisimulations.com/display/SQF/closeDialog).	Display
onWebViewEvent	Allows the handling of JavaScript events coming from the HTML UI, loaded by a `CT_WEBVIEW` control. Essentially, the event handler forwards calls to the `WebViewListenerAPI` method `JavaScriptEvent` (see `WebViewListenerAPI` in the VBS Simulation SDK Manual), trying to convert the incoming data from JSON to SQF arrays.	Returns the control and any data converted to an SQF data structure from JSON. If the JavaScript event data is not convertible to JSON, a string with raw data is returned. For SQF arrays to JSON conversion rules, see Using the CWebView UI Control in the VBS Developer Reference.	WebView Control

EXAMPLE

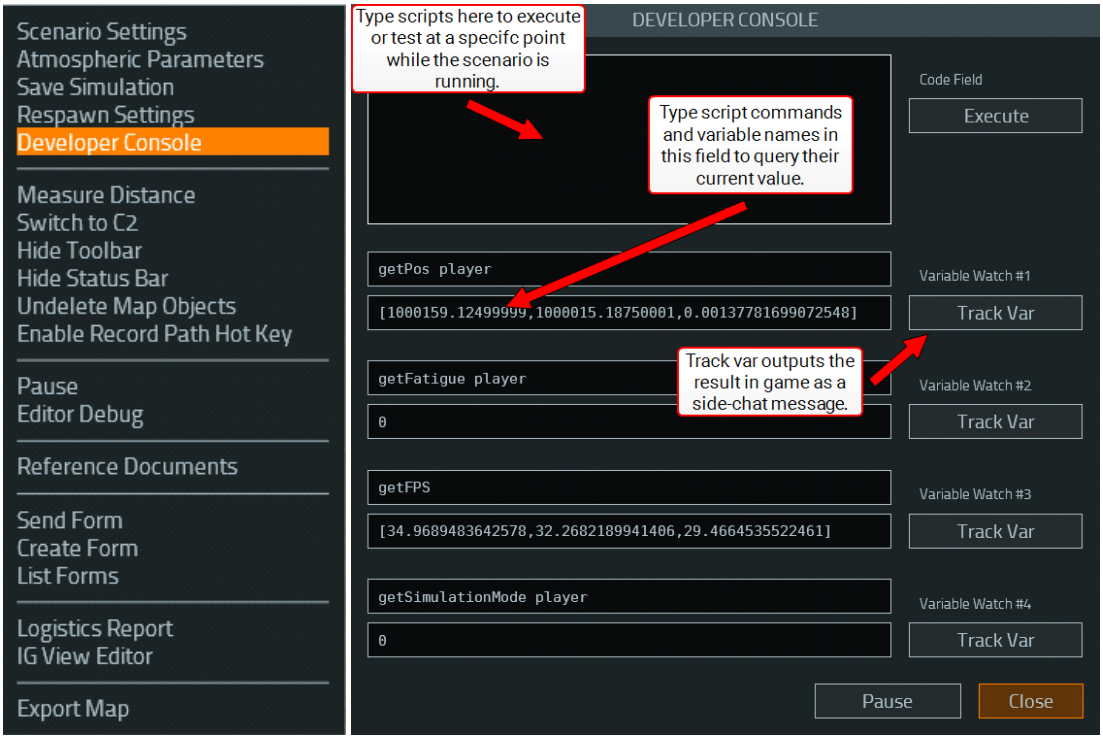
```
_ehId = _control ctrlAddEventHandler
    ["WebViewEvent",'hint format
    ["Control: %1, JS Event Data: %2",
    _this select 0,
    _this select 1]']
```

3.6 Developer Console

The Developer Console is available in the Editor (Prepare / Execute Mode) under the Tools menu. The console allows in-game execution of script code, and the reading and assigning of variables. It can be used by admins setting up a mission using scripting, or any script developer to test basic script statements that are used in the **Object Properties** dialog, as described in Using Basic Scripts (on page 15).

> **NOTE**
>
> When in the Editor (Execute Mode), executing some SQF code requires the Editor (Execute Mode) to be closed and reopened to take effect.

Image-11: The Developer Console



The code field stores a history of previous entries, use the up arrow and down arrow to switch.

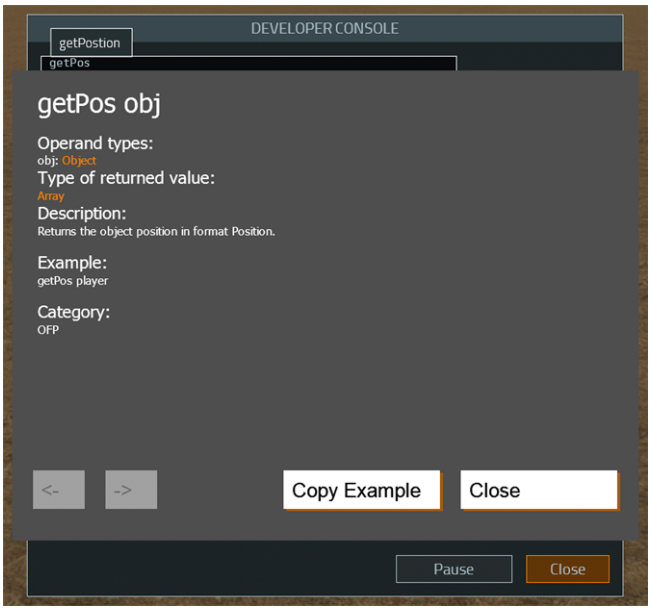
The full controls are:

Control	Description
Execute	Click **Execute** to run the code. **NOTE** In multiplayer, an additional option called **Execute Global** appears in the dialog, alongside **Execute**. This option allows you to execute your script commands on all clients and servers in a multiplayer session. Care should be taken when calling script commands, such as createVehicle (https://sqf.bisimulations.com/display/SQF/createVehicle), whose effects are already applied globally by default.
Track Var	During scenario execution, enter a variable in the code field and click the **Track Var** to track the value of the variable.
Show Variables / Write Variables	Click **Show Variables** to fix the contents of the variables in the boxes below the code fields (the variable values are not updated). Click **Write Variables** to update the variables to the values indicated in the boxes below the code fields. **NOTE** The controls are only available in the Editor (Prepare Mode).
Pause / Move On	During scenario execution, click **Pause** to pause the simulation and **Move On** to resume it.

3.6.1 Scripting Help - In-Game Help

Press **F1** to open the in-game command help dialog. This provides a description of the command with examples. It works in any editable field (for example, Initialization Statements in an Object (on page 16)).

Image-12: In-game command help



The in-game command help dialog has these sections:

Section	Description
Syntax	Specifies the command syntax.
Operand types	The data types of the operands / parameters used by the command.
Type of returned value	The data type of the return value.
Description	The general description of the command and what it does.
Example	An example of how to use the command.
Category	The category the command belongs to.

And these actions:

Action	Description
Backward (<-) / Forward (->) arrows	Switch to the previous / next command help, respectively.
Copy Example	Copies the example from the Example section.
Close	Closes the in-game command help dialog.

3.7 Debug Console Plugin

The Debug Console is a tool (and a plugin) for VBS4. It adds additional debugging functionality to the one found in the Developer Console (on page 138).

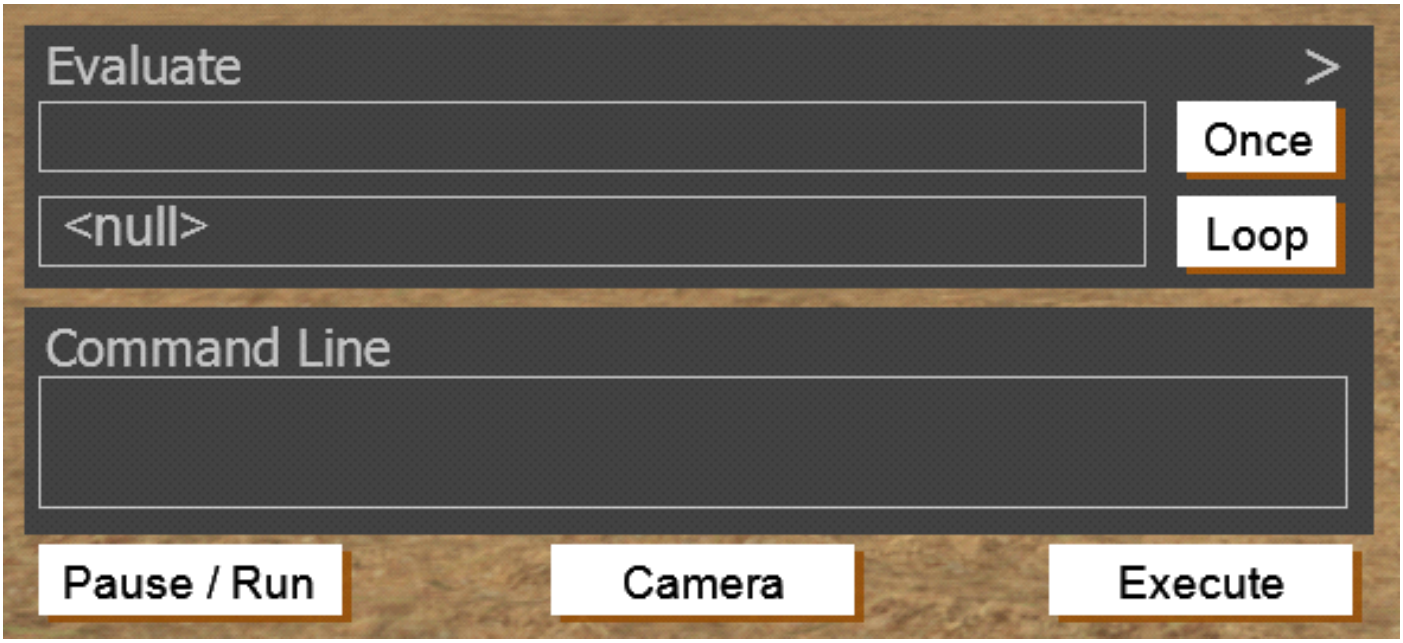
3.7.1 Using the Debug Console

To bring up the Debug Console, use the createDialog (https://sqf.bisimulations.com/display/SQF/createDialog) command to initiate the dialog resource "`Dlg_DebugConsole`" of the Debug Console:

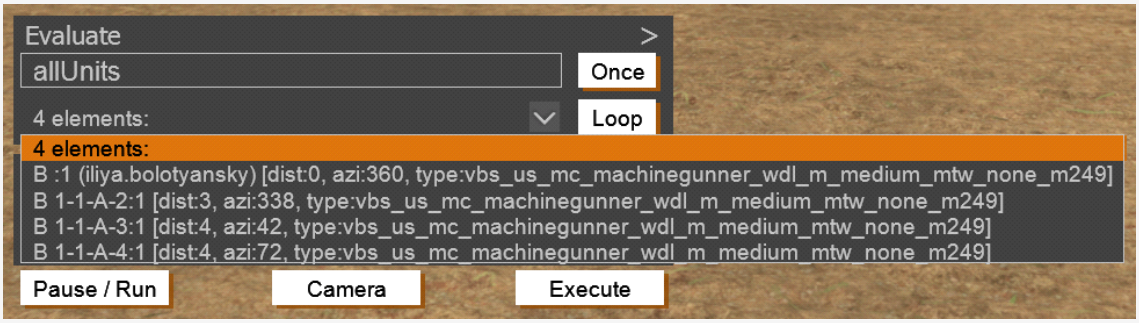
```
nul=createDialog "Dlg_DebugConsole"
```

> **NOTE**
> The Debug Console is only available in Preview / Execute mode.

Image-13: The Debug Console



3.7.2 Debug Console Actions

Action	Description
`Evaluate`	Evaluates an expression (for example, a global variable, an object, and so on) and outputs the result in the field below. • As the expression is being typed the output is automatically created. If the **Once** button is clicked (or **Enter** is pressed), the evaluation is saved, and can be recalled by pressing the **Arrow-Up** key. Saved entries can be removed from the history by selecting it, and pressing **Ctrl + X**. • If an evaluation returns an array with more than three numeric elements the content is displayed in a list box. Any element in the listed array that is an object also has its class name, direction and distance from the player shown in its entry line. If a list box item is selected, that element is then available (for further evaluation) using the global variable '`vbs_dc_obj`' (the whole listed array is available in '`vbs_dc_arr`'). • **Ctrl + C** copies the data directly to the Microsoft Windows clipboard, without an intermediate dialog, and can then be pasted into the wanted application by **Ctrl + V**. **Image-14: Evaluate example** 
`Once`	Displays the result of the Evaluate field and turns off a looped evaluation. • The result of the evaluations is stored in the '`vbs_dc_obj`' variable. The evaluation expression is saved, and can be recalled by pressing the **Arrow-Up** key. Pressing **Alt** while clicking the **Once** button, outputs the result to the DebugView console (developer version only).

Action	Description

`Loop`	Constantly (every 1/10 seconds) update the value of the expression in the Evaluate field.
	• Values are only displayed if they have changed (or if it has been more than 30 seconds since the last update).
	• To turn off a loop, either click the **Once** or **Loop** button.
	• If turned off using **Loop**, the history of the evaluation values is shown in a list box.
	• Some results are displayed at the top of the screen (for example, getPos (https://sqf.bisimulations.com/display/SQF/getPos)).
	• Pressing **Alt** while clicking the **Loop** button outputs to the DebugView console (developer version only).
	• The following options only have an effect if the dialog is closed, or the output is sent to the DebugView console:
	  ▪ Pressing **Shift** while clicking the **Loop** button puts a time stamp in front of the value.
	  ▪ Pressing **Ctrl** while clicking the **Loop** button outputs just the value, but not the evaluated expression in front of it.

`Command Line / Execute`	Executes the entered command. • The executed command is saved and can be recalled pressing the **Arrow-Up** key.
`Pause / Run`	Pauses or continues the simulation. • While paused, expressions or commands cannot be executed.
`Camera`	Starts the camera mode. • To leave the camera mode, press **V**. • To remove the crosshair, press **L**. • To toggle the cinematic borders, press **C**.

3.7.3 Debug Console Pseudo Commands

The Debug Console supports the use of pseudo commands, which can be entered in the Evaluate field. The following pseudo commands are currently defined:

Pseudo Command	Description
`@`	Lists all available pseudo commands.
`@autoammo`	Gives infinite ammo with automatic reloading.
`@coll`	Toggles collision detection for the player (and his vehicle).

Pseudo Command	Description
`@fix`	Give a unit (and his vehicle) full health, ammo, and fuel. Arguments: • None: (for example, `@fix`) Apply to player and his vehicle • object: (for example, `@fix tank`)
`@fps`	Toggles FPS display.
`@info`	Returns the position of the player position ASL, ASL, and grid format, as well as the current direction, map, and mission name.
`@jump (distance) (,height) (,"group")`	Teleports the player (or the group of the player). The moved unit(s) is invincible for the duration of the teleport. Arguments: • No arguments (for example, `@jump`) ◦ Teleports the player to the shift-clicked map position. ◦ If **Alt** is pressed, the whole player group is moved. • "aim" (for example, `@jump "aim"`) ◦ Teleports the player to the position he is aiming at. • distance (and height) (for example, `@jump 100,10`) ◦ The player is moved in the direction he is facing (by the distance specified). • positional array (for example, `@jump [1234, 5678]`) ◦ The player is placed at the entered destination. • grid (for example, `@jump "12345678"`) ◦ Player is placed at the entered grid destination (grid coordinates can be entered with or without a separating period (for example, "12345678" or "1234.5678"). • "group" (for example, `@jump "12345678","group"`) ◦ The whole player group is moved.
`@nearest (min) (,max)`	Shows the nearest objects in front of the player, excluding any environmental ones (for example, bugs, footsteps, and so on). Arguments: • min, max (for example, `@nearest 1,10`) default = 5,20 ◦ The minimum and maximum distance to check.

Pseudo Command	Description
`@parents object (,classTree)`	Returns the hierarchical list of inheritances and config definitions for the supplied object (or class name). Arguments: • object (for example, `@parents player, or @parents "vbs2_cone"`) ◦ The object or class name to be evaluated. • configFile (for example, `@parents primaryWeapon player,"cfgWeapons"`) ◦ The config file to use (default: "`cfgVehicles`"). • "detail" (for example, `@parents player,"detail"`) ◦ All the class definitions of the hierarchy tree are listed. • "sort" (for example, `@parents player,"detail","sort"`) ◦ The class definitions are listed alphabetically. • "all" (for example, `@parents player,"detail","all"`) ◦ Must be used together with the "detail" argument. Even class definitions for base classes like "AllVehicles" are listed (normally the content of classes without an underscore in the name is suppressed).
`@reset ('hist') (,'dialog')`	Resets the size and position of the Debug Console, and clears the history. Arguments: • None: (for example, `@reset`) ◦ All options are reset. • "hist" (for example, `@reset "hist"`) ◦ Only the history is reset. • "dialog" (for example, `@reset "dialog"`) ◦ Only the dialog size and position are reset.
`@wepsel`	Opens the weapon-selection dialog to choose from any of the available weapons. Arguments: • "autoammo" (for example, `@wepsel "autoammo"`) ◦ Supplies the player with infinite ammo.

NOTE

If a predefined pseudo command with the entered name does not exist, an SQF script with the entered name is attempted to start. This script can reside in the mission folder. Also, arguments can be passed to the script (for example, if `{@myscript 100}` was entered, `{_nul = [100] execVM "myscript.sqf"}` is executed).

3.8 Script Debugger Plugin

The Script Debugger Plugin is a tool that can be used by Mission Designers writing SQF code. The tool UI is similar to common Integrated Development Environments (IDEs) with advanced debugging features, such as code breakpoints, step-by-step SQF code execution, variable watch, and so on.

The following aspects are described:

- Installing the Script Debugger (below)
- Script Debugger Layout (below)
- Using the Script Debugger (below)
- Limitations (on page 151)

3.8.1 Installing the Script Debugger

Copy **ScriptDebuggerGUI.dll** from:

`\VBS_Installation\optional\scriptDebugger\`

To:

`\VBS_Installation\plugins64\`

3.8.2 Script Debugger Layout

It is possible to save the current tab layout by going to the Developer menu and selecting `Save settings`. Selecting `Default view` from the View menu restores everything to default.

> **NOTE**
> The layout is saved automatically when VBS4 is closed.

3.8.3 Using the Script Debugger

After starting VBS4 with the plugin installed, if you plan to use the Script Debugger often, the Script Debugger icon can be set up to always appear in the Windows Taskbar.

Image-15: Script Debugger icon

Follow these steps:

1. Click **Show hidden icons**.
2. Drag the Script Debugger icon where you want it to appear in the Taskbar area.

The icon always appears in the Taskbar.

3.8.3.1 Connect

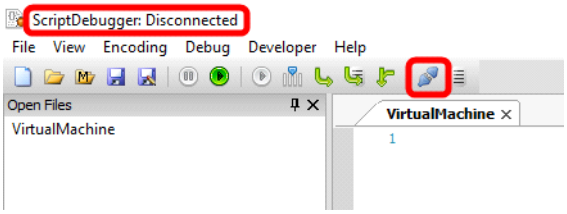
To use Script Debugger, you need to connect it to VBS4.

Follow these steps:

1. Double-click or right-click and select **Show window** to open the Script Debugger.
2. The script debugger starts with **Disconnected** showing in the title bar.
3. To start debugging, click the **Connect** icon in the toolbar.
4. The icon and title bar changes to **Connected**.

Image-16: Connect icon

Image-17: Script Debugger disconnected



3.8.3.2 Load script

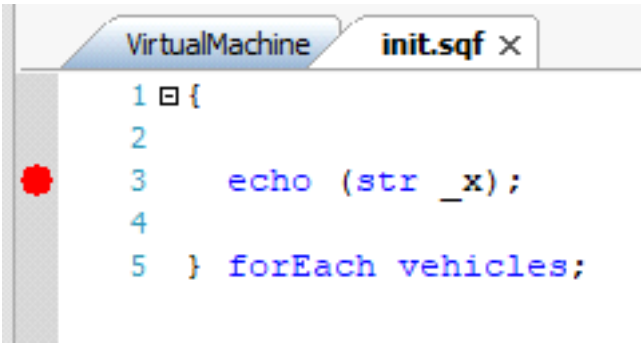
You need to load the mission that is running in VBS4 to debug it.

Follow these steps:

1. In VBS4, load a mission in the Editor that has an `.sqf` script in the mission folder.
2. In **Script Debugger**, click the **Mission Directory**.
3. Expand the tree under the mission to show the `.sqf` files in the mission folder.
4. Double-click one of the `.sqf` files to open it.

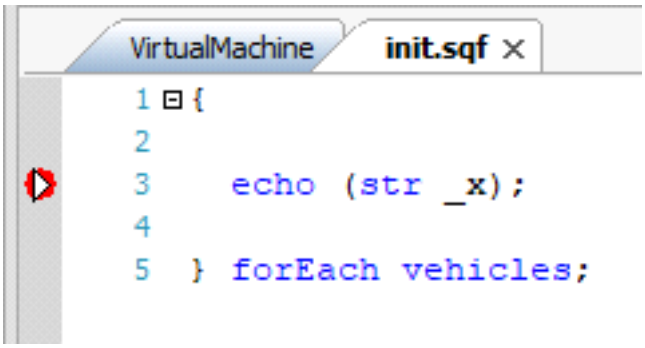
5. You can view the script and click in the bar to the left of the code to set a break point (press the F9 key to toggle a break point) which appears as a red circle.

   Image-18: Break point added before starting mission



6. Start the mission in VBS4, and when the break-point line of code is run VBS4 pauses and highlights the break point with an arrow over the red circle.

   Image-19: VBS4 paused and break point highlighted



Now that the code has run and is paused you can check variables in the **Memory Watcher** tab.

In addition, you can do the following:

- Put a variable name from the script into the **Memory name** column to see the value (you can only see variables that have been defined in the code before the break point).
- Hover over variables to display their current runtime value.
- Press the **Step into** button (or **F11**) in the toolbar to allow VBS4 to run the line of code at the break point and continue to the next line of code (any variables that have been changed can be checked in the **Memory name** column).
- Press the **Play** button in the toolbar to continue until the next break point is evaluated.
- To close a script, click its tab using the **MMB** or click the **X** on the tab.

3.8.3.3 Scripting Errors

Scripting errors (see Error Messages (on page 33)) can be viewed in the **Debug log** tab.

To enable the scripting-error output in the Debug log tab, follow these steps:

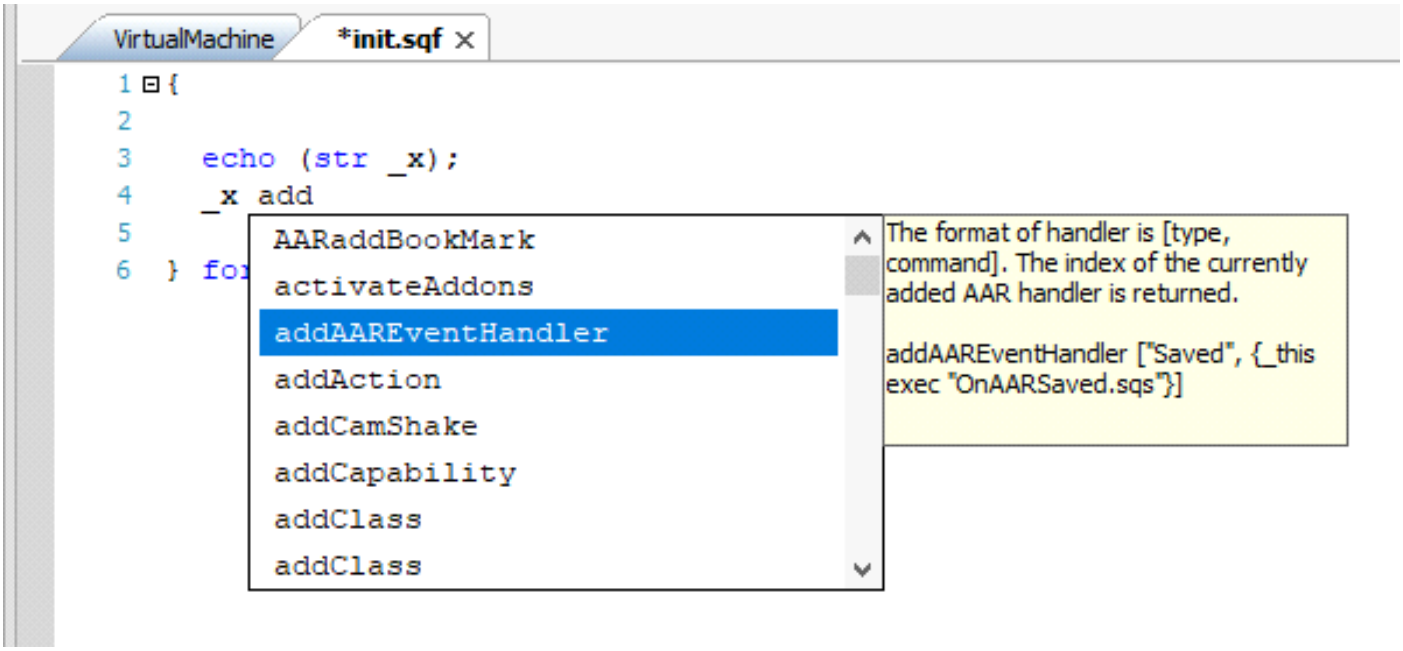
1. In the **Debug log** tab, click **Utils** and make sure **Enable output** is checked.
2. In the **Debug log** tab, click **Utils** and then select **Filters**. Make sure the **Include** field is set to "*****" (without the quotation marks) and that the other fields are blank.

The scripting-error output is enabled in the **Debug log** tab.

3.8.3.4 Add / Edit scripts

New scripts can be added to the mission folder through the menu (**File > New**). Script command help is also provided when entering the code.

Image-20: Script command help



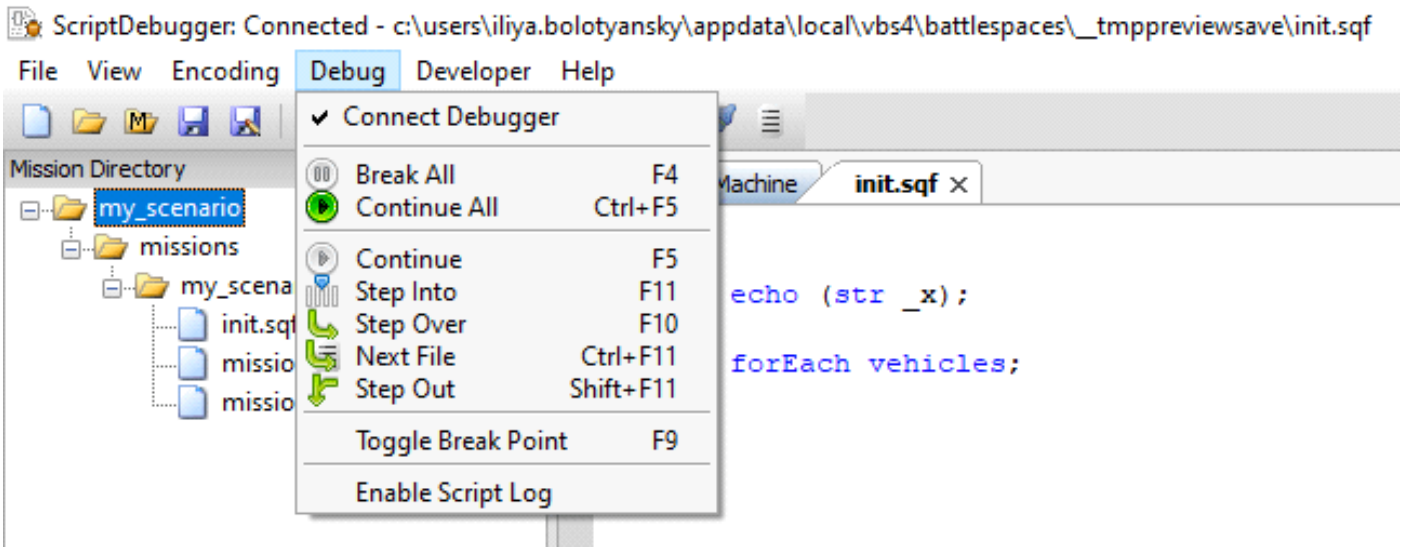
3.8.3.5 Debug Functions

The Step functions are for use with called scripts.

EXAMPLE

```
[_i] call compile preproccessFile "script.sqf"
```

Image-21: Debug functions



Debug Function	Description
Break All	Switches on a toggle that stops every script which runs in VBS4 (as it is possible to run more scripts in parallel). The icon turns red when **Break all** is on.
Continue All	Continue running all scripts. Switches the **Break all** toggle off and continues the currently stopped script.
Continue	Continue the current script only.
Step into	Steps into the next line of code, and goes into called scripts. The called script opens in another tab and the main script and the called script are displayed in the **Call Stack** tab. **NOTE** The **Call Stack** tab is currently not functional.
Step over	Steps over and waits for the highlighted code scope to complete. If the call statement is the current break line, then the call statement is skipped, otherwise works like **Step into** (if in the top level scope) or **Step out** (if within a sub scope).

Debug Function	Description
Step out	Steps out of the current code scope. If the break point is on the top level of a called script, then the rest of the script is skipped.
Toggle Break Point	Sets a break point.
Enable Script Log	Opens a log window to display script processing details. **NOTE** Script logging may impact performance. This impact may be reduced by filtering out **Script Loaded** and **Script Terminated** messages.

3.8.3.6 Encoding

This sets the encoding of the currently open document. Script Debugger attempts to display the file with the selected encoding.

3.8.4 Limitations

Script Debugger has the following limitations:

Limitation Area	Limitations
Debug Console tab	Return value is limited to 5458 characters.
Globals tab	Values are limited to 511 characters.
Locals tab	Values are limited to 511 characters.
Memory Watcher tab	Values are limited to 5458 characters.

4. Advanced Scripting

Some VBS4 scripts make use of more Advanced Data Types (on the next page) than the Basic Data Types (on page 20).

Advanced users of scripting can get familiar with the preprocessor and understand how Preprocessor Commands (on page 163) can be used to make the script code more readable and easier to maintain.

In addition to the VBS4 Function Library (on page 60), you can create your own functions to make reusable code easier to maintain and lower code redundancy. This process is described in: Creating Functions (on page 176).

You can run SQF scripts from VBS Editor and mission events - see Scripting in VBS Editor and Mission Events (on page 185).

You can add your own custom content such as UI dialogs, game sounds, and configure unit identities through Scripting with description.ext (on page 187).

Most scripts and missions are created for a multiplayer (MP) environment. MP considerations for scripting are addressed in: Multiplayer Scripting (on page 200).

To make your code faster and more efficient, various optimization techniques are discussed here: Optimization (on page 203).

4.1 Advanced Data Types

In addition to the Basic Data Types (on page 20), there are some more advanced data types in VBS4. These data types are encountered in commands and VBS4 function-library functions.

Data Type	Description
Code	The Code data type represents data consisting of commands and their parameters (script statements). For example, the contents of SQF and SQS files, are Code. Code variables can also be passed as parameters to commands and / or functions. Code literals are usually represented by enclosing a text in curly brackets: '{' and '}'. Any such code is precompiled by the script engine. Sometimes it may become necessary to first read in certain commands as String. In order to convert the code from the String data type to the Code data type, the command compile (https://sqf.bisimulations.com/display/SQF/compile) can be used.
Color	The Color data type represents a 4-element Array `[red, green, blue, alpha]` with the elements based on the RGB color standard.
Config	The Config data type represents a handle that is used to communicate with configuration files. The handle can represent either the whole configuration file (which can be saved or loaded into a disc file), or a class within it. An example of a configuration path: `configFile>>"CfgVehicles">>"LandRover">>"simulation"`
Control / Display	The Control data type represents a UI control used in UI displays / dialogs (represented by the Display data type). A UI control can be a button, a text field, a check box, and so on.
Editor Object	The Editor Object data type represents any Editor Object (for example, the Editor map).

Data Type	Description
Location	The Location data type represents an extended type of marker: • Locations have a name, a side, a 3D position, a 2D area, and an orientation. • They have a non-scaling map representation (icon and / or text, depending on their type). They require a class definition to define basic properties that can be changed using script commands. Classes are defined in: `\bin\Config.bin\CfgLocationTypes\` • They can be attached to objects, with all of the relevant properties inherited from the object. • They can use setVariable and getVariable (on page 157). • Locations are local in multiplayer, their properties are not synchronized. • Existing locations are set in an islands `.pew` file. When the island is exported to `.wrp` the `islandname.hpp` is also produced - this contains the location names used in the `.pew` file. This `.hpp` file is then incorporated into the configuration for the island using an `#include` statement. • This section included in the island configuration ensures any locations set in the `.pew` file are included in the island during packing. • The terrain configuration cannot be changed, using the location script commands, but it can be found and read. This provides a single command method of finding nearby high points, towns, and so on.
Namespace	The Namespace data type represents a namespace as defined in Namespaces (on page 157).
NetObject	The NetObject data type represents a special type of object used with winch and joint objects. NetObjects, like regular objects, are serialized when the mission is saved, and when loaded re-reference the same winch or joint object.

Data Type	Description
Position	The Position data type represents any coordinate-based position. A position is essentially an Array that has a different number of elements, depending on the position sub-type: • Position - A format descriptor for position arrays containing either 2 or 3 elements. • Position2D - A format descriptor for position arrays containing 2 elements. • Position3D - A format descriptor for position arrays containing 3 elements. • PositionAGL - A format descriptor where the Z-coordinate (elevation) is measured from the nearest "ground" level. The "ground" level is determined in the following order: 1. Snow level 2. Object roadways 3. Water (including tides) 4. Terrain • PositionASL - A format descriptor where the Z-coordinate (elevation) is measured from the sea level (snow is not considered in ASL positions). • PositionATL - A format descriptor where the Z-coordinate (elevation) is measured from the terrain level (ground or sea floor - snow is not considered in ASL positions). • PositionECEF - Earth-Centered, Earth-Fixed (ECEF) position, based on the VBS4-augmented WGS84 (https://en.wikipedia.org/wiki/World_Geodetic_System) standard, called BIWGS84, where the WGS84 axes Y and Z are switched around.
Script Handle	The Script Handle data type is used to identify scripts in script operations called using spawn (https://sqf.bisimulations.com/display/SQF/spawn) or execVM (https://sqf.bisimulations.com/display/SQF/execVM). When the called script is finished, the handle contains a `<NULL-script>` value. At that point the scriptDone (https://sqf.bisimulations.com/display/SQF/scriptDone) command returns true. The script can be terminated by using its handle with terminate (https://sqf.bisimulations.com/display/SQF/terminate).
Structured Text	The Structured Text data type represents a subset of HTML tags (see supported tags (https://sqf.bisimulations.com/display/SQF/Structured+Text)), to allow some formatting within hint (https://sqf.bisimulations.com/display/SQF/hint) messages and dialogs.
Target	The Target data type represents an object that is of special interest to another object. "Targets" are only used internally, by certain script commands (for example, getVehicleLockTarget (https://sqf.bisimulations.com/display/SQF/getVehicleLockTarget)), to keep track of specific entities.

Data Type	Description
Waypoint	The Waypoint data type represents a Waypoint. It is essentially an Array that has two elements `[Group, Number]`: • Group - The group for which the waypoint is created. • Number - The index of the waypoint. **NOTE** Waypoints are based on the Object (on page 21) data type.
Scripted Data Structure	Scripted data structures are custom data structures that can be defined for use with the following SQF struct functions: • fn_vbs_newStructure (https://sqf.bisimulations.com/display/SQF/fn_vbs_newStructure) • fn_vbs_setStructValue (https://sqf.bisimulations.com/display/SQF/fn_vbs_setStructValue) • fn_vbs_getStructValue (https://sqf.bisimulations.com/display/SQF/fn_vbs_getStructValue) The structures must be defined as configuration classes within class `CfgScriptDataStructs`. For more information, see Class CfgScriptDataStructs in the VBS Developer Reference.

4.2 Advanced Variables / Parameters

In addition to the aspects of Basic Variables / Parameters (on page 24), there are several more advanced aspects to consider:

- Namespaces (below)
- setVariable and getVariable (below)
- Scope (on the next page)
- Variables in Multiplayer (on page 161)

4.2.1 Namespaces

A namespace is essentially a container for variables. Within VBS4 there are several different namespaces available. For example, all global variables are stored in a namespace. If no namespace is explicitly specified, then **missionNamespace** is used by default. Within this block, global variables are accessed from the specified namespace. Variables declared in a namespace are local to that namespace. Namespaces are also Advanced Data Types (on page 153).

4.2.2 setVariable and getVariable

Another very useful way to access variables on a namespace is using setVariable (https://sqf.bisimulations.com/display/SQF/setVariable) and getVariable (https://sqf.bisimulations.com/display/SQF/getVariable). These commands can be used to set or get the value of a variable on a specified namespace without having to switch into that namespace.

In addition to using set- and **getVariable** on namespaces, these commands can be used on many different object types to store variables specific to that object instance.

NOTE

These object variables are not the same as namespaces, but the **setVariable** and **getVariable** commands support both namespaces and objects. **setVariable** should be used wherever possible to pass local variables between scripts.

EXAMPLE

```
player setVariable ["myVar","hello world!"];
// Prints "hello world!"
hint (player getVariable "myVar");
// Prints the value of myOtherVar. If it is undefined prints N/A instead.
hint (player getVariable ["myOtherVar", "N/A"]);
```

4.2.3 Scope

Code scope in SQF is very important. Scoping in SQF is hierarchical, similar to other common languages like C / C++ and Java. A scope defines in what contexts a variable is valid. Variables declared within a scope are only valid within that scope or any child scopes. A child scope is created when entering a new code block (for example, function call, if-then statement, for loop). Within a child scope, you can still access variables from the parent scope, but new variables declared within this scope become invisible once the code returns to the parent scope.

EXAMPLE

```
_myString = "hello";
if (_myString == "hello") then
{ // Entering new child scope
  // This line succeeds, and _myNewString contains: This is my string:
"hello"
  _myNewString = ("This is my string: " + str _myString);
};
// With this code by itself, this line fails because _myNewString is
undefined within this scope
hint _myNewString;
```

In order to make this example work, one of two things must be done:

- The hint statement must be moved to within the child scope.
- The variable `_myNewString` must be previously declared within the parent scope.

To declare a variable within a scope, use the following commands:

- private (https://sqf.bisimulations.com/display/SQF/private)
- local (https://sqf.bisimulations.com/display/SQF/local)
- privateAll (https://sqf.bisimulations.com/display/SQF/privateAll)

Operations on that variable within any child scopes are preserved when returning to the parent scope. When used within a child scope, the **private** command declares a new variable for the specified names / identifiers within that scope, even if a variable with the same name exists in the parent scope. In this case, any operations done on the variable within the child scope **ARE NOT** reflected in the parent scope.

4.2.3.1 Using local and private

The following example demonstrates the use of the **local** and **private** commands:

EXAMPLE

```
private ["_myString"];
// Example of private declaration
_myString = "hello";
// This prints: hello
hint _myString;

// Example of direct declaration using local
local _myOtherString = "hi";
// This prints: hi
hint _myOtherString;

sleep 5;
if (_myString == "hello") then
{ // Entering new child scope
   local _myString = "goodbye";
   // This prints: goodbye
   hint _myString;
   sleep 5;
};
// This again prints: hello
hint _myString;
```

- This example uses the **private** command to declare `_myString` as private, and then sets its value to `"hello"`.
- This example uses the **local** keyword to declare a local variable to be private to that scope, at the same time when defining it.
- The `"goodbye"` string in the inner scope does not carry over to the parent scope.

4.2.3.2 Using privateAll and import

Another method that is recommended, especially for function usage, is using **privateAll**, which makes variables private to the current scope and any child scopes. To be able to access variables from above scopes you need to use the import (https://sqf.bisimulations.com/display/SQF/import) command (with some exceptions, for special variables such as `_this`).

EXAMPLE

```
local _myString = "Hello";
local _anotherString = "Hi";

[_anotherString] call {
  privateAll;
  import "_myString";

  // Prints "Hello"
  hint _myString;
  // Gives error because 'anotherString' is undefined
  hint _anotherString;
  // Prints "Hi", as given argument to this code block
  hint (_this select 0);

  if (true) then
  {  // Entering child scope
     _myString = "Goodbye";
  };
  // Prints "Goodbye", because _myString was not declared to be private in
the child scope
  hint _myString;
};

// Prints "Hello", because changes to a private variable in a lower scope do
not carry upwards
hint _myString;
```

- In this example, you need to import `"_myString"` to be able to use variables from upper scopes when **privateAll** is used.
- Variables declared within a scope are not accessible outside it.
- The call (https://sqf.bisimulations.com/display/SQF/call) command creates a new child scope, this also includes calling functions, or external files (see Creating Functions (on page 176)). It is highly recommend to declare all variables used in functions as private, to prevent unexpected changes in one of the parent scopes.

4.2.3.3 Using spawn and execVM

Code executed using the spawn (https://sqf.bisimulations.com/display/SQF/spawn) and execVM (https://sqf.bisimulations.com/display/SQF/execVM) commands neither inherit any variables nor can they be imported. Instead, you should pass wanted variables as arguments, or make them global.

EXAMPLE

```
local _myString = "Hello";
[_myString] spawn {
   // Prints "Hello", by accessing passed arguments
   hint (_this select 0);
   /* Gives an error, because spawned code is not aware of
      local variables from upper scopes
   */
   hint _myString;
}
```

4.2.4 Variables in Multiplayer

By default, all variables are local to each machine in multiplayer. There are a few ways to transfer variables over the network.

SetVariable

The **setVariable** command has an extra parameter that will broadcast the variable and data over the network. This is very useful for making functions MP compatible.

```
_unit = player;
// Assuming the player is in a vehicle right now
_myVariable = vehicle _unit;
_unit setVariable ["VBS_myCar", _myVariable, true];
/* Wait until the player has run away from the vehicle,
   or run this from another script, or even from another computer in MP
*/
sleep 15;
// Get car and set its position
_car = _unit getVariable "VBS_myCar";
_car setPosASL2 (getPosASL2 _unit);
```

publicVariable

Global variables can be broadcast over the network in multiplayer using the publicExec (https://sqf.bisimulations.com/display/SQF/publicExec) command.

For instance machine A expresses:

```
myVar = 1;
```

Later on machine B expresses:

```
myVar = 2;
spublicVariable "myVar";
```

Machine A value of "myVar" is now 2.

> **NOTE**
>
> Be advised that synchronization of new public variable values across the network is not instant and may be delayed by up to a few seconds, depending on script lag and network latency.

4.3 Preprocessor Commands

Before a script file used by VBS4 is actually processed, the engine looks for certain preprocessor commands, and executes them before anything else. Preprocessor commands are mainly used when other script files need to be included for some functionality, and when defining macros.

Preprocessor commands start either with the pound symbol (#) or with a double-underscore (__), and have no semicolon (;) at the end of the line. They can be indented, just as regular script commands. All preprocessor commands are case-sensitive.

Preprocessor commands that are longer than one line require the line-continuation symbol (\) at the end of each line.

NOTE

The line-continuation symbol must be the last symbol in the line, nothing is allowed to follow after it — no comments, and not even spaces.

EXAMPLE

A line-continuation character followed by a space and a comment causes an error.

```
// This comment would cause an error during preprocessing
#define ICONS(icn) icon=\SomeAddon\icons\##icn;  \
model=\SomeAddon\##icn.p3d;
```

This example works if the \ character is the last character before the line-break character.

A file containing any preprocessor commands has to have ANSI encoding.

This chapter covers the following preprocessor commands:

Command	Description
#include (on the next page)	Includes file content into a script.
#define (on page 165)	Defines a macro to be used multiple times in a script.
#undef (on page 169)	Undefines (removes) a defined macro.
#ifdef (on page 169)	Checks if a macro is defined. If it is, a block of code is executed.
#ifndef (on page 170)	Checks if a macro is not defined. If it is not, a block of code is executed.
#else (on page 170)	If the condition in either `#ifdef` or `#ifndef` is not fulfilled, a block of code is executed.
#endif (on page 170)	Closes the preprocessor block opened by `#ifdef`, `#ifndef` or `#else`.

Command	Description
__LINE__ (on page 170)	Substitutes the keyword `__LINE__` with the line number where it is used.
__FILE__ (on page 171)	Substitutes the keyword `__FILE__` with the path to the current processed file.
__EXEC (on page 171)	Executes a script command.
__EVAL (on page 172)	Evaluates an expression.
Comments (on page 172)	Adding comments to preprocessor commands.

For common preprocessor errors, see Common Preprocessor Errors (on page 173).

4.3.1 #include

The purpose of an `#include` statement is to share common definitions among several files (for example, constants or functions).

The included file is first merged into the location of the `#include` statement, after which the combined file (included file and the file with the `#include` statement) is being processed.

Multiple includes can be used, and they can be nested (an included file can include other files).

Included files (just like regular scripts) are interpreted sequentially. That is, any macro that is used in the included file must be defined before the include statement.

Syntax:

```
#include <PathAndFileName>
```

Or:

```
#include "PathAndFileName"
```

EXAMPLE

```
#define DEBUG_OUTPUT
// functions.hpp uses the define DEBUG_OUTPUT
#include "functions.hpp"
```

4.3.2 #define

A #define command creates a macro with either an empty, or an assigned value.

In all occurrences where this macro is used, it is replaced by its assigned value.

Syntax:

```
#define NAME
```

Or:

```
#define NAME value
```

EXAMPLE

```
#define GREETING "HELLO"

hint GREETING;
```

Before the hint command is executed, GREETING is replaced with "HELLO".

NOTE

The replacement is literal. Therefore, the double quotes in the example are also part of the replaced script content.

The defined name is case sensitive. The value parameter is optional. Within #define arguments, the token 'concatenation operator' ##, and 'stringize operator' # may be used.

The scope of a define is limited to the file it is created in, so in order to share a define among several files it is recommended to place them into a separate definition file (for example, with the .hpp extension), and include this file wherever the defines are required.

EXAMPLE

File `mydefines.hpp` is a definitions file:

```
#define FIRST_VALUE 10
#define SECOND_VALUE 11
#define THIRD_VALUE 12
...
```

File `test.sqf` is the script that makes use of `mydefines.hpp`:

```
#include "mydefines.hpp"

if ((player distance enemy) < FIRST_VALUE) then {
    hint format["Enemy closer than %m",FIRST_VALUE]
};
```

4.3.2.1 #define Data Types

Defines can hold different data types, which are interpreted automatically.

EXAMPLE

```
#define A "aaa"
#define B 100
#define C [1]
#define D true

// Returns "STRING, SCALAR, ARRAY, BOOL"
systemChat format[
  "%1, %2, %3, %4",
  typeName A,typeName B,typeName C,typeName D
];
```

4.3.2.2 #define usage in Strings

Defines can be processed as part of a string that uses single quotes.

EXAMPLE

```
#define MY_NUM 2

player sidechat 'The number you are thinking of is MY_NUM';
```

```
#define ISALIVE(_x_) alive _x_
#define PRINT(_y_) player sideChat _y_

if (ISALIVE(player)) then {
    PRINT("hello world")
};
```

Alternatively, the `#define` can be used with the format (https://sqf.bisimulations.com/display/SQF/format) command.

EXAMPLE

```
#define MY_NUM 45

player sidechat format["The number you are thinking of is %1",MY_NUM];
```

4.3.2.3 Macro Expansion

Defines can include tokens which are substituted when expanded.

EXAMPLE

```
#define ADD(x,y) (x + y)
hint str ADD(5,3);  // prints 8
```

However, sometimes it is not clear what should be expanded in the macro. If you need to insert a token into continuous text, delimit it with ##.

EXAMPLE

Having the `#define`:

```
#define ICONS(addon,icn) \
icon=\##addon##\icons\##icn;  \
model=\##addon##\##icn.p3d;
```

And then using:

```
ICONS(SomeAddon,Peter);
ICONS(SomeAddon,Fred);
```

Is expanded during processing to:

```
icon=\SomeAddon\icons\Peter;
model=\SomeAddon\Peter.p3d;
icon=\SomeAddon\icons\Fred;
model=\SomeAddon\Fred.p3d;
```

The purpose of expanded macros is to allow simplified syntax, and to reduce typing load (and the likelihood of typos). In the case of the example above, the `#define` macro creates more explicit, legible files, instead of having to do more typing.

4.3.2.4 Script Commands

It is possible to execute script commands in defines. A `#define` can contain any kind of command. Its result can be used directly in other script statements.

EXAMPLE

The **systemChat** command below outputs "2 is BIGGER THAN 1".

```
#define BIGGER2 (format["2 is %1 than 1",if (2>1) then {"BIGGER"} else
{"SMALLER"}])

systemChat BIGGER2;
```

4.3.2.5 Nested Defines

Defines can be nested, and they are all expanded.

> **NOTE**
>
> Multi-line defines cannot be used inside another `#define`, or passed as a parameter of a macro.

Within nested defines, the # operator works after expanding all defines.

> **EXAMPLE**

```
#define fn_myfunc compile preprocessfile 'myfile.sqf'
#define addquotes(x) #x

// result is: "call compile preprocessfile 'myfile.sqf'"
addquotes(call fn_myfunc);
```

4.3.3 #undef

The `#undef` undefines (deletes) a macro previously set by `#define`.

Syntax:

```
#undef NAME
```

4.3.4 #ifdef

The `#ifdef` controls the execution of the enclosed code (`conditional segment`). Only if a `#define` with the name `NAME` exists, the enclosed code is executed.

Syntax:

```
#ifdef NAME
    conditional segment
#endif
```

EXAMPLE

```
#define DEBUG
_dist = player distance enemy;
#ifdef DEBUG
    hint str _dist;
#endif
```

NOTE

`#ifdef` blocks can be nested.

4.3.5 #ifndef

Executes the enclosed code (`conditional segment`) only if a `#define` with the name `NAME` does not exist.

Syntax:

```
#ifndef NAME
    conditional segment
#endif
```

4.3.6 #else

If a `#define` with the name `NAME` exists, executes `conditional segment 1`, otherwise executes `conditional segment 2`.

Syntax:

```
#ifdef NAME
    conditional segment 1 (used if NAME IS defined)
#else
    conditional segment 2 (used if NAME IS NOT defined)
#endif
```

4.3.7 #endif

Ends a conditional block as shown in the descriptions of `#ifdef` and `#ifndef` above.

4.3.8 __LINE__

The __LINE__ keyword is replaced with the line number in the file where it is found. For example, if __LINE__ is found on the 10th line of a file, the word __LINE__ is replaced with the number 10.

4.3.9 __FILE__

The __FILE__ keyword is replaced with the path to the currently processed file.

4.3.10 __EXEC

Executes a script command.

Syntax:

```
__EXEC(expression)
```

EXAMPLE

```
// Assign an initial value to variable _y
__EXEC (_y = .345);
class Ctrl1 {
    // Use this variable in a config property (via __EVAL)
    y = __EVAL(_y)
    ...
};
// Increment _y
__EXEC (_y = _y + .1)
class Ctrl2 {
    // And use it again
    y = __EVAL(_y)
    ...
};
```

NOTE

__EXEC terminates at the first closing parenthesis ")" it encounters.

EXAMPLE

The following statements are illegal and cause errors.

```
// ILLEGAL!
__EXEC (_val = (_arr select 0)*10);
// ILLEGAL!
__EXEC (_str = (_this select 0) setDamage 1);
```

4.3.11 __EVAL

Evaluates expressions, including previously assigned internal variables.

Syntax:

```
__EVAL(expression)
```

EXAMPLE

```
y = __EVAL (_y);
text = __EVAL (_str1 + _str2);
```

Unlike __EXEC, __EVAL can contain other parentheses, making more complex, and even conditional operations possible.

EXAMPLE

```
x = __EVAL (if (_idx>5) then {0} else {.5});
```

__EVAL does not allow semicolons (;) to be used in expressions.

EXAMPLE

This generates an error:

```
condition = __EVAL (_result=time>5 ; _result);
```

4.3.12 Comments

There are two types of comments that can be used with preprocessor commands.

4.3.12.1 Single-Line Comment

A comment starts with two forward slashes, and ends at the next line break.

NOTE

The preprocessor interprets forward slashes, even in strings.

Syntax:

```
// This line consists only of a comment
#define TRUE 1 // Only this part of the line is a comment
```

EXAMPLE

Having forward slashes in a string can generate an error:

```
// This may generate an error
#define URL "http://www.vbs2.com"
```

To circumvent this issue, the forward slashes can either be defined separately:

```
#define SLASH "/"
#define URL __EVAL("http:" + SLASH + SLASH + "www.vbs2.com")
```

Or the protocol part ("http://") can be excluded from the `#define`, and merged later, as a regular string (for example, using the openURL (https://sqf.bisimulations.com/display/SQF/openURL) command):

EXAMPLE

```
#define URL "www.vbs2.com"
openURL ("http://" + URL)
```

4.3.12.2 Multi-Line Comment

A multi-line comment starts with the characters /* and ends with the characters */

Syntax:

```
/* This
is a multi-line
comment */
```

4.3.13 Common Preprocessor Errors

This section lists the most common preprocessor errors. Most errors return an error code.

Preprocessor Error	Error Code	Description
Incorrect File Encoding	N/A	If a file fails to be preprocessed and produces an error, but the contents of the file works elsewhere, or it seems that the file is not being read, then the file could be encoded incorrectly.

Preprocessor Error	Error Code	Description
Incorrect Condition	0	Incorrect condition. Missing `#endif` after condition statement. **EXAMPLE** `#ifdef something`

Invalid Operation	"not found"	Invalid `#include` operation.
		EXAMPLE
		`/* The file a.hpp cannot be included (usually because it is missing) */`
		`#include "a.hpp"`

Incorrect Include	2	Incorrect `#include` syntax.
		EXAMPLE
		`// Including a number is incorrect syntax`
		`#include 999`

Incorrect Macro Arguments	4	Incorrect macro arguments. Token name does not start with a letter, or closing parenthesis is missing.
		EXAMPLE
		`#define func(100)`
		`/* Incorrect argument, as the closing parenthesis is missing */`
		`#define func(arg`

Preprocessor Error	Error Code	Description

Incorrect Condition	6	Incorrect condition. Preceding `#ifdef` missing when interpreting `#else` or `#endif`.
		EXAMPLE
		`#define something`
		`// Requires #ifdef or #else before #endif`
		`#endif`

Unknown Command	7	Unknown command.
		EXAMPLE
		`// "whatisthis" is not a known preprocessor command`
		`#whatsthis`

Incorrect Argument	11	Incorrect argument in conditional command. Argument is missing or does not start with a letter.
		EXAMPLE
		`// Missing argument`
		`#ifdef`
		`#ifdef 100`

4.4 Creating Functions

A function is a block of code which performs a specific task and is relatively independent of the remaining code. Functions often accept input parameters and sometimes return values back to the script that called them.

Bohemia Interactive Simulations provides a VBS4 Function Library with existing functions that can be used. For more information, see Function Library (on page 60).

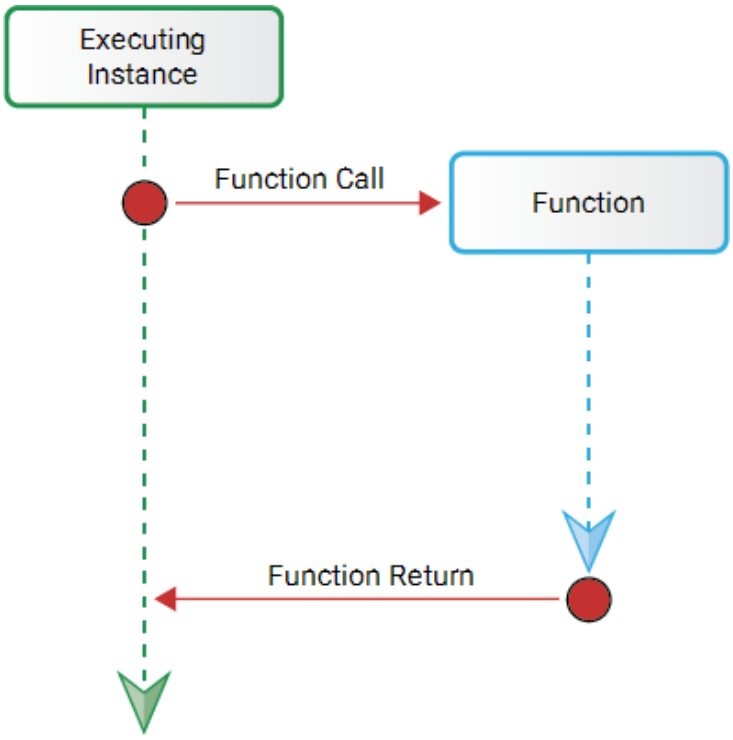
The following is discussed:

4.4.1 Using a Function

Functions can reuse code. The function code can be executed by calling the function from different scripts. When the function code is updated, it is updated for all scripts that call the function. If no function is created, the reused code has to be copied and pasted to all the scripts calling the function, creating redundancies.

4.4.2 Executing a Function

The function execution flow is defined in the following way:



The **Executing Instance** calling the **Function** can be one of the following:

- Other scripts
- Other functions
- Scripting lines in the Mission Editor
- Event Handlers in addon config files

Functions are first loaded as strings from a file using the preprocessFile (https://sqf.bisimulations.com/display/SQF/preprocessFile) command. They are then executed using the call (https://sqf.bisimulations.com/display/SQF/call) or spawn (https://sqf.bisimulations.com/display/SQF/spawn) command.

4.4.2.1 Using the 'call' command

EXAMPLE

```
myFunction = compile preprocessFile "myFunction2.sqf";

_result1 = call myFunction;
_result2 = [1, 2] call myFunction;
```

Functions executed using **call** are run within the executing instance, which waits for the result of the function. Unlike scripts, functions halt all other simulation-engine processes until the function has completed its instructions. This means that functions need to run faster than scripts. It can also mean that if a function takes too long to run it will have an adverse effect on performance.

NOTE

A large function or a CPU-intensive function can cause the simulation to freeze until the function completes its execution.

4.4.2.2 Using the 'spawn' / 'execVM' commands

EXAMPLE

```
myFunction = compile preprocessFile "myFunction2.sqf";

spawn myFunction;
[1, 2] spawn myFunction;
```

Functions may also be executed using **spawn** or execVM (https://sqf.bisimulations.com/display/SQF/execVM), but then the function result is not accessible, making it behave more like a procedure. Spawned functions run asynchronously or alongside the executing instance. This helps to prevent large CPU-intensive functions from causing the simulation to freeze.

4.4.3 Function Return Value

The function return value depends on whether **call** or **spawn/execVM** is used.

4.4.3.1 Using the 'call' command

A function called through **call** returns the result of the last expression executed.

NOTE

Whether the last expression is followed by a semicolon (;) or not has no effect on what is returned.

EXAMPLE

The function file `return.sqf`:

```
STATEMENT 1;
STATEMENT 2;
RETURN_VALUE
```

The script `test.sqf` calling the function:

```
// Value is now RETURN_VALUE
value = call compile preprocessFile "return.sqf";
```

EXAMPLE

The function file `return.sqf`:

```
STATEMENT 1;
STATEMENT 2;
RETURN_VALUE
```

The script `test.sqf` calling the function:

```
// Valid, but RETURN_VALUE is not saved anywhere
call compile preprocessFile "return.sqf";
```

EXAMPLE

```
/* Locally defined function, returning a string that is
    determined within an if...then condition:
 */
_func = {

    _val = _this select 0;
    /* As this is the last command in the function,
       its result will be returned
    */
    if (_val>5) then {
        "bigger"
    } else {
       "smaller"
    };
};

// Outputs "smaller"
player sidechat ([1] call _func);
// Outputs "bigger"
player sidechat ([7] call _func);
```

4.4.3.2 Using the 'spawn' / 'execVM' commands

A function called through **spawn / execVM** returns its handle. The handle can be used with the commands scriptDone (https://sqf.bisimulations.com/display/SQF/scriptDone) or terminate (https://sqf.bisimulations.com/display/SQF/terminate).

4.4.4 Custom Functions in PBOs

You can also define your functions and pack them into PBO addons.

Follow these steps:

1. Create an addon that contains the custom function SQF files, preferably in the `\data\scripts\` sub-folder. For more information, see Packing in the VBS Developer Reference.
2. Create a `config.cpp` file for the addon. For more information, see Creating the Config.cpp and Model.cfg in the VBS Developer Reference.
3. Within the `config.cpp` file, add a section for class `CfgFunctions`.

```
class CfgFunctions
{
  ...
};
```

4. Within class `CfgFunctions`, add a new class for the function.

> **WARNING**
>
> The name of the class should be the exact name the function is supposed to have when being called. Therefore, it should start with `fn_name`, where `name` is an arbitrary but unique namespace to group your functions under.

```
class CfgFunctions
{
  class fn_myAddon_myFunction1
  {
    ...
  };
};
```

5. Ensure that the function class inherits from class `vbs_functions_base`. For more information, see Classes and Inheritance in the VBS Developer Reference.

```
// Include the following for the inheritance to work
class vbs_functions_base;

class CfgFunctions
{
  class fn_myAddon_myFunction1: vbs_functions_base
  {
    ...
  };
};
```

6. Within the function class, point to the SQF file, using the `path` parameter.

```
// Include the following for the inheritance to work
class vbs_functions_base;

class CfgFunctions
{
  class fn_myAddon_myFunction1: vbs_functions_base
  {
    path = "\myAddonPath\data\scripts\myFunction1.sqf";
  };
};
```

7. Optionally, you can execute a function on VBS4 startup by specifying the `preStart` and `postStart` Boolean parameters (values: 0 - false, 1 - true) in class `CfgFunctions`.
   - `preStart` - If set to 1, executes the function on VBS4 startup, before the VBS4 Main Menu (see the Introduction to VBS4 Guide) loads.
   - `postStart` - If set to 1, executes the function on VBS4 startup, after the VBS4 Main Menu loads.

```
class CfgFunctions
{
  class fn_myAddon_myFunction1: vbs_functions_base
  {
    preStart = 1;
    path = "\myAddonPath\data\scripts\myFunction1.sqf";
  };
};
```

> **NOTE**
>
> If a global variable is created with `preStart` during the loading screen appearance, it gets cleared on entering the VBS4 Main Menu. Similarly, when a global variable is created with `postStart` in the VBS4 Main Menu, it gets cleared when exiting the main menu (when loading a Battlespace).
>
> > **TIP**
> >
> > To avoid the clearance of global variables, set the variable in `uiNamespace`:
> >
> > ```
> > uiNamespace setVariable ["myVariable", true];
> > ```

8. Add additional classes for each function, as necessary.
9. Pack the addon and deploy it into the VBS4 addons folder. For more general information, see Packing in the VBS Developer Reference.

The resulting definition should resemble the following:

```
// Include the following for the inheritance to work
class vbs_functions_base;

class CfgFunctions
{
  class fn_myAddon_myFunction1: vbs_functions_base
  {
    path = "\myAddonPath\data\scripts\myFunction1.sqf";
  };
  class fn_myAddon_myFunction2: vbs_functions_base
  {
    path = "\myAddonPath\data\scripts\myFunction2.sqf";
  };
};
```

Once packed and loaded in VBS4, the functions can be called in the same way as any standard SQF function:

```
_return = [_param1, _param2, ..., _paramN] call fn_myAddon_myFunction1;
```

4.5 Scripting in VBS Editor and Mission Events

You can run SQF scripts from VBS Editor and mission events.

This is achieved using the following VBS4 configuration classes:

- **Class CfgEditorInitGlobal** - Allows you to run SQF scripts when:
  - The VBS Editor loads in Prepare Mode.
  - The VBS Editor saves a Battlespace mission in Prepare Mode.
  - The VBS Editor loads a Battlespace mission in Prepare Mode.
  - The VBS Editor clears a Battlespace mission in Prepare Mode.
- **Class CfgRealTimeEditorInit** - Allows you to run SQF scripts when the VBS Editor is initialized during Battlespace mission start in Preview / Execute Mode.
- **Class CfgEditorSave** - Allows you to run SQF scripts when VBS Editor saves a Battlespace mission in Prepare / Preview / Execute / Assess Mode.
- **Class CfgMissionInitGlobal** - Allows you to run SQF scripts when VBS4 starts a Battlespace mission in Preview / Execute Mode.

Each class has the following structure:

```
class Class_Name
{
  parameter_name1 = "SQF_script_path1";
  parameter_name2 = "SQF_script_path2";
  ...
};
```

NOTE

The following considerations apply to each class:

- The names of the parameters, through which the SQF scripts are specified, are arbitrary and only designate the SQF script order of execution.
- The SQF script path can be relative to the Battlespace mission folder, a configuration path, or an absolute path.

For more information about the VBS Editor, see Mission Designer Interface in the VBS4 Editor Manual and Instructor Interface in the VBS4 Instructor Manual.

For more information about VBS4 configuration in general, see Introduction to Configuration in the VBS Developer Reference.

EXAMPLE

Usage of class `CfgEditorInitGlobal`, where the `init.sqf` script (specified with a configuration path) initializes atmospheric parameters in Prepare Mode:

```
class CfgEditorInitGlobal
{
  atmospheric_params_init =
  "\vbs2\vbs_plugins\atmospheric_params\data\scripts\init.sqf";
};
```

Usage of class `CfgRealTimeEditorInit`, where the `init.sqf` script (specified with a configuration path) initializes atmospheric parameters in Preview / Execute Mode:

```
class CfgRealTimeEditorInit
{
  atmospheric_params_init =
  "\vbs2\vbs_plugins\atmospheric_params\data\scripts\initRTE.sqf";
};
```

Usage of class `CfgEditorSave`, where the `saveAdvancedWounding.sqf` script (specified with a configuration path) processes and saves advanced-wounding data in Preview / Execute / Assess Mode:

```
class CfgEditorSave
{
  editor_save_adv_wounds =
  "\vbs2\plugins\advanced_wounding\data\scripts\saveAdvancedWounding.sqf";
};
```

Usage of class `CfgMissionInitGlobal`, where the `initCompass.sqf` script (specified with a configuration path) initializes the Compass (see Standard Equipment in the VBS4 Trainee Manual) in Preview / Execute Mode:

```
class CfgMissionInitGlobal
{
  hud_compass =
  "\vbs2\exclusive\ig\plugin_hud_compass\data\scripts\initCompass.sqf";
};
```

4.6 Scripting with description.ext

The file `description.ext` can be created for a mission (there is only one `description.ext` file per mission), where additional simulation elements can be defined, configured and used in scripts.

The default file location is:

`\Documents\VBS4\Battlespaces\Battlespace_Name\Missions\Battlespace_Name\description.ext`

You can define any custom User Interface (UI) which you want to introduce in a mission (for example, custom / new dialogs). However, `description.ext` also allows the configuration / modification of additional simulation elements (for example, introduce and control custom sounds, or modify various attributes in the appearance of units). The syntax used in `description.ext` is the same as in `config.cpp` (for more information, see `config.cpp` in the VBS Developer Reference).

> **NOTE**
> The Developer Reference is in the `\docs\` folder of the VBS Developer Suite installation.

4.6.1 Respawn Control

You can customize the respawn behavior for killed units in `description.ext`.

> **NOTE**
> Respawn only works for players and playable units.

Respawn Option	Description
`respawn`	Controls the type of respawn - see Respawn Details (on the next page).
`respawnDelay`	Controls the respawn delay (in seconds).

Respawn Option	Description
`respawnVehicleDelay`	Controls the vehicle respawn delay (in seconds). **WARNING** This delay is ignored - vehicles respawn according to the delay set in Init.sqf (on page 37), using the respawnVehicle (https://sqf.bisimulations.com/display/SQF/respawnVehicle) command.
`respawnDialog`	Hides / shows the scoreboard and the respawn countdown timer for the player who is killed with a `respawn=3` ("`BASE`"). The values it can have are: 1 (show the dialog) or 0 (hide the dialog).

EXAMPLE

```
respawn=4;
respawnDelay=5;
respawnDialog=true;
```

4.6.1.1 Respawn Details

The following controls are available for the `respawn` option:

`0 or "NONE"` - No respawn.

`2 or "INSTANT"` - Respawn at the spot where the unit died.

`3 or "BASE"` - Respawn in base. Requires these markers for respawn points:

- `respawn_west`
- `respawn_east`
- `respawn_guerrila`
- `respawn_civilian`

NOTE

For example, you can add markers that have in their name the prefix `respawn_west` and any suffix (for example, `respawn_westABC`, `respawn_west1`, `respawn_west_2`, and so on) for multiple random respawn points. Similarly for `respawn_east`, `respawn_guerrila` and `respawn_civilian`.

Vehicle respawn in base requires these markers for respawn points:

- `respawn_vehicle_west`
- `respawn_vehicle_east`
- `respawn_vehicle_guerrila`
- `respawn_vehicle_civilian`

WARNING

In order for vehicles to respawn, add the following SQF script to Init.sqf (on page 37), that uses the respawnVehicle (https://sqf.bisimulations.com/display/SQF/respawnVehicle) command:

```
// respawns each vehicle with a 10 sec delay
{_x respawnVehicle [10]} forEach allVehicles;
```

`4 or "GROUP"` - Respawn in your group.

NOTE

The unit only respawns in the group if there are AI units available. If there are no units available the unit stays dead, or respawns as a seagull if the **Seagull Respawn** setting is enabled in the Multiplayer.

`5 or "SIDE"` - Respawn as an AI unit on your side.

NOTE

The unit only respawns in the side if there are AI units available. If there are no units available the unit stays dead, or respawns as a seagull if the **Seagull Respawn** setting is enabled in the Multiplayer.

With this respawn type, team switch is also available for any AI-controlled playable units.

4.6.2 Custom User Interface (UI)

You can add new and modify existing UI elements in the mission, using `description.ext`. Just as in any `.hpp` file referenced by `config.cpp`, you can control and configure your own instances / definitions of these UI-element types in `description.ext` (for more information, see Implementing the UI in the VBS Developer Reference):

- Resource
- Display
- Dialog
- Control

When you define your own Dialog, Resource, Display, or Control class, you can refer to it in a script with these frequently used commands:

Resource Commands:

- cutRsc (https://sqf.bisimulations.com/display/SQF/cutRsc)

  Display a resource.

- titleRsc (https://sqf.bisimulations.com/display/SQF/titleRsc)

  Display a resource (can be used in combination with **cutRsc**).

Display Commands:

- displayAddEventHandler (https://sqf.bisimulations.com/display/SQF/displayAddEventHandler)

  Add an Event Handler to display (supports adding multiple Event Handlers per event).

- displayCtrl (https://sqf.bisimulations.com/display/SQF/displayCtrl)

  Return a child control of the display.

- displayRemoveAllEventHandlers (https://sqf.bisimulations.com/display/SQF/displayRemoveAllEventHandlers)

  Remove all the Event Handlers of a specific type from a display.

- displayRemoveEventHandler (https://sqf.bisimulations.com/display/SQF/displayRemoveEventHandler)

  Remove a specific Event Handler from a display.

- displaySetEventHandler (https://sqf.bisimulations.com/display/SQF/displaySetEventHandler)

  Add an Event Handler to a display.

- findDisplay (https://sqf.bisimulations.com/display/SQF/findDisplay)

  Return a specific display.

Dialog Commands:

- createDialog (https://sqf.bisimulations.com/display/SQF/createDialog)

  Create a dialog.

- closeDialog (https://sqf.bisimulations.com/display/SQF/closeDialog)

  Close a dialog.

Control Commands:

- ctrlActivate (https://sqf.bisimulations.com/display/SQF/ctrlActivate)

  Executes the actions attached to a button-based control.

- ctrlAddEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlAddEventHandler)

  Add an Event Handler to a control (supports adding multiple Event Handlers per event).

- ctrlClassName (https://sqf.bisimulations.com/display/SQF/ctrlClassName)

  Return the class name of the control.

- ctrlEnable (https://sqf.bisimulations.com/display/SQF/ctrlEnable)

  Enable / disable a control.

- ctrlEnabled (https://sqf.bisimulations.com/display/SQF/ctrlEnabled)

  Check if a control is enabled.

- ctrlIDC (https://sqf.bisimulations.com/display/SQF/ctrlIDC)

  Return the IDC of a control.

- ctrlParent (https://sqf.bisimulations.com/display/SQF/ctrlParent)

  Return the display of a control.

- ctrlPosition (https://sqf.bisimulations.com/display/SQF/ctrlPosition)

  Return the position of a control.

- ctrlRemoveAllEventHandlers (https://sqf.bisimulations.com/display/SQF/ctrlRemoveAllEventHandlers)

  Remove all the Event Handlers of a specific type from a control.

- ctrlRemoveEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlRemoveEventHandler)

  Remove a specific Event Handler from a control.

- ctrlSetBackgroundColor (https://sqf.bisimulations.com/display/SQF/ctrlSetBackgroundColor)

  Set the background color of a control.

- ctrlSetEventHandler (https://sqf.bisimulations.com/display/SQF/ctrlSetEventHandler)

  Add an Event Handler to a control.
- ctrlSetFocus (https://sqf.bisimulations.com/display/SQF/ctrlSetFocus)

  Set the focus on a non-static control.
- ctrlSetText (https://sqf.bisimulations.com/display/SQF/ctrlSetText)

  Set the text of a control of the currently active dialog.
- ctrlShow (https://sqf.bisimulations.com/display/SQF/ctrlShow)

  Toggles the visibility of a control.
- ctrlText (https://sqf.bisimulations.com/display/SQF/ctrlText)

  Return the text or image path of the control.
- ctrlType (https://sqf.bisimulations.com/display/SQF/ctrlType)

  Return the control type.
- ctrlVisible (https://sqf.bisimulations.com/display/SQF/ctrlVisible)

  Check if a control is visible.

4.6.3 Custom Sounds and Conversations

Custom sounds can be defined in four different sound classes in `description.ext` (for more information about each class, see the VBS Developer Reference):

- CfgMusic
- CfgSFX
- CfgSounds
- CfgRadio

> **NOTE**
> This is part of an old VBS voice technology, and is not part of VBS Radio.

To work with custom sounds, these commands can be used:

Sound Class	Command and Description
CfgMusic	playMusic (https://sqf.bisimulations.com/display/SQF/playMusic) Play music based on sound class name.
CfgSFX	createSoundSource (https://sqf.bisimulations.com/display/SQF/createSoundSource) Create a sound source.

Sound Class	Command and Description
CfgSounds	say (https://sqf.bisimulations.com/display/SQF/say) Makes a unit produce a sound.
	playSound (https://sqf.bisimulations.com/display/SQF/playSound) Play a sound.
CfgRadio	commandRadio (https://sqf.bisimulations.com/display/SQF/commandRadio) Send a message to the command radio channel.
	directSay (https://sqf.bisimulations.com/display/SQF/directSay) Send a message to the direct channel.
	globalRadio (https://sqf.bisimulations.com/display/SQF/globalRadio) Make a unit send a message over the global radio channel.
	groupRadio (https://sqf.bisimulations.com/display/SQF/groupRadio) Make a unit send a message over the group radio channel.
	sideRadio (https://sqf.bisimulations.com/display/SQF/sideRadio) Send a message to the side radio channel.
	vehicleRadio (https://sqf.bisimulations.com/display/SQF/vehicleRadio) Send a message to the vehicle radio channel.
	NOTE This is part of an old VBS voice technology, and is not part of VBS Radio.

Sound definitions follow the standard of a Array Sound, which contains three elements:

```
[sound, volume, pitch]
```

Sound Array Element	Description
`sound`	The path to the sound file (the format can be OGG or WSS). If no extension is specified, WSS is used by default. **WARNING** Only OGG and WSS file formats are accepted. **NOTE** The path must start with a backslash to designate either the root of an addon, or the root of a mission folder. Sub-folders can be specified, but again, have to start with a backslash. Also, the sound is not added to any UI drop-down, such the **Track** setting in Advanced Trigger Settings in the VBS4 Editor Manual.
`volume`	The volume of the sound at the sound source (the location where the sound is coming from).
`pitch`	The pitch of the sound.

After the customized sounds are defined in `description.ext`, they can be used with scripting commands (see say (https://sqf.bisimulations.com/display/SQF/say), playSound (https://sqf.bisimulations.com/display/SQF/playSound), createSoundSource (https://sqf.bisimulations.com/display/SQF/createSoundSource)).

Example:

Use a sound defined in the standard VBS4 content:

```
soundClick[] = {"\vbs2\sounds\animals\dog_bark01", 1, 1};
```

Use a sound defined in a user addon:

```
soundClick[] = {"\myAddon\sounds\beep.ogg", 1, 1};
```

Use a sound defined in the root folder of the mission:

```
soundClick[] = {"\beep.ogg", 1, 1};
```

Defining new sounds under CfgSounds in `description.ext`:

```
class CfgSounds {
  sounds[] = {};
  class wolf1 {
```

```
    name = "";
    sound[] = {"\sound\wolf1.ogg", 1, 1};
    titles[] = {};
  };
  class wolf2 {
    name = "Wolfsong";
    sound[] = {"\sound\wolf2.ogg", 1, 1};
    titles[] = {};
  };
};
```

Defining new sounds under CfgRadio in `description.ext`:

```
class CfgRadio {
  sounds[] = {};
  class RadioMsg1 {
    name = "";
    sound[] = {"\sound\filename1.ogg", db-100, 1.0};
    title =
    "Well all the civilians are now safe in the lodge.  I am ready for your
orders.";
  };
  class RadioMsg2 {
    name = "";
    // .wss implied
    sound[] = {"\sound\filename2", db-100, 1.0};
    title = {$STR_RADIO_2};
  };
};
```

Defining new sounds under CfgMusic in `description.ext`:

```
class CfgMusic {
  tracks[]={};
  class MarsIntro {
    name = "";
    sound[] = {"\music\filename.ogg", db+0, 1.0};
  };

  class Ludwig9 {
    name = "";
    sound[] = {"\music\filename.ogg", db+10, 1.0};
  };
};
```

> **NOTE**
>
> The field `name` can be left blank, as in the previous examples. A name has to be specified only if you wish to access the sound via the environment options of a trigger. The `title` field is the text which is displayed on the screen when the sound file is played.

Example:

Insert a custom OGG sound to your scenario and use a trigger to activate it:

1. In the Editor Object List, select **(F1) Unit** and click a position on the map to place a unit.
2. Select any BLUFOR player unit and click **OK**.
3. In Editor Objects List, select **(F7) Trigger** and click a position on the map, next to the unit.
4. Set the trigger area size, using **Size (Left-Right)** and **Size (Up-Down)**, so that the unit is right next to the area.
5. Set **Activation** to **BLUFOR** and **Activation Type** to **Present**.
6. In **On Activation**, type: `playMusic "TestMusic"`.
7. Click **OK**.
8. Save your scenario and close it.
9. In your scenario folder, add a sub-folder called `\Music\`.

   > **NOTE**
   >
   > The default scenario folder location is:
   >
   > `\Documents\VBS4\Battlespaces\Scenario_Name\`

10. Place your OGG file in the `\Music\` sub-folder, and call the file `TestFile.ogg`.
11. In your scenario folder, create a `description.ext` file with the following content:

```
class CfgMusic {
  tracks[]={};
  class TestMusic {
    name = "Play Music";
    sound[] = {"\music\TestFile.ogg", db+0, 1.0};
  };
};
```

12. Open your scenario again and run it.
13. Walk the player unit into the trigger area.

The custom sound plays.

For custom conversations and their related commands, see the Conversation System (on page 212).

4.6.4 Custom Unit Appearance

To control the appearance of a unit, the `CfgIdentities` class is used for defining custom unit identities in `description.ext`.

The command that controls which unit identity should be used is: setIdentity (https://sqf.bisimulations.com/display/SQF/setIdentity)

CfgIdentities Field	Description
`name`	The name of the unit.
`face`	The face of the unit.
`glasses`	The glasses the unit is wearing (can be only "None" for now).
`speaker`	The voice of the speaker. Can be: • `voice_male_01` • `voice_male_02` • `voice_male_03` • `voice_male_04` • `voice_male_05` • `voice_female_01`
`pitch`	The pitch sets the tone of the voice: • `1.0` - normal voice • `<1.0` - deep voice • `>1.0` - high-pitched voice

EXAMPLE

```
class CfgIdentities
{
  class John_Doe
  {
    name="John Bartholemew Doe";
    face="Face20";
    glasses="None";
    speaker="voice_male_03";
    pitch=1.1;
  };
};
```

4.6.5 Keys

It is possible to define keys for every mission. Keys are used in SP missions for marking the mission as completed in the mission list, and for unlocking hidden content. To mark a mission as completed or unlock hidden content, a key or a number of keys need to be activated via scripting commands activateKey (https://sqf.bisimulations.com/display/SQF/activateKey)and isKeyActive (https://sqf.bisimulations.com/display/SQF/isKeyActive).

Key Option	Description
`keys`	The list of keys needed for `keysLimit`.
`keysLimit`	The number of keys needed from the `keys` list to unlock a mission.
`doneKeys`	The list of keys that have to be activated to mark a SP mission as completed / done.

EXAMPLE

```
keys[] = {"key1","key2","key3"};
keysLimit = 2;
doneKeys[] = {"key4"};
```

4.6.6 Miscellaneous Scenario Customization Options

There are miscellaneous scenario customization options that can be defined in `description.ext` and used for scenario control.

Miscellaneous Option	Description
`briefing`	Enables / disables the briefing screen for SP missions. Can have `0` (disable) or `1` (enable) as its value (for example, `briefing = 0`). **NOTE** If no `briefing.html` is present in the scenario folder, it is skipped anyway. The briefing remains displayed until all the clients connect.
`disabledAI`	Enables / disables the removal all the playable units which do not have a human player in an MP scenario. Can have `0` (disable) or `1` (enable) as its value (for example, `disableAI = 0`).
`showCompass`	Enables / disables the visibility of the compass. Can have `0` (disable) or `1` (enable) as its value (for example, `showCompass = 0`).

Miscellaneous Option	Description
`showGPS`	Enables / disables the visibility of the GPS. Can have `0` (disable) or `1` (enable) as its value (for example, `showGPS = 0`).
`showMap`	Enables / disables the visibility of the Map. Can have `0` (disable) or `1` (enable) as its value (for example, `showMap = 0`).
`showNotePad`	Enables / disables the visibility of the Notepad after the start of the mission. Can have `0` (disable) or `1` (enable) as its value (for example, `showNotePad = 0`).
`showWatch`	Enables / disables the visibility of the watch. Can have `0` (disable) or `1` (enable) as its value (for example, `showWatch = 0`).
`MinScore`	Sets the minimum game score (for example, `MinScore = 0`).
`AvgScore`	Sets the average game score (for example, `AvgScore = 1800`).
`MaxScore`	Sets the maximum game score (for example, `MaxScore = 7500`).

4.7 Multiplayer Scripting

Since most of the scripts run in a multiplayer (MP) environment, aspects of locality should be considered (on which machines in an MP environment does a set of script statements or an entire script runs).

4.7.1 Locality in Multiplayer

The definitions of server and client in VBS4 are:

- **Server** - The dedicated server or the client that is hosting the simulation.
- **Client** - A simulation instance used by a player. Normally, each client is on a separate computer.

MP entities are simulated always on a single computer at a time. The entities are said to be "local" to that computer. In some cases, entities can exist purely locally on a single PC, for example when a vehicle is created using createVehicleLocal (https://sqf.bisimulations.com/display/SQF/createVehicleLocal).

To know if a unit is local, use the local (https://sqf.bisimulations.com/pages/viewpage.action?pageId=1511777) script command.

4.7.1.1 Basic Rules

The basic rules to determine locality are as follows:

- The unit of the player is always local to its client.
- AI units are always local to the client of their leader.
- AI leaders are always local to the server, if placed in the mission editor.
- AI units created after mission start using scripting are local to the computer that issued the command.
- Empty vehicles / objects placed in the mission editor are local to the server.
- Empty vehicles / objects created after mission start using scripting (for example, with createVehicle (https://sqf.bisimulations.com/display/SQF/createVehicle)) are local to the machine that issued the command.

4.7.1.2 Effects of Different Localities

Knowing the locality of a unit is very important, since certain commands only affect local units (for example, moveInDriver (https://sqf.bisimulations.com/display/SQF/moveInDriver)). Other commands have only a local effect, nothing is done outside of the computer where the command was issued.

In general, there are two locality possibilities for the command / function arguments, and two locality possibilities for the command / function effects, indicated by the following icons that are mentioned in the various script commands in the VBS Scripting Reference (https://sqf.bisimulations.com/display/SQF/VBS+Scripting+Reference):

- The object the command is applied to has to be local to the computer the script runs on (for example, playMove (https://sqf.bisimulations.com/display/SQF/playMove)).

- The command can be applied to any object in the mission, independently of its locality (for example, getPos (https://sqf.bisimulations.com/display/SQF/getPos)).

- The command only affects the object (and the effect is only visible) on the local computer (for example, hint (https://sqf.bisimulations.com/display/SQF/hint)).

- The command affects an object independently of its locality (and is visible throughout the MP session) (for example, setPos (https://sqf.bisimulations.com/display/SQF/setPos)).

A unit can change locality during its lifetime. Typical cases are:

- A player dying (his squad AI units are passed to the server).
- Use of the join (https://sqf.bisimulations.com/display/SQF/join) command.
- Units or players entering or exiting a vehicle.
- The TeamSwitch action (for example, with teamSwitch (https://sqf.bisimulations.com/display/SQF/teamSwitch)).
- Use of the selectPlayer (https://sqf.bisimulations.com/display/SQF/selectPlayer) command.

4.7.1.3 isServer

The isServer (https://sqf.bisimulations.com/display/SQF/isServer) command checks whether a local machine is the server running the MP simulation.

EXAMPLE

```
if (!isServer) then { hint "Running on server." };
```

4.7.1.4 publicExec

The publicExec (https://sqf.bisimulations.com/display/SQF/publicExec) command executes the code on all machines in an MP environment where a certain condition is true.

EXAMPLE

Execute the code `"_this setFuel 0"` on a machine where the object `tank1` is local:

```
publicExec ["local _this","_this setFuel 0",tank1];
```

Execute the code `"player setDamage 1"` on all machines which belong to side `EAST`:

```
publicExec ["side player == EAST","player setDamage 1"];
```

Execute the code `"_this setVelocity [0,0,10]"` only on the server machine:

```
publicExec ["isServer","_this setVelocity [0,0,10]"];
```

4.8 Optimization

Script execution has an effect on simulation performance (that is, how fast the VBS4 engine runs). If the script does redundant work, meaning that it could have been written more efficiently to run faster and achieve the same results, the simulation might run slower (with a reduced frame rate) and sometimes freeze up for a certain period of time. To avoid this, code efficiency has to be considered. Below are some key points for script optimization and code efficiency.

4.8.1 If you have written it twice, put it in a function

Pre-compilation by the simulation engine can save up to 20 times the amount of processing required to execute the code, even if the initial processing time is slightly lengthened. If you have written some script code twice, or if there is a loop which is consistently compiled (perhaps a script run by the execVM (https://sqf.bisimulations.com/display/SQF/execVM) command), make it into a function.

The 'preprocessFile' Command

```
myfunc = compile preprocessFile "myfile.sqf";
```

The preprocessFile (https://sqf.bisimulations.com/display/SQF/preprocessFile) command caches what it has done, so loading a file once will load it into memory, therefore if you would like to refrain from using global variables, and yet would like to have a function precompiled but not saved, you could use:

```
call compile preprocessFile "file";
```

The loss of performance in this case would only be the compilation time of the script file "`file`" and then the call of the code itself.

4.8.2 Run it Locally

In a multiplayer environment, you can execute the code only on the computer where the specified object is local.

```
executeWhereLocal [tank1, "_object setVelocity _this",[0,0,10]];
```

4.8.3 Optimizing Code Length

If any script or function is longer than 200-300 lines, then you may need to rethink the structure of the script itself, and whether it does not exceed the scope of the functionality required.

Fewer statements => Faster code

Optimize the code by removing redundant statements. The following code examples do the same thing, but the latter is 1.5 times faster.

EXAMPLE

```
_arr = [1,2];
_one = _arr select 0;
_two = _arr select 1;
_three = _one + _two;
```

```
_arr = [1,2];
_three  = (_arr select 0) + (_arr select 1);
```

4.8.4 Condition Optimization

Here is an example with 4 conditions in an if-statement:

```
if (_group knowsAbout vehicle _object > 0 &&
     alive _object && canMove _object &&
     count magazines _object > 0) then {
       // custom code
};
```

You may expect the engine to stop evaluating the remaining conditions after the first condition (`_group knowsAbout vehicle _object`) is, for example, found to be false, but that is not the case. The engine continues evaluating the remaining conditions even if any of the previous conditions were evaluated as false.

```
if (_group knowsAbout vehicle _object > 0) then {
    if (alive _object && canMove _object &&
        count magazines _object > 0) then {
          // custom code
    };
};
```

In the above modified example, the engine only continues reading the conditions after the group has some knowledge about the object. Alternatively you can use the so-called lazy evaluation syntax. If normal evaluation syntax is `(condition1 .. condition2 .. condition3 .. ...)`, lazy evaluation syntax is `(condition1 .. {condition2} .. {condition3} .. ...)`. Let us look at the above modified example using lazy evaluation:

```
if (_group knowsAbout _vehicle object > 0 &&
     {alive _object} && {canMove _object} &&
     {count magazines _object > 0}) then {
        // custom code
};
```

If Else If Else If Else ...

If you cannot escape from using a switch-statement to replace a chain of if-then-else-statements, then try to rethink the functionality. Especially if only one condition needs to be fulfilled.

On the other hand, the switch-statement is slower than the if-then-else-statement. To keep tidiness of the switch statement and the speed of the if-statement, use the if-statement with exitWith (https://sqf.bisimulations.com/display/SQF/exitWith) combined with call (https://sqf.bisimulations.com/display/SQF/call):

```
call {
    if (CONDITION1) exitWith {
      STATEMENT1;
    };
    if (CONDITION2) exitWith {
      STATEMENT2;
    };
    if (CONDITION3) exitWith {
      STATEMENT3;
    };
    STATEMENT4;
};
```

Here is a list of various if-statements, ordered from fastest to slowest:

1. `if () then {}`
2. `if () exitWith {}`
3. Either of the two:
    a. `if () then {} else {}`
    b. `if () then [{},{}]`

4.8.5 Optimizing with Constants

If you are using a hard coded constant more than once, use a preprocessor #define.

Example:

Either:

```
a = _x + 1.053;
b = _y + 1.053;
```

Or:

```
_buffer = 1.053;
a = _x + _buffer;
b = _y + _buffer;
```

Becomes:

```
#define BUFFER 1.053
_a = _x + BUFFER;
_b = _y + BUFFER;
```

This way, the constant is modified only in one place (in the #define), instead of in every occurrence of the constant.

4.8.6 Making Loops Faster

These first two loop types are nearly identical in speed.

Loop 1:

```
for "_y" from # to # step # do { ... };
```

Loop 2:

```
{ ... } foreach [ ... ];
```

The next two loop types are about 3 times slower:

Loop 3:

```
while { expression } do { code };
```

Loop 4:

```
for [{ ... },{ ... },{ ... }] do { ... }
```

The waitUntil-loop can be used when you want some code to only run once per frame, which can be handy for limiting scripts that use many resources.

Loop 5:

```
waitUntil {expression};
```

10,000 iterations limit in loops

A while-loop is limited to 10,000 iterations in a non-scheduled environment. In a scheduled environment such a limit does not apply.

count vs forEach

Both commands / statements, forEach and count, step through the array of elements one by one and both commands / statements contain a reference to the currently iterated element in the special `_x` variable. However, the count-loop is a little faster than the forEach-loop, but it does not have the `_forEachIndex` variable and the code inside the count-loop expects a Boolean or Nothing at the end of the loop block while it returns a Number.

EXAMPLE

```
{consoleLog _x} count [1,2,3,4,5,6,7,8,9];
// Is faster than
{consoleLog _x} forEach [1,2,3,4,5,6,7,8,9];
```

```
_someoneIsNear = {_x distance [0,0,0] < 1000} count allUnits > 0;
// Is still faster than
_someoneIsNear = {
    if (_x distance [0,0,0] < 1000) exitWith {true};
    false
} forEach allUnits;
```

Nested Loops

Try to avoid nested loops, when possible. For example, a loop within a loop takes `N` iterations, where:

`N = (number of outer loop iterations) x (number of inner loop iterations)`

4.8.7 Threads

There are two ways you can run your code: in a scheduled and non-scheduled way.

Some basic examples:

- Triggers execute in non-scheduled environment
- All pre-init code executes without scheduling
- Finite-State-Machine (FSM) conditions are evaluated without scheduling
- Event handlers (for units and the UI) are executed without scheduling
- SQF code called from an SQS code executes without scheduling

The 3ms Run Time

A scheduled script runs for exactly 3ms before it is put in suspension to be resumed in the next frame. This interrupted execution repeats until the script code finishes executing. Suspension time depends on FPS. For example, at 20 FPS the suspension is 50ms.

This means that if a scheduled script cannot be completed under 3ms, the execution can stretch for an undefined amount of time, depending on engine load, FPS and other non-scheduled scripts running in parallel.

Scheduled scripts always start with a slight delay, depending on engine load.

4.8.8 Deprecated / Slow commands

Adding Elements to an Array

The set (https://sqf.bisimulations.com/display/SQF/set) command is currently the fastest command to place an element in an Array.

EXAMPLE

```
_a set [count _a,_v]
```

Is about twice as fast as:

```
_a = _a + [_v]
```

Removing Elements from an Array

When you want to remove the first element in an Array that was also the first element to be inserted into the Array, removal using the **set** command works best, even if it makes a copy of the new Array.

EXAMPLE

```
ARRAYX set [0, objnull];
ARRAYX = ARRAYX - [objnull];
```

4.8.9 Comparing Arrays

To compare two Arrays and see if they are equal, use fn_vbs_arrayCompare (https://sqf.bisimulations.com/display/SQF/fn_vbs_arrayCompare) from the VBS4 function library.

4.8.10 Position related commands

The ASL position commands getPosASL2 (https://sqf.bisimulations.com/display/SQF/getPosASL2) is about twice faster than the commands getPos (https://sqf.bisimulations.com/display/SQF/getPos), position (https://sqf.bisimulations.com/display/SQF/position).

The command nearEntities (https://sqf.bisimulations.com/display/SQF/nearEntities) runs much faster than the command nearestObjects (https://sqf.bisimulations.com/display/SQF/nearestObjects). For example, if the object range radius was set to more the 100 meters, it is highly recommend to use **nearEntities** instead of **nearestObjects**.

NOTE

nearEntities only searches for objects which are alive. Killed units, destroyed vehicles, static objects and buildings are ignored by the **nearEntities** command.

4.8.11 Measure how fast it takes to run your code

You can use the diag_tickTime (https://sqf.bisimulations.com/display/SQF/diag_tickTime) command to measure how fast (in seconds) it takes to run your code to decide which code alternative works faster for you.

NOTE

The diag_tickTime (https://sqf.bisimulations.com/display/SQF/diag_tickTime) SQF command is only available in special VBS executables. For more information, contact support@bisimulations.com.

EXAMPLE

```
_benchmark_start = diag_tickTime;
// Put your code to be tested here
...
_benchmark_end = diag_tickTime;
systemChat format [
  "The code took %1 sec to run.",
  _benchmark_end - _benchmark_start
];
```

5. Scripting Tutorials

This chapter contains advanced miscellaneous topics of scripting such as:

If you wish to enroll in a VBS4 scripting course, visit:

- Standard VBS4 scripting courses are available at: https://bisimulations.com/training

5.1 Adding Videos to a Mission

Mission editors are able to create dialogs that display a video within the simulation.

> **NOTE**
>
> The video must be in the `.ogv` format.

The video dialog is defined in `description.ext` (see Custom User Interface (UI) (on page 190)). For the dialog parameters, see Configuring UI Elements and UI Configuration in the VBS Developer Reference.

To get a dialog to play an `.ogv` video, these parameters must be defined in the dialog:

```
autoplay = 1;
loops = 10;
```

- **autoplay** - Controls whether the video should play on startup (0 = false, 1 = true).
- **loops** - The number of times the video loops before it stops playing.

The **Style** must be `ST_PICTURE`.

To play the video, run the ctrlSetText (https://sqf.bisimulations.com/display/SQF/ctrlSetText) command on the control, with the text being the path to the video.

> **EXAMPLE**
>
> ```
> _display = findDisplay 20000;
> _ctrl = _display displayCtrl 20000;
> _ctrl ctrlSetText "Video1.ogv";
> ```

The video can be located within the mission folder `pbo` sub-folder or at some other location on the computer (for example, an addon PBO).

5.2 Conversation System

VBS4 has a conversation system. It allows mission designers to create scripted conversations between the player and the AI units.

It can also be used to converse between two players using typed messages that only those two players can see.

Image-22: Conversation system

The conversation system can be implemented as a Finite State Machine (FSM). For more information on how to create FSMs, see:

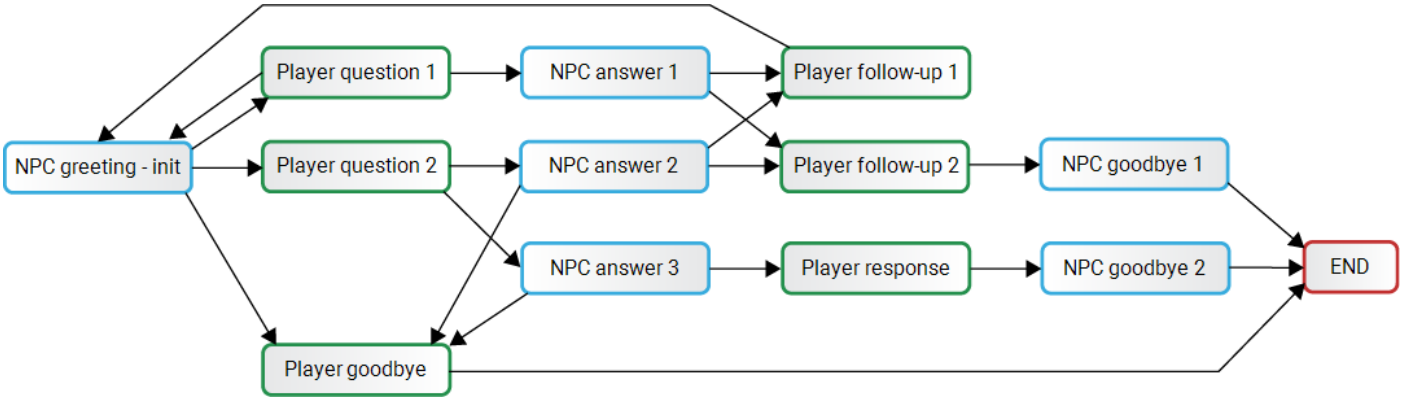
- **FSM Editor** in the VBS Developer Reference.
- **FSM Conversations in VBS** in the VBS Developer Reference.

> **NOTE**
>
> The Developer Reference is in the `\docs\` folder of the VBS Developer Suite installation.

5.2.1 Player to AI

The player-AI conversation can be a simple question-answer scenario, or it can have multiple options and branches, each of which may depend on external criteria.

Image-23: Example conversation flow



All conversations are defined in the `cfgConversations` class. This definition can reside either in an addon (to be used in multiple missions), or in the `description.ext` file of a mission.

> **WARNING**
>
> Make sure to save the mission in VBS4 after creating the `description.ext` file. Otherwise, VBS4 cannot find the file during mission preview.

One or more conversation classes can be defined in `cfgConversations`, each containing several sub-classes (phrases) which define what the player or the AI can say. Each of these phrase classes can also contain transition definitions. These define how the conversation should continue after the phrase has been shown.

For example, the first conversation class shown could contain a greeting by the AI, and multiple questions (transitions) that the player could ask.

Once one of those questions has been selected by the player, the transition definition for that phrase would point to one or more possible answers by the AI.

A basic conversation, where the player asks an NPC for his name, is shown in the (simplified) definition below:

```
class cfgConversations
{
  // Name of conversation (there can be several)
  class TalkDemo
  {
    // Class the conversation starts with
    initState = "Hello";
    class Hello
```

```
    {
      // Initial message
      text = "Hello there.";
      class Transitions
      {
        /* Only one question / transition available,
           which points to 'Question'
        */
        class Ask
        {
          to = "Question"
        };
      };
    };

    // Player asks for the name
    class Question
    {
      text = "What's your name?";
      class Transitions
      {
        // Only one answer / transition available, which points to 'Answer'
        class Reply
        {
          to = "Answer"
        };
      };
    };

    // Response by the AI
    class Answer
    {
      text = "I'm Richard.";
      class Transitions
      {
      };
      // End of conversation - no further transitions
    };
  };
};
```

5.2.2 Player to player

The conversation system can also be used to conduct a conversation between two players.

Only the main conversation class has to be defined for those types of interactions, along with an initial "greetings" phrase. Once the initial phrase has been shown, the conversation consists only of messages typed back and forth between the players (so the transitions class should be empty).

It is possible to temporarily take control of an AI unit, using conversationTakeControl (https://sqf.bisimulations.com/display/SQF/conversationTakeControl), in order to switch the conversation from the player-AI mode to the player-player mode, so that another person (most likely an admin) can handle the conversation of an AI unit. Once a unit is player-controlled, the conversation automatically switches to the player-player mode, where messages are typed directly (rather than being chosen from a list).

5.2.3 Properties

The following properties are used for the different sub-classes.

5.2.3.1 Main conversation

Property	Data Type	Description
`colorParticipant0`	Color	The dialog color for AI messages.
`colorParticipant1`	Color	The dialog color for player messages.
`initState`	String	The class the conversation should start with.

5.2.3.2 Phrase

Sub-class of main conversation:

Property	Data Type	Description
`name`	String	The name of the conversation (required, but not currently utilized).
`title`	String	The phrase caption shown in the selection menu.
`text`	String	The displayed message text. Can contain variables (`%1`, `%2`, ...) which are defined in `textParams`.
`textParams`	Array	An array of Strings representing the values to be used by the variables referenced in the `text` property.
`participant`	Number	Who is saying the phrase: • 0 - AI • 1 - Player

Property	Data Type	Description
`canRepeat`	Boolean	Can the phrase node be reached multiple times: • true - yes • false - no
`statement`	String	The scripted action to be performed when the phrase node gets played ('`_this select 0`' represents the player, '`_this select 1`' represents the AI object).

5.2.3.3 Transition

The transition is a sub-class of a phrase definition. It can be empty, or contain one or more sub-classes, which point to the next phrase. If the transition points to a phrase an NPC should say, then there should be only one possible transition (only one defined, or only one whose condition is true):

Property	Data Type	Description
`priority`	Number	The display priority. A lower priority number shows up higher in the list.
`to`	String	The name of the next phrase class to use.
`condition`	String	The condition that returns a Boolean, and which must return true in order for the transition to be used.

5.2.4 Example

Here is a complete conversation, with all required properties, where the player asks for the time:

```
class cfgConversations {
  class ConversationCiv {
    colorParticipant0[] = {1,.5,0,1};
    colorParticipant1[] = {0,1,.2,1};
    initState = "StartConversation";
    class StartConversation {
       name = "StartConversation";
       // Initial message
       text = "Hello. Can I help you?";
       // Said by AI
       participant = 0;
       // Can be repeated
       canRepeat = true;
       // Play greeting animation
       statement = "(_this select 1) switchGesture 'GestureHi'";
       class Transitions {
         // Only one question / transition, to ask for the time
         class Time {
           priority = 0;
           // Class name of next phrase to play
           to = "Time_Ask";
           condition = "";
        };
      };
    };
    // player asks for the time
     class Time_Ask {
       name = "TimeAsk";
       title = "Ask for the time";
       text = "Hello. What's the time?";
       participant = 1;
       canRepeat = true;
       statement = "";
       class Transitions {
       // Only one possible response by AI, defined in 'Time_Reply'
       class Time {
         priority = 0;
         to = "Time_Reply";
         condition = "";
       };
     };
   }
    // AI response to time inquiry
     class Time_Reply {
```

```
        name = "Time_Reply";
        // Reply, using several variables
        text = "Right now it's %1:%2 and %3 seconds.";
        // Variable values
        textParams = ["date select 3","date select 4","systemTime select 5"];
        participant = 0;
        canRepeat = true;
        // Animation to have AI look at watch
        statement = "(_this select 1) playMove 'AmovPercMstpSnonWnonDnon_seeWatch'";
        class Transitions {
        };
        // No further transitions - end of conversation
    };
  };
};
```

5.2.5 Implementation

Once a conversation class has been defined, it can then be initiated by the conversationStart (https://sqf.bisimulations.com/display/SQF/conversationStart) command (most likely activated by either a user action or a trigger. This command opens the conversation dialog, where the player can chose from the available phrases, and the AI responds accordingly (for navigation, see Usage (on the next page)).

A conversation can be aborted by using the conversationEnd (https://sqf.bisimulations.com/display/SQF/conversationEnd) command. As the conversation system currently does not consider the distance between the speakers or even whether they are alive or not, the mission designer might have to handle these situations manually.

There are also several event handlers available, to keep track of conversations: OnConversationStart (on page 112), OnConversationEnd (on page 113), and OnConversationSay (on page 113).

The main commands relevant to end-user utilization of the conversation system are **conversationStart**, **conversationEnd**, inConversation (https://sqf.bisimulations.com/display/SQF/inConversation), and **conversationTakeControl**. Some other conversation-related commands are available, but they are only used for the internal management of the conversation dialog system, and unless you plan to implement your own conversation dialog, they can be ignored: conversationSay (https://sqf.bisimulations.com/display/SQF/conversationSay) and conversationShowPage (https://sqf.bisimulations.com/display/SQF/conversationShowPage).

5.2.6 Usage

Once the conversation dialog is up, the player is presented with the possible phrases on the right side of the dialog. Clicking on the caption (or the plus sign on the right) shows the whole phrase associated with this selection.

If more phrases are available and they do not fit into the dialog, then the other pages can be selected via the `NEXT` and `PREV` buttons at the bottom.

When a phrase has been selected, a `SAY` button appears. Pressing this button shows the associated sentence in the left pane, as well as the defined response by the AI.

If further phrases are defined, they can now be selected on the right side; or the dialog can be closed by pressing **Esc**.

For player-player conversations, once the initial phrase has been shown, either player can type a message in the right side of the conversation dialog. When the `SAY` button or the **Enter** key is pressed, it shows up in the left side of the dialog of the other player.

> **NOTE**
>
> The "AI" label next to the dialog pages indicates an AI conversation. This label changes to "HUMAN" when talking to another player.

5.2.7 Using a Custom Conversation in Intelligence Reports

You can also use the Intelligence Reports Editor Object (EO) (see Intelligence Reports in the VBS4 Editor Manual) with a custom conversation.

Follow these steps:

1. Place any unit on the map.
2. From the **Editor Objects List**, place an **Intelligence Reports** Editor Object on the map.
3. In the **Custom conversation** drop-down, select **true**.
4. Specify the name of your conversation class (in the case of the Example (on page 217), it is `ConversationCiv`).
5. Right-click the **Intelligence Reports** Editor Object, select **Link to Unit**, and click the unit you wish to add the conversation system to.

The unit has a conversation system.

5.3 AAR Scripting

The After-Action Review (AAR) system can be scripted. For example, you can automate the starting and stopping of an AAR recording, saving bookmarks and also write custom text to the network event log. If you wish to script AAR recording, you may use the following commands.

- AARrecord (https://sqf.bisimulations.com/display/SQF/AARrecord)

  Starts the recording.

  > **NOTE**
  >
  > Clears the previous recording, so be sure to save your previous recording before calling `AARrecord` again.

  > **WARNING**
  >
  > This command is obsolete. Instead, use the `fn_vbs_aar_recordStart` function to start the recording, which is reflected correctly in the Editor (Execute Mode).

- fn_vbs_aar_recordStart (https://sqf.bisimulations.com/display/SQF/fn_vbs_aar_recordStart)

  Starts the recording. Use this function instead of `AARrecord`.

- AARisRecording (https://sqf.bisimulations.com/display/SQF/AARisRecording)

  Check if the AAR recorder is recording.

- AARcurrentTime (https://sqf.bisimulations.com/display/SQF/AARcurrentTime)

  Returns the current length of the recording.

- AARstopRecording (https://sqf.bisimulations.com/display/SQF/AARstopRecording)

  Stops the recording.

- AARsave (https://sqf.bisimulations.com/display/SQF/AARsave)

  Saves the recording in a file.

- AARaddBookMark (https://sqf.bisimulations.com/display/SQF/AARaddBookMark)

  Saves a bookmark in the recording.

Mission-Event Log

A Mission-Event log is automatically written on the server when a network mission ends. The log file is named `mpreport.txt`, and it is located in the `\VBS4 Installation\` folder.

It is possible to write custom events to the log through scripting. The following commands apply:

- VBS_addHeader (https://sqf.bisimulations.com/display/SQF/VBS_addHeader)

  Writes text to the header of the Mission-Event log.

- VBS_addEvent (https://sqf.bisimulations.com/display/SQF/VBS_addEvent)

  Writes a new event to the Mission-Event log.

  > **NOTE**
  >
  > Typically, the **VBS_addEvent** command would be called through a trigger in the **On Activation** field.

- VBS_addFooter (https://sqf.bisimulations.com/display/SQF/VBS_addFooter)

  Writes text to the footer of the Mission-Event log.

EXAMPLE

Suppose the following line of script code is entered in the **On Activation** field:

```
VBS_addHeader "test head"; VBS_addEvent "test event"; VBS_addFooter "test footer"
```

The following data would be written to `mpreport.txt`:

> **NOTE**
>
> The `Killed` event can be ignored, it is written automatically by VBS4 when a unit dies.

```
----------**Start**----------
Mission: myMission.Intro
Players: BLUFOR (3)
Start mission: 2008/4/3, 16:27:51
test head
----------**Events**----------
0:00:14 : Nathan Jones (AI) killed by Pete at [2518, 2570]
0:00:24 : test event
----------**Summary**----------
Total amount of deaths: BLUFOR (1)
Total amount of kills: BLUFOR (1)
test footer
----------**End**----------
```

5.4 Procedural Textures

Procedural textures are textures that are generated by VBS4 and applied to objects, using SQF, on the basis of specifying specially-formatted strings.

Procedural textures take very little space on the hard drive, but still use video memory.

The main SQF commands used to generate and apply procedural textures are:

- setObjectTexture (https://sqf.bisimulations.com/display/SQF/setObjectTexture) / setObjectTextureGlobal (https://sqf.bisimulations.com/display/SQF/setObjectTextureGlobal)
- getObjTexture (https://sqf.bisimulations.com/display/SQF/getObjTexture)

5.4.1 Procedural-Texture Types

There are several procedural-texture types:

- Color (on the next page)
- Text (on the next page)
- Render to Texture (RTT) (on page 225)
- Fresnel / Fresnel Glass (on page 226)

All of these types have the following optional common parameters in the procedural-texture string:

Parameter	Description
`format`	Texture color format. Can be: • `rgb` • `argb`
`width`	Number of pixels along the X-axis, in power of two (for example, 1, 2, 4, 8, 16, 32, 64, 128, and so on).
`height`	Number of pixels along the Y-axis, in power of two (for example, 1, 2, 4, 8, 16, 32, 64, 128, and so on).
`mipmapsNum`	Number of Mipmaps (https://en.wikipedia.org/wiki/Mipmap) to generate.

5.4.1.1 Color

Creates a color texture. The following Color parameters can be specified in the procedural-texture string:

Parameter	Description
`r, g, b, a`	RGBA texture channels (Red, Green, Blue, Alpha).

Example

```
_unit setObjectTexture ["swap_tex_01","#(argb,8,8,3)color(1,0,0,1)"];
```

5.4.1.2 Text

Creates a text texture. The following Text parameters can be specified in the procedural-texture string:

Parameter	Description
`text`	Text to display, delimited by quotation marks.
`width`	Text texture width.

Parameter	Description
`height` (Optional)	Text texture height.
`fontSize` (Optional)	Positive integer specifying the font size in points.
`fontFamily` (Optional)	Name of an installed font family. May contain spaces, but must not contain commas. Falls back to another font, if the specified font family not installed.
`fontColor` (Optional)	Font color as a hexadecimal representation of RGBA, case insensitive.
`fontStyle` (Optional)	Font style. Can be: • `normal` • `oblique` • `italic`
`fontWeight` (Optional)	Font weight. Can be: • `normal` • `bold` • `heavy`

Example

```
_unit setObjectTexture [
  "swap_rank",
  "#text(""hello, world!"", 1024, 1024, 200, Arial, FFFFFFFF, normal, bold)"
]
```

5.4.1.3 Render to Texture (RTT)

Creates a texture on a Render to Texture (RTT) surface. The following RTT parameters can be specified in the procedural-texture string:

Parameter	Description
`surface`	Name that is used as a reference in the camera script.
`type`	RTT texture type. Can be: • `mirror` • `camera`
`aspectRatio`	Aspect ratio of the texture. For example, a value of 1.333 gives an aspect ratio of 4:3.

Example

```
camera = "camera" camCreate[0,0,0];
camera cameraEffect["internal","back", "mycam"];
player setObjectTexture [
  "swap_rank",
  "#(argb,512,512,3)rendertotexture(mycam,camera,1.333)"
];
```

5.4.1.4 Fresnel / Fresnel Glass

Creates a texture based on the Fresnel equation for light. The following Fresnel / Fresnel Glass parameters can be specified in the procedural-texture string:

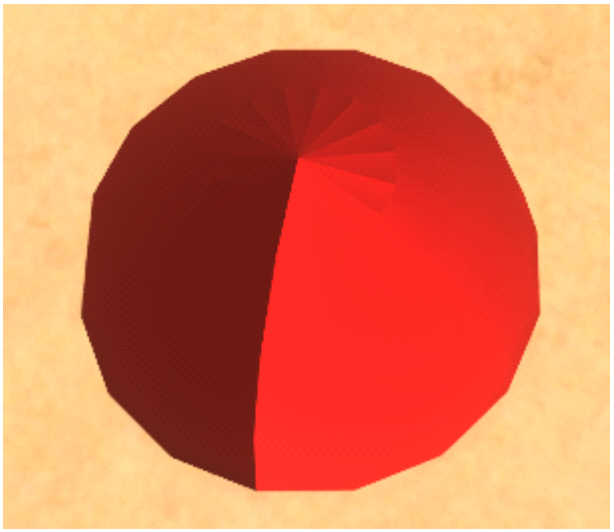
Parameter	Description
`refIndex`	Refractive index.
`absCoef`	Absorption coefficient.

The following material-based values for `refIndex` and `absCoef` can be used for reference:

Material	Wavelength (in Microns)	Ref. Index	Abs. Coef.	Material	Wavelength (in Microns)	Ref. Index	Abs. Coef.
Aluminum	600	1.3	7	Nickel	855	2.59	4.55
Cobalt	600	0.2	3	Palladium	827	2.17	5.22
Copper	850	2.08	7.15	Platinum	827	2.92	5.07
Gold	600	0.3	3	Silver	600	0.2	3
Iron	886	3.12	3.87	Titanium	821	3.21	4.01
Lead	850	1.44	4.35	Vanadium	984	2.94	3.50
Molybdenum	954	2.77	3.74	Tungsten	827	3.48	2.79

Example

```
sphere setObjectTexture [0,"#fresnel(1.2, 0.8)"];
```



5.5 Modifying Unit Appearance

Many of the character models in VBS4 are configured using hidden selections, which allow easy swapping of clothing texture colors and also (in some cases) different kit items (for example, a beret can be swapped for a helmet).

This functionality is used to randomize civilian clothing and to produce woodland and desert camouflage uniforms from the same model base.

In certain situations, you may want a person to always appear in the same clothes (for example, a civilian target fitting a specific description). The following sections describe how this can be done:

- Clothing Control (below)
- UCS Control (on page 230)
- United Nations (UN) Aid Worker Vest (on page 232)
- Custom Textures (on page 233)

5.5.1 Clothing Control

This applies to models which have several selections available for random (or custom) skinning. When these models are placed in a mission, their clothing textures are chosen at random from a predefined list of textures.

You can override the random selection of clothing textures upon mission start.

Follow these steps:

1. Find what textures are available for clothing.
2. Find the selection on the model which that texture is being applied to.

3. Create a trigger that uses the setObjectTextureGlobal (https://sqf.bisimulations.com/display/SQF/setObjectTextureGlobal) command, with the following settings:

   - **Size (Left-Right):** 0
   - **Size (Up-Down):** 0
   - **Activation:** None
   - **Condition:** true
   - **On Activation:**

     ```
     _init = [] spawn {
       sleep 0.01; {
         _x setObjectTextureGlobal [0, texture_path]
       } forEach [unitA, unitB, unitC, ...]
     };
     ```

     - `texture_path` - The path to the clothing texture.
     - `unitA, unitB, unitC` - The object names of unit A, unit B, and unit C.

The clothing texture is now overridden.

Example:

The following script statement returns a nested array with the textures that are available for the different selections of a unit called `testman`:

```
getArray (
  configFile >> "cfgVehicles" >> typeof testman >>
  "hiddenSelectionsRandomTextures"
)
```

For an Afghan man, the returned result is the following five arrays:

```
[
  "\VBS2\people\afg\afg_man\data\afg_robeblack_co",
  "\VBS2\people\afg\afg_man\data\afg_robeblue_co",
  "\VBS2\people\afg\afg_man\data\afg_robebrown_co",
  "\VBS2\people\afg\afg_man\data\afg_robepink_co",
  "\VBS2\people\afg\afg_man\data\afg_robegreen_co",
  "\VBS2\people\afg\afg_man\data\afg_robewhite_co"
]
["\VBS2\people\afg\afg_man\data\afg_equip_co","","",""]
[
  "\VBS2\people\afg\afg_man\data\afg_scarfblack_co",
  "\VBS2\people\afg\afg_man\data\afg_scarfblue_co",
  "\VBS2\people\afg\afg_man\data\afg_scarfbrown_co",
  "\VBS2\people\afg\afg_man\data\afg_scarfgreen_co",
```

```
  "\VBS2\people\afg\afg_man\data\afg_scarfpink_co",
  "\VBS2\people\afg\afg_man\data\afg_scarfwhite_co",
  "","","","","","","","","","","",""
]
[""]
[""]
```

These arrays correspond to the number that is given as the first parameter in the `setObjectTextureGlobal` command, which is used to apply the clothing texture to the person.

The names of the selections can be determined with the following script command:

```
getArray (configFile >> "cfgVehicles" >> typeof testman >> "hiddenSelections")
```

Which returns:

```
[
  "swap_robe","hide_vest","hide_scarf",
  "hide_bombvest","hide_bombvest_wire","hide_teamcolor"
]
```

Example:

In the case of an Afghan man unit called `testman`, you need to create a trigger with the following On Activation script statement:

```
_init = [] spawn {
  sleep 0.01; {
   _x setObjectTextureGlobal [0, "\vbs2\people\afg\afg_man\data\*"]
  } forEach [testman]
};
```

Where * is one of the following values:

```
afg_robeblack_co
afg_robeblue_co
afg_robebrown_co
afg_robepink_co
afg_robegreen_co
afg_robewhite_co
```

And the first parameter of the `setObjectTextureGlobal` command is the index position in the selection array of the selection that you want to change. In this case, 0 equates to the `swap_robe` selection of the model.

5.5.2 UCS Control

Use the addComponentToSlot (https://sqf.bisimulations.com/display/SQF/addComponentToSlot) and removeComponent (https://sqf.bisimulations.com/display/SQF/removeComponent) script commands to modify the visual appearance of appropriately created unit models, by controlling their UCS (Universal Component System) components.

For example, SE Police units have been modeled with a set of individual pieces of equipment.

SE Police units have the following equipment:

- Pepper Spray
- Holster
- Handcuffs
- Magazine Pouch
- Radio
- Baton

To add equipment to SE Police units, use the following SQF code:

```
// Pepper Spray
obj_added ="vbs_se_police_pepperspray_blk" addComponentToSlot
  [
    player getComponents "slot_character_waistgear" select 0 select 0,
    "slot_Equipment_01",0
  ];
// Holster
obj_added = "vbs_se_police_holster_blk" addComponentToSlot
  [
    player getComponents "slot_character_waistgear" select 0 select 0,
    "slot_Equipment_02",0
  ];
// Handcuffs
obj_added = "vbs_se_police_handcuffs_blk" addComponentToSlot
  [
```

```
    player getComponents "slot_character_waistgear" select 0 select 0,
    "slot_Equipment_03",0
  ];
// Magazine Pouch
obj_added = "vbs_se_police_magpouch_blk" addComponentToSlot
  [
    player getComponents "slot_character_waistgear" select 0 select 0,
    "slot_Equipment_04",0
  ];
// Radio
obj_added = "vbs_se_police_radio_blk" addComponentToSlot
  [
    player getComponents "slot_character_waistgear" select 0 select 0,
    "slot_Equipment_05",0
  ];
// Baton
obj_added = "vbs_se_police_baton_blk" addComponentToSlot
  [
    player getComponents "slot_character_waistgear" select 0 select 0,
    "slot_Equipment_06",0
  ];
```

To remove equipment from SE Police units, use the following SQF code:

```
// Pepper Spray
removeComponent
  (
    player getComponents "slot_character_waistgear" select 0 select 0
    getComponents "slot_Equipment_01" select 0 select 0
  );
// Holster
removeComponent
  (
    player getComponents "slot_character_waistgear" select 0 select 0
    getComponents "slot_Equipment_02" select 0 select 0
  );
// Handcuffs
removeComponent
  (
    player getComponents "slot_character_waistgear" select 0 select 0
    getComponents "slot_Equipment_03" select 0 select 0
  );
// Magazine Pouch
removeComponent
  (
    player getComponents "slot_character_waistgear" select 0 select 0
```

```
    getComponents "slot_Equipment_04" select 0 select 0
  );
// Radio
removeComponent
  (
    player getComponents "slot_character_waistgear" select 0 select 0
    getComponents "slot_Equipment_05" select 0 select 0
  );
// Baton
removeComponent
  (
    player getComponents "slot_character_waistgear" select 0 select 0
    getComponents "slot_Equipment_06" select 0 select 0
  );
```

5.5.3 United Nations (UN) Aid Worker Vest

The United Nations (UN) Aid Worker Vest is available as a UCS (Universal Component System) Gear Component, which allows any UCS unit to be dressed as a UN Aid Worker.

Image-24: UCS unit with a UN vest

You can replace the current vest component of a unit with a UN one, using the following SQF code:

WARNING

The SQF code only applies to UCS characters. Most characters, introduced since VBS3 18.3, are UCS compatible.

```
vest1 = player getComponents "slot_character_vest";
removeComponent (vest1 select 0 select 0);
vestUN="vbs_vest_aidworker_01_blu"
addComponentToSlot[player,"slot_character_vest",0]
```

5.5.4 Custom Textures

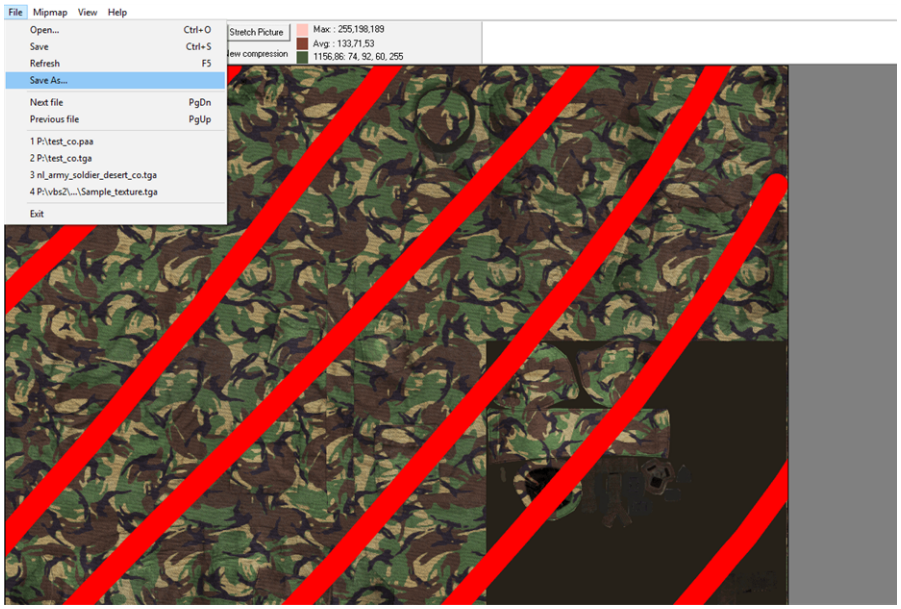
You can change custom character textures at runtime, using the setObjectTexture (https://sqf.bisimulations.com/display/SQF/setObjectTexture) script command. Each character model has various sections (represented by index numbers used in `setObjectTexture` - for example, 0 represents uniform texture) that vary from model to model.

Face textures can be changed at runtime using the setFace (https://sqf.bisimulations.com/display/SQF/setFace) script command.

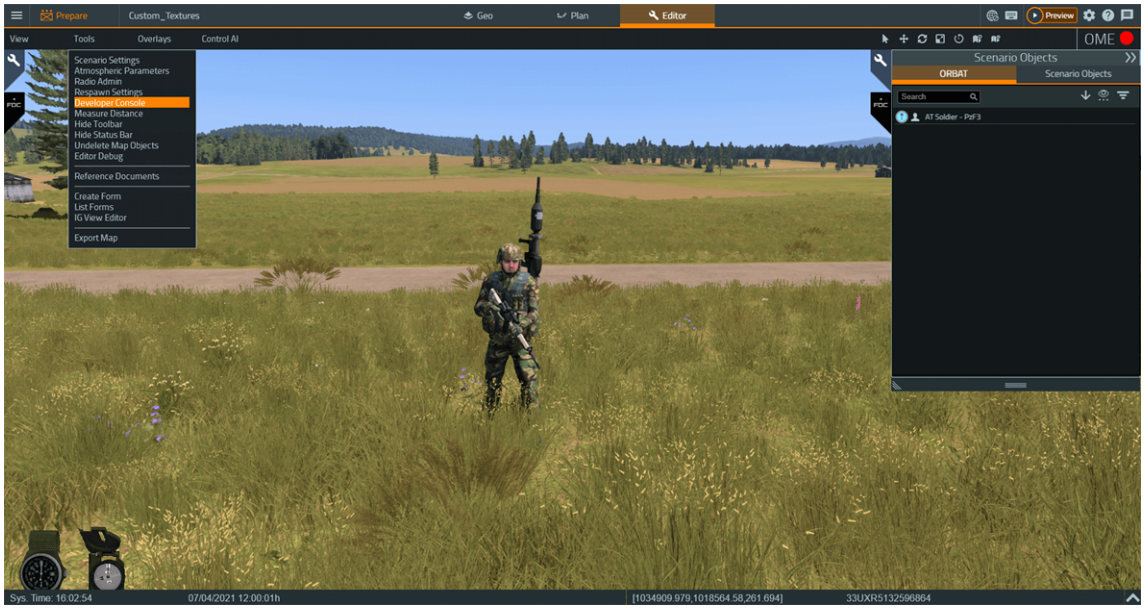
Follow these steps:

1. Make the required texture modifications in any graphic software you wish to use (for example, Photoshop).
2. Save the file as `.tga` in:

   `\VBS_Installation\myData\Blue\files\`
3. Open the `.tga` file in **TexView** (found under `P:\Tools` in the VBS Developer Suite).
4. In **TexView**, open the menu **File > Save As**.



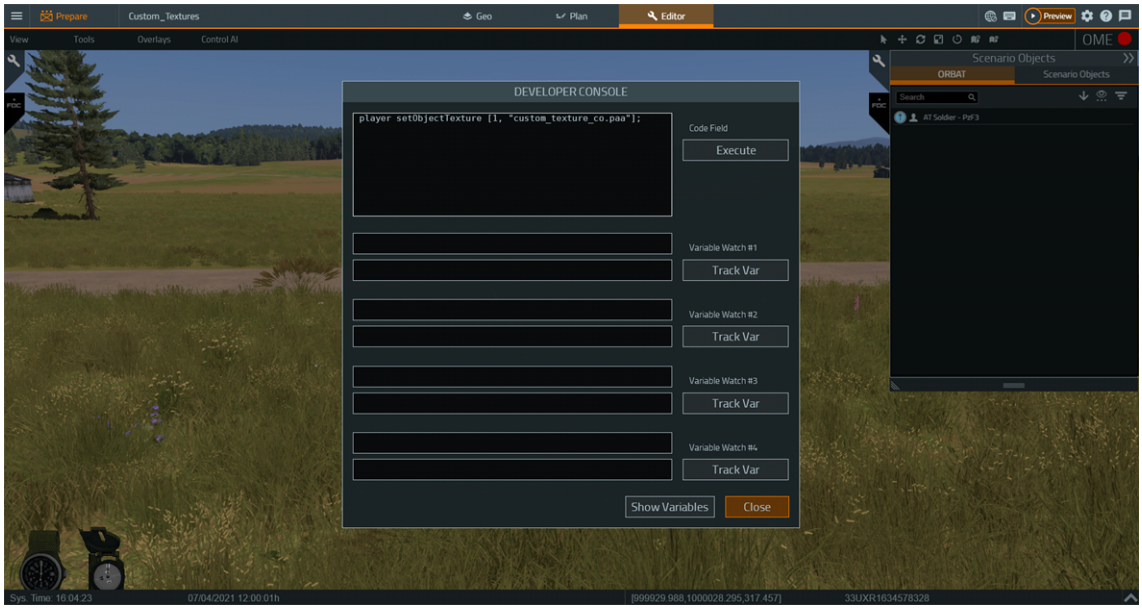
5. Select ***.paa, *.pac** in **Save as type**, enter the new `.paa` file name (for example, `custom_texture_co.paa`), and click **Save**.
6. Load the mission from step 2 in VBS4 and place a character on the map.
7. In the **Editor (Prepare Mode)**, navigate to **Tools > Developer Console**.



8. In the **Code Field**, type:

```
player setObjectTexture [1, "custom_texture_co.paa"]
```

> **NOTE**
>
> Use the `.paa` file name from step 5 (for example, `custom_texture_co.paa`).



9. Click **Execute**.

The character texture is now changed.

The following command examples change face textures.

EXAMPLE

```
UnitName setFace "vbs_face_male_kp_civilian_medium_02";
UnitName setFace "vbs_face_male_kp_civilian_medium_03";
```

You can use a custom face texture from outside of VBS4.

```
UnitName setFace [
  "vbs_face_male_kp_civilian_medium_01",
  ((getDirectory 1) + "myDemoFaceTexture_co.paa")
]
```

NOTE

Change `UnitName` to the name of the unit whose face texture you want to change.

5.6 Post-Processing Effects

You can use Post-Processing Effects (PPEs), which are video filters that allow you to add special effects (for example, to simulate camera and / or optic views, vision impairment, among others).

The following SQF commands can be used to control PPEs:

- camPpEffectCreate (https://sqf.bisimulations.com/display/SQF/camPpEffectCreate)

  Creates the given PPE for the given camera.

- ppEffectAdjust (https://sqf.bisimulations.com/display/SQF/ppEffectAdjust)

  Modifies the given PPEs, so that they can be applied with ppEffectCommit (https://sqf.bisimulations.com/display/SQF/ppEffectCommit).

- ppEffectCommit (https://sqf.bisimulations.com/display/SQF/ppEffectCommit)

  Commits the given PPE modifications, done by ppEffectAdjust (https://sqf.bisimulations.com/display/SQF/ppEffectAdjust).

- ppEffectCommitted (https://sqf.bisimulations.com/display/SQF/ppEffectCommitted)

  Check whether the PPE modifications commit has finished.

- ppEffectCreate (https://sqf.bisimulations.com/display/SQF/ppEffectCreate)

  Creates the given PPE for the simulation.

- ppEffectDestroy (https://sqf.bisimulations.com/display/SQF/ppEffectDestroy)

  Destroys the given simulation PPE, created using ppEffectCreate (https://sqf.bisimulations.com/display/SQF/ppEffectCreate).

- ppEffectEnable (https://sqf.bisimulations.com/display/SQF/ppEffectEnable)

  Enables / disables the simulation PPE, created using ppEffectCreate (https://sqf.bisimulations.com/display/SQF/ppEffectCreate).

- ppEffectLocation (https://sqf.bisimulations.com/display/SQF/ppEffectLocation)

  Assigns / updates the given PPE at a given location on the map.

- ppEffectParams (https://sqf.bisimulations.com/display/SQF/ppEffectParams)

  Returns the PPE data for the given location, assigned to it using ppEffectLocation (https://sqf.bisimulations.com/display/SQF/ppEffectLocation).

The following PPEs are supported in VBS4:

- Color Correction (on the next page)
- Digital Zoom (on page 238)
- Film Grain (on page 239)

- Dynamic Blur (on page 240)
- Combining PPEs (on page 241)

5.6.1 Color Correction

Adds color correction to the camera view.

The color correction specifications consist of the blend color and of the colorization itself.

Parameter	Type	Description
`blendColor`	[R, G, B, A]	Blend color.
`colorizeColor`	[R, G, B, A]	Colorization color.

EXAMPLE

```
"ColorCorrections" ppEffectEnable true;
"ColorCorrections" ppEffectAdjust [[1,1,0,0.5], [10,1,1,0.5]];
"ColorCorrections" ppEffectCommit 0;
```

5.6.2 Digital Zoom

Adds digital zoom to the camera view.

You can specify the zoom level and the optional pixelation.

> **NOTE**
> The zoom level is a positive number in the range [1, 100].

Image-25: Zoom level only

Parameter	Type	Description
`zoom`	Number	Zoom / magnification level.
`pixelation`	Number	Pixelation level (optional).

Zoom level only:

```
"DigitalZoom" ppEffectEnable true;
"DigitalZoom" ppEffectAdjust [10];
"DigitalZoom" ppEffectCommit 0;
```

Zoom level and pixelation:

```
"DigitalZoom" ppEffectEnable true;
"DigitalZoom" ppEffectAdjust [3,4];
"DigitalZoom" ppEffectCommit 0;
```

5.6.3 Film Grain

Adds film grain to the camera view.

The film grain specifications consist of intensity, grain size, and whether the noise is monochromatic or not.

> **NOTE**
>
> Film grain is currently unsupported in Night Vision (NV) and Thermal Imaging (TI) modes in VBS4.

Parameter	Type	Description
`intensity`	Number	Film grain intensity.
`grainSize`	Number	Film grain size.
`isMonochrome`	Boolean	Whether the film grain is monochromatic (`true`) or not (`false`).

> **EXAMPLE**

```
"FilmGrain" ppEffectEnable true;
"FilmGrain" ppEffectAdjust [1, 3, false];
"FilmGrain" ppEffectCommit 0;
```

5.6.4 Dynamic Blur

Adds dynamic blur to the camera view.

Parameter	Type	Description
`blur`	Number	Blurriness level.

EXAMPLE

```
"DynamicBlur" ppEffectEnable true;
"DynamicBlur" ppEffectAdjust [2];
"DynamicBlur" ppEffectCommit 0;
```

5.6.5 Combining PPEs

You can combine several PPEs in a camera view.

Image-26: Color Correction and Dynamic Blur

> **NOTE**
>
> Combining Film Grain (on page 239) and Digital Zoom (on page 238) is currently unsupported in VBS4.

5.7 Scripted Damage Control

Administrators and mission designers can apply damage to vehicle components before or during a mission in the Prepare / Execute Mode, using scripts added in the Editor.

> **NOTE**
>
> Scripted vehicle component damage control requires specifically configured vehicles. For more information, see Complex Vehicle Damage in the VBS Developer Reference.

Scripts used to add damage to components of a vehicle have the following syntax:

```
[vehicle,"part_class_name"] call scripted_damage_function_name
```

Parameters inside the square brackets define the affected vehicle and the component to damage. These are followed by Scripted Damage Functions (on the next page).

Use the Editor to apply damage as described in Damage Implementation (on page 245).

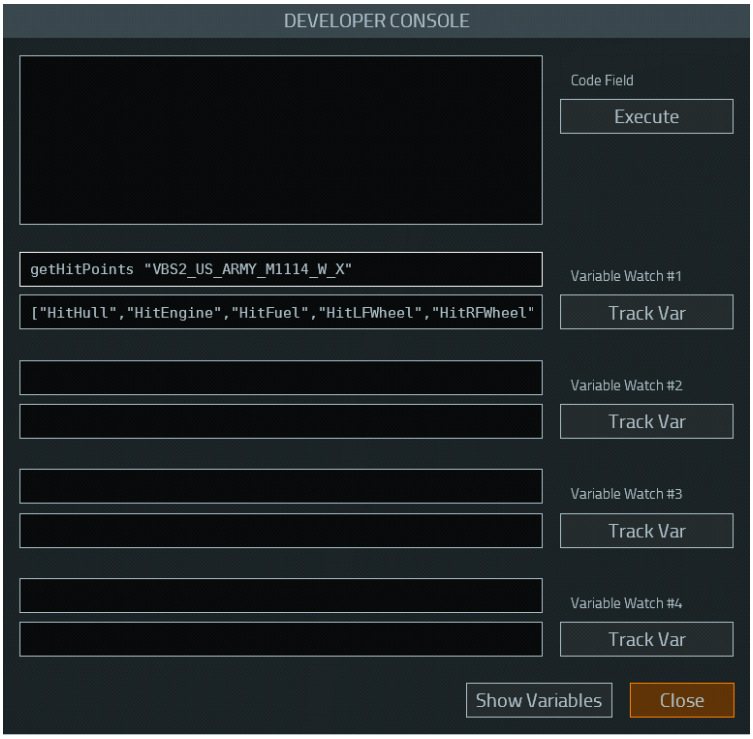
Many vehicles include the following set of default components, that enable scripted damage if the vehicle and components are properly configured:

Component	Parameter	Description
Engine	HitEngineProxy	Engine loses power.
Fuel Tank	HitFuel	Fuel tank is damaged (fuel capacity is reduced to 5%). The vehicle may switch to a reserve tank if configured.
Gearbox	HitGearbox	The gearbox is disabled.
Vehicle Weapons	Name of the weapon class in the vehicle turret configuration.	The weapon is individually damaged instead of affecting all vehicle weapons.

You can find the vehicle components that can be damaged by using the getHitPoints (https://sqf.bisimulations.com/display/SQF/getHitPoints) command in the following way:

```
getHitPoints "vehicle_class_name"
```

Image-27: Developer Console with damageable vehicle components



5.7.0.1 Scripted Damage Functions

Use the following functions to control vehicle and component damage:

- fn_vbs_damageHitpart (below)
- fn_vbs_repairHitpart (on the next page)
- fn_vbs_allowDamage (on the next page)

5.7.0.1.1 fn_vbs_damageHitpart

Allows the administrator to assign predetermined damage using parameters and error codes, to be applied when the vehicle receives damage from any VBS4 entity during a mission.

EXAMPLE

```
[vehicle, "hitEngineProxy"] call fn_vbs_damageHitpart
[vehicle, "701"] call fn_vbs_damageHitpart
```

5.7.0.1.2 fn_vbs_repairHitpart

Allows the administrator to reverse damage caused by `fn_vbs_damageHitpart`.

The example command is used to repair damage to the stabilization and engine simulations.

EXAMPLE

```
[vehicle, "hitEngineProxy"] call fn_vbs_repairHitpart
```

5.7.0.1.3 fn_vbs_allowDamage

Enables the administrator to turn vehicle damage on and off.

- Set to `false` to make a vehicle invulnerable and prevent the crew from being harmed in a collision, or suffering side-effects from shell shock.
- Set to `true` to enable damage.

EXAMPLE

```
[vehicle, false] call fn_vbs_allowDamage
```

Vehicle Vulnerability

Use the `fn_vbs_allowDamage` function to make a vehicle invulnerable. The syntax is as follows:

```
[vehicle, boolean] call fn_vbs_allowDamage
```

Once executed, either of the following commands prevent the vehicle (`VEH01`) from taking damage:

```
[VEH01, false] call fn_vbs_allowDamage
[VEH01] call fn_vbs_allowDamage
```

To return the vehicle back to its default state, change the boolean value to `true`:

```
[VEH01, true] call fn_vbs_allowDamage
```

setHit Command

The setHit (https://sqf.bisimulations.com/display/SQF/setHit) script command is available to set the damage level of a specific component with the following syntax:

```
vehicle setHit [part, damage]
```

Where *vehicle* is the name of the vehicle, *part* is the component name, and *damage* is a number between 0 and 1.

In order to register a component as damaged, set the damage to 0.5 or higher.

> **NOTE**
> Some components may have a different damage threshold.

Vehicle Repair

When a vehicle is destroyed, the component damage functions do not repair it. Do one of the following:

- Use the setDamage (https://sqf.bisimulations.com/display/SQF/setDamage) command with the following syntax: `vehicle setDamage 0`
- In the Editor (Execute Mode), right-click the vehicle, and select **Repair Vehicle**.

5.7.0.2 Damage Implementation

Vehicle damage scripts can be added to the following:

- **Vehicle Editor Object**
  - Editor (Prepare Mode) - Damage is applied on mission startup.
  - Editor (Execute Mode) - Damage is applied during the mission.
- **Triggers** - Damage is delayed until the trigger is activated.
- **Waypoints** - Damage is applied when the vehicle reaches a specific waypoint.
- **Developer Console** - Damage is applied during a mission on demand.

Follow these steps:

1. Open the relevant **Object Properties** dialog or the **Developer Console**.
2. Enter the damage script:
   - Vehicle Editor Object - In the **Initialization Statement** field.
   - Trigger - In the **On Activation** field.
   - Waypoint - In the **On Activation** field.
   - Developer Console - In the **Code** field.
3. Adjust any other properties as required, and click **OK**.

Damage is applied when the script entered in the Object Properties dialog is executed.

5.8 Custom Command Menus

You can create custom command menus (see Commanding Subordinates in the VBS4 Trainee Manual), using SQF scripts.

For information on how to define custom command menus in addon configuration, see Custom Command Menus in the VBS Developer Reference.

A custom menu is defined using a global array, which contains the definitions for the menu entries. It can have multiple sub-levels, and be changed dynamically by reconfiguring the values in the array.

Each menu (the main one, as well as each of the possible sub-menus) must be defined in separate arrays. The naming is not limited, but in order to avoid possible conflicts with other global variables, they should start with a unique identifier / prefix (for example, `"MYMENU_menu1"`).

To display the custom menu, use the showCommandingMenu (https://sqf.bisimulations.com/display/SQF/showCommandingMenu) command:

```
showCommandingMenu "#USER:MY_MENU_communication"
```

To add the custom menu to the standard command menu, assign its definition array to `BIS_MENU_GroupCommunication`, which makes it available under the **Communications** entry in the regular command menu (press **0 - 8** to access it):

```
BIS_MENU_GroupCommunication = +MY_MENU_communication
```

Menu Syntax

The menu syntax has the following array structure:

First array element:

```
[title, standard_cursor]
```

Parameter	Type	Description
`title`	String	Label to display at the bottom of the menu (should not be empty, otherwise just a black block is displayed).
`standard_cursor`	Boolean	If set to true, then custom icons (defined in the menu entries) are not displayed.

Second and higher array elements:

> **NOTE**
>
> Up to 10 menu elements can fit the menu frame (additional elements do not fit the menu frame).

```
[label, key, submenu, elementType, script, visible, active, icon]
```

Parameter	Type	Description
`label`	String	Menu selection caption.
`key`	Array	Key-press to activate the selection: • 0: No key • 1: **Esc** • 2 - 9: **F1** - **F10** • 10,11,12, ...: Numbers **9** and **10**, **-**, and the rest of the keyboard (corresponding to DIK assignments). More than one key can be assigned, but only the first is displayed in the menu.
`submenu`	String	User menu name string (for example, `"#USER:MY_SUBMENU_NAME"`), use "" for a script to execute. Must be global, and preceded by `"#USER:"`.
`elementType`	Number	Menu control element: • -1: Separator line (any defined title text is ignored). • -2: No action (used for group headers). • -3: Hides the command menu. • -4: Goes up one level in the menu. • -5: Executes the defined script.
`script`	Array	Always defined in the following way: `[["expression", command]]` The command to execute on activation (if `type` is 5 and `submenu` is "").
`visible`	Number	Defines menu entry visibility (0 means hidden). Can use system variables (see Conditions (below)).
`active`	Number	Defines menu entry highlighting (0 means grayed out). Can use system variables (see Conditions (below)).
`icon`	String	Path to a custom cursor, which is displayed when the selection is active (optional).

Conditions

Conditions are strings that control whether a menu entry can be seen or selected, and can contain one or more keywords.

Each keyword is determined by the engine, and automatically assigned a value of either 1 (true) or 0 (false).

It is not possible to use regular script commands in the condition string.

Keywords can be negated by subtracting them from 1 (for example, `"1-CursorOnGround"` is only true, if the cursor points at the sky), and combined by multiplying them (for example, `"CursorOnGround * IsCommander"`).

Condition String	Description
`"AmmoLow"`	Player ammunition is low (last magazine).
`"AreActions"`	Menu entry is an action (as opposed to a sub-menu, for example).
`"CanAnswer"`	Is able to reply to radio messages.
`"CommandsToGunner"`	Can give orders to gunner.
`"CommandsToPilot"`	Can give orders to driver / pilot.
`"CursorOnEmptyVehicle"`	Cursor points at an empty vehicle.
`"CursorOnEnemy"`	Cursor points at an identified enemy.
`"CursorOnFriendly"`	Cursor points at a unit of the same side.
`"CursorOnGround"`	Cursor points at the ground.
`"CursorOnGroupMember"`	Cursor points at a group member.
`"CursorOnGroupMemberSelected"`	Cursor points at the selected group member.
`"CursorOnHoldingFire"`	Cursor points at the group member that has been ordered to hold fire.
`"CursorOnNeedFirstAID"`	A non-medic unit is selected, and the cursor points at an unconscious unit.
`"CursorOnNeedHeal"`	A medic is selected, and the cursor points at a heavily injured (damage more than or equal to 0.45) or unconscious unit.
`"CursorOnNotEmptySubgroups"`	Unit under cursor is in a different subgroup than the player.
`"CursorOnVehicleCanGetIn"`	At least one of the currently selected units is not in a vehicle under the cursor.
`"FormationDiamond"`	Group formation is "DELTA" (diamond).
`"FormationLine"`	Group formation is "LINE".
`"FuelLow"`	Player vehicle is low on fuel.
`"HasRadio"`	Player has radio actions available.
`"IsAlone"`	No group members, or group members confirmed dead.

Condition String	Description
"IsAloneInVehicle"	Only one unit in a vehicle.
"IsCommander"	Vehicle commander (or driver, if the vehicle has no turrets). If not in vehicle, then true for group leader.
"IsLeader"	Group leader (group must have members).
"IsSelectedToAdd"	Left or right control key is pressed.
"IsTeamSwitch"	Team switching is enabled.
"Multiplayer"	Multiplayer session in progress.
"NotEmpty"	Group members are selected.
"NotEmptyBlueTeam"	Group members are assigned to team Blue.
"NotEmptyCommanders"	Unit commanding a vehicle is currently selected.
"NotEmptyGreenTeam"	Group members are assigned to team Green.
"NotEmptyInVehicle"	Group members in a vehicle are selected.
"NotEmptyMainTeam"	Player commands subordinates.
"NotEmptyRedTeam"	Group members are assigned to team Red.
"NotEmptyYellowTeam"	Group members are assigned to team Yellow.
"NotEmptySoldiers"	At least one of the selected units is not in a vehicle (unit type does not matter - can be soldier, civilian, or even animals).
"NotEmptySubgroups"	At least one of the selected units is in a different subgroup than the main subgroup of the group (for example, in a different vehicle than the player).
"PlayableLeader"	Leader of the player group is a playable unit.
"PlayableSelected"	Only one unit is selected and it is playable.
"PlayerOwnRadio"	Player has radio actions available.
"PlayerVehicleCanGetIn"	Player vehicle has some free seats (even if locked).
"SomeSelectedHaveTarget"	At least one of the group members has a target assigned.
"SomeSelectedHoldingFire"	At least one of the group members is ordered to hold fire.
"VehicleCommander"	Vehicle commander (or driver, if the vehicle has no turrets). Only true, if in a vehicle.

For the custom menu shown in the upper part of the menu (**Removal options**), the conditions are:

- visibility: `"CursorOnGroupMember"` (only shows, when the cursor is on the group member).
- selection: `"CursorOnGroupMember"` (can only be selected, when the cursor is on the group member).

For the lower part (**Placement options**), the conditions are:

- visibility: `"1"` (always visible).
- selection: `"CursorOnGround * 1-CursorOnGroupMember"` (can only be selected, when the cursor points to the ground, and not on a group member).

The results are as follows (images from left to right):

- **1st image:** The cursor is on the ground, but does not point at a group member, so the option to place a unit at that position is visible and can be selected.
- **2nd image:** Since the cursor points at the group member, `"CursorOnGroupMember"` returns 1, and makes the condition false. Therefore, the option to create a unit at that location is disabled (even though it is still visible).
- **3rd image:** The cursor points at the sky (`"CursorOnGround"` returns 0), so the removal options are hidden, and the placement options are disabled.

The SQF script is as follows (for example, it can be placed inside `init.sqf` - see Init.sqf (on page 37), or a separate SQF file)

```
/* Only show placement option if the cursor is on the ground,
    but not pointing at a unit or vehicle
 */
_placementCondition =
"CursorOnGround * 1-CursorOnGroupMember * 1-CursorOnEmptyVehicle";

/* Main menu (contains some top-level actions ("Removal options")
   and calls to sub-menus
*/
MY_MENU_communication =
[
  ["Custom Menu",true],
  // header text for removal options
  ["Removal options", [0], "", -2, [["expression", ""]], "1", "1"],
  // Option is always shown
  [
    " Remove all", [2], "", -5,
    [["expression", "['remove','all'] execVM 'place.sqf'"]],
    "1", "1"
  ],
  // Only show option if cursor is on a group member
  [
```

```
    " Remove unit", [3], "", -5,
    [["expression", "['remove','unit'] execVM 'place.sqf'"]],
    "CursorOnGroupMember","CursorOnGroupMember"
  ],
  // Only show option if cursor is on an empty vehicle
  [
    " Remove vehicle", [3], "", -5,
    [["expression", "['remove','vehicle'] execVM 'place.sqf'"]],
    "CursorOnEmptyVehicle", "CursorOnEmptyVehicle"
  ],
  // Separator line
  ["(separator)", [3], "", -1, [["expression", ""]], "1", "1"],
  // Header text for placement options
  ["Placement options", [0], "", -2, [["expression", ""]], "1", "1"],
  /* Option is always shown, but can only be selected if cursor
     condition is fulfilled
  */
  [
    " Place units", [4], "#USER:MY_MENU_placeUnit", -5,
    [["expression", ""]], "1", _placementCondition
  ],
  [
    " Place vehicles", [5], "#USER:MY_MENU_placeVehicle", -5,
    [["expression", ""]], "1", _placementCondition
  ]
];

// Sub-menu to create units
MY_MENU_placeUnit = [
  ["Unit placement",true],
  ["Type of unit to place", [0], "", -2, [["expression", ""]], "1", "1"],
  [
    " Unarmed", [2], "", -5,
    [
      [
        "expression",
        "['place','unit','VBS2_US_ARMY_Interpreter_W'] execVM 'place.sqf'"
      ]
    ],
    "1", _placementCondition],
  [
    " Machinegun", [3], "", -5,
    [
      [
        "expression",
        "['place','unit','VBS2_US_ARMY_MGunner_W_M249_none'] execVM 'place.sqf'"
      ]
```

```
    ],
    "1", _placementCondition],
  [
    " Anti-Tank", [4], "", -5,
    [
      [
        "expression",
        "['place','unit','VBS2_us_army_ATsoldier_W_Javelin_none'] execVM
'place.sqf'"
      ]
    ],
    "1", _placementCondition]
];

// Sub-menu to create vehicles
MY_MENU_placeVehicle = [
  ["Vehicle placement", true],
  ["Type of veh. to place", [0], "", -2, [["expression", ""]], "1", "1"],
  [
    " Land Rover", [2], "", -5,
    [
      [
        "expression",
        "['place','vehicle','VBS2_AU_Army_Landrover_D_X'] execVM 'place.sqf'"
      ]
    ],
    "1", _placementCondition
  ],
  [
    " Humvee", [3], "", -5,
    [
      [
        "expression",
        "['place','vehicle','VBS2_US_ARMY_M1114_D_X'] execVM 'place.sqf'"
      ]
    ],
    "1", _placementCondition
  ],
  [
    " Abrams", [4], "", -5,
    [
      [
        "expression",
        "['place','vehicle','VBS2_US_ARMY_M1A1_D_X'] execVM 'place.sqf'"
      ]
    ],
    "1", _placementCondition
```

```
  ]
];

// Open the custom menu
showCommandingMenu "#USER:MY_MENU_communication";
```

5.8.1 Custom Command Menu Example

Example of a custom menu structure in `init.sqf`:

```
player addAction ["Open custom menu","placeMenu.sqf"];

// Only show placement option if the cursor is on the ground, but not pointing at a
unit or vehicle
_placementCondition = "CursorOnGround * 1-CursorOnGroupMember * 1-
CursorOnEmptyVehicle";

MY_MENU_communication =
  [
    [
      "Personal menu", true
    ],
    // Header text for removal options
    [
      "Removal options", [0], "", -2,
      [["expression", ""]], "1", "1"
    ],
    // Option is always shown
    [
      " Remove all", [2], "", -5,
      [["expression", "['remove','all'] execVM 'place.sqf'"]], "1", "1"],
    // Only show option if cursor is on a group member
    [
      " Remove unit", [3], "", -5,
      [["expression", "['remove','unit'] execVM 'place.sqf'"]],
      "CursorOnGroupMember", "CursorOnGroupMember"
    ],
    // Only show option if cursor is on an empty vehicle
    [
      " Remove vehicle", [3], "", -5,
      [["expression", "['remove','vehicle'] execVM 'place.sqf'"]],
      "CursorOnEmptyVehicle", "CursorOnEmptyVehicle"
    ],
    // Separator line
    [
      "(separator)", [3], "", -1,
```

```
      [["expression", ""]], "1", "1"
    ],
    // Header text for placement options
    [
      "Placement options", [0], "", -2,
      [["expression", ""]], "1", "1"
    ],
    // Option is always shown, but can only be selected if cursor condition is
fulfilled
    [
      " Place units", [4], "#USER:MY_MENU_placeUnit", -5,
      [["expression", ""]], "1", _placementCondition],
      [" Place vehicles", [5], "#USER:MY_MENU_placeVehicle", -5,
      [["expression", ""]], "1", _placementCondition
    ]
  ];

// Sub-menus
MY_MENU_placeUnit =
  [
    [
      "Unit placement", true
    ],
    [
      "Type of unit to place", [0], "", -2,
      [["expression", ""]], "1", "1"
    ],
    [
      " Unarmed", [2], "", -5,
      [["expression", "['place','unit','VBS2_US_ARMY_Interpreter_W'] execVM
'place.sqf'"]],
      "1", _placementCondition
    ],
    [
      " Machinegun", [3], "", -5,
      [["expression", "['place','unit','VBS2_US_ARMY_MGunner_W_M249_none'] execVM
'place.sqf'"]],
      "1", _placementCondition
    ],
    [
      " Anti-Tank", [4], "", -5,
      [["expression", "['place','unit','VBS2_us_army_ATsoldier_W_Javelin_none']
execVM 'place.sqf'"]],
      "1", _placementCondition
    ]
  ];
```

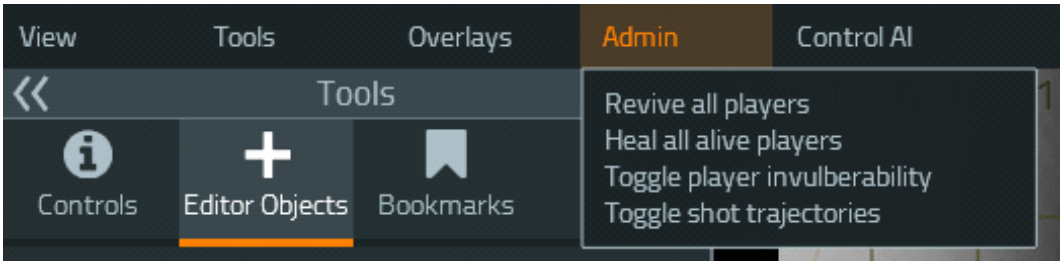
```
MY_MENU_placeVehicle =
  [
    [
      "Vehicle placement", true],
    [
      "Type of veh. to place", [0], "", -2,
      [["expression", ""]], "1", "1"
    ],
    [
      " Land Rover", [2], "", -5,
      [["expression", "['place','vehicle','VBS2_AU_Army_Landrover_D_X'] execVM
'place.sqf'"]],
      "1", _placementCondition
    ],
    [
      " Humvee", [3], "", -5,
      [["expression", "['place','vehicle','VBS2_US_ARMY_M1114_D_X'] execVM
'place.sqf'"]],
      "1", _placementCondition], [" Abrams", [4], "", -5,
      [["expression", "['place','vehicle','VBS2_US_ARMY_M1A1_D_X'] execVM
'place.sqf'"]],
      "1", _placementCondition
    ],
    [
      " Abrams", [4], "", -5,
      [["expression", "['place','vehicle','VBS2_US_ARMY_M1A1_D_X'] execVM
'place.sqf'"]],
      "1", _placementCondition
    ]
  ];


// Add custom menu to regular command menu (under "Communications")
BIS_MENU_GroupCommunication =+ MY_MENU_communication;

titleText ["Either use the user action to display the custom menu by itself,\nor
call the standard menu by pressing '0', and then selecting '8' for the custom
entries.","plain down"];
```

5.9 Administrator Tools

As a VBS4 Administrator, you can use the optional SQF-based Administrator Tools addon to facilitate scenario control.

The main intent is for administrators with SQF scripting knowledge to be able to introduce "hooks" to run scripts on different events, which allows administrators to customize these scripts to support various training needs. Without the Administrator Tools addon, there is no easy way to run a script on scenario start, scenario save, and so on, regardless of the scenario type.

Image-28: Administrator Tools in the Editor Menu



The optional Administrator Tools are found in the following ZBO file:

`\VBS_Installation\optional\admin_tools\admin_tools.zbo`

To use the Administrator Tools, several files need to be copied.

Follow these steps:

1. Copy the ZBO file (`admin_tools.zbo`) from:

   `\VBS_Installation\optional\admin_tools\`

   To:

   `\VBS_Installation\myData\Blue\content\`

2. Copy the following sub-folder:

   `\VBS_Installation\optional\admin_tools\admin_tools\`

   To the root of your VBS4 installation folder, so that you get:

   `\VBS_Installation\admin_tools\`

The Admin Tools are now ready to use in VBS4.

The Administrator Tools provide the following functionality, using SQF:

- `\VBS_Installation\optional\admin_tools\admin_tools\onEditorInit.sqf`

  Adds the **Admin** menu to the Editor Menu of the VBS Editor, when you open it in Prepare mode.

  By default, the following options are added to the Admin menu:

  - **Delete All Objects** - Deletes all the Editor Objects in the scenario.
  - **Set All to Playable** - Sets all units / vehicles in the scenario to playable.

- `\VBS_Installation\optional\admin_tools\admin_tools\onRTEInit.sqf`

  Adds the **Admin** menu to the Editor Menu of the VBS Editor, when you open it in Preview / Execute mode.

  By default, the following options are added to the Admin menu:

  - **Revive All Players** - Revives all player units in the scenario.
  - **Heal All Alive Players** - Heals any wounded players in the scenario.
  - **Toggle Player Invulnerability** - Toggles the invulnerability for all player units in the scenario on / off.
  - **Toggle Shot Trajectories** - Toggles the visualization of shot trajectories on / off.

- `\VBS_Installation\optional\admin_tools\admin_tools\onScenarioStart.sqf`

  Any SQF code you want to execute when the scenario starts in Preview / Execute mode.

- `\VBS_Installation\optional\admin_tools\admin_tools\onScenarioSave.sqf`

  Any SQF code you want to execute when the scenario is saved.

6. Scripting Reference

The scripting reference provides a reference to all the script commands and functions mention in this manual. The reference is divided into 3 parts:

- Scripting Cheat Sheet (on the next page) - Common scripting rules.
- Common Command Categories (on page 272) - Frequently used commands in VBS4.
- VBS Gateway Script Commands (on page 274) - Commands used for interoperable simulation using VBS Gateway.
- Waypoint Functions and Parameters (on page 276) - SQF functions used for Control AI waypoints.

  WARNING

  These SQF functions and their parameters are experimental and subject to change in future releases of VBS4.

- File Operations (on page 284) - A VBS4 plugin that allows various file operations.

VBS Scripting Reference

The VBS Scripting Reference is a Wiki that contains a comprehensive list of all the scripting commands and functions from the VBS4 Function Library. It contains the most current information. A search is available to help users to find commands and functions.

https://sqf.bisimulations.com/display/SQF/

Wiki.zip

The `Wiki.zip` is an offline version of the VBS Scripting Reference. The file can be found in:

`\VBS_Installation\docs\`.

Extract the ZIP file and open `SQF_Reference.html` in a browser.

Content Listing

Object, People, Vehicle, Weapon class names, and other details can be found in:

`VBS4_Content.chm`, in: `\VBS_Installation\docs\`

Allconfig.cpp

The `exportcpp` command line switch generates a full listing of the `config.cpp` used by VBS4. This provides the configuration information for every item in the simulation. Use a good text editor to access the file, Microsoft Notepad does not work due to the size of the file.

For information on generating `Allconfig.cpp`, see ExportCPP in the VBS Developer Reference.

6.1 Scripting Cheat Sheet

This section contains common scripting rules.

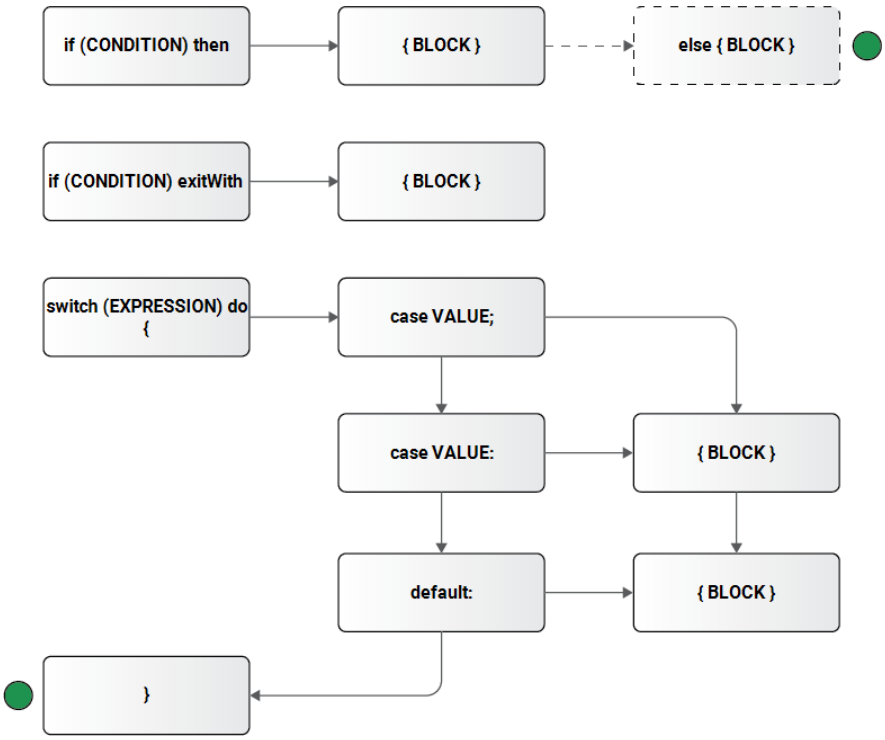
6.1.1 Operator Precedence

The scripting operators have the following evaluation precedence from highest to lowest priority:

Priority	Operators
1	|| or
2	&& and
3	== != > < >= <= >>
4	All other binary operators.
5	else
6	+ - max min
7	* / % mod atan2
8	^
9	All unary operators.

6.1.2 Conditional Control Structure

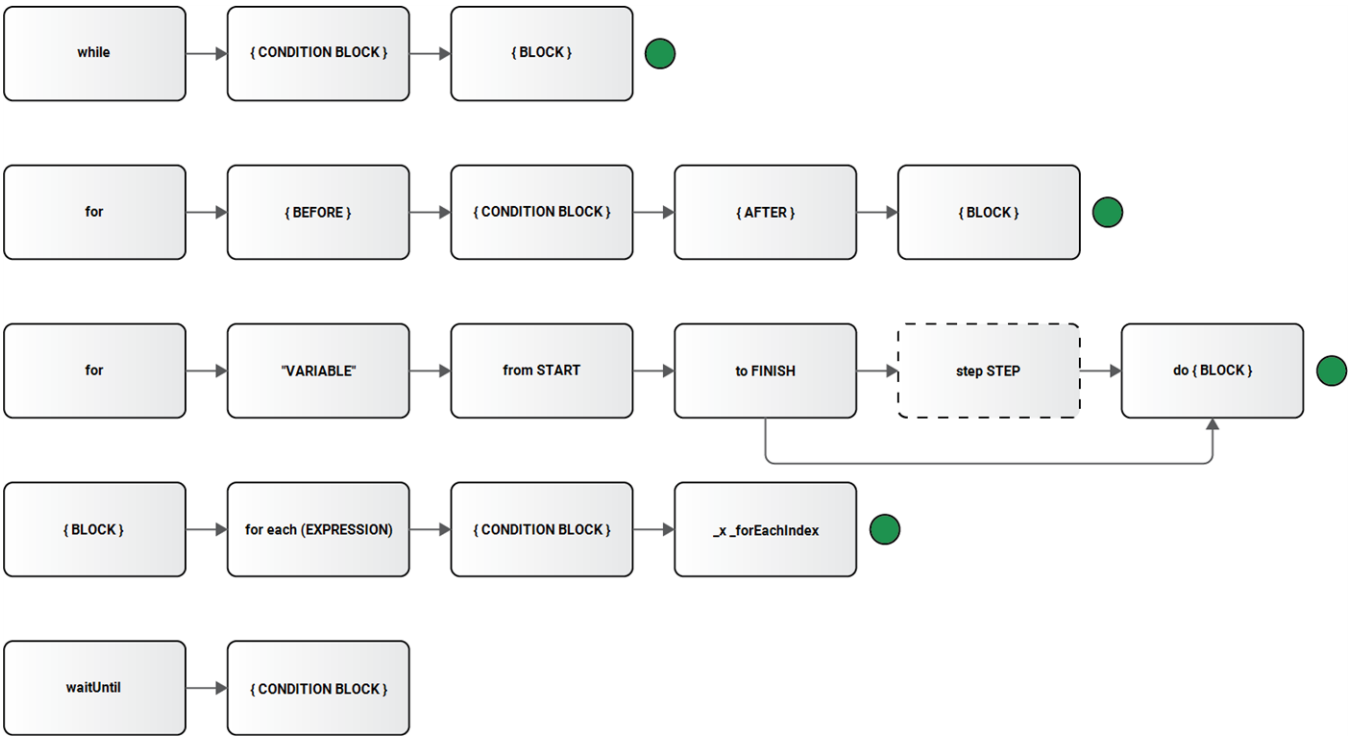
The conditional control structures workflow is:



- CONDITION is a condition Boolean statement and BLOCK is a block of code.
- The green circle indicates that the selected BLOCK value is returned (nil is returned if no BLOCK is executed).

6.1.3 Iterative (Loop) Control Structures

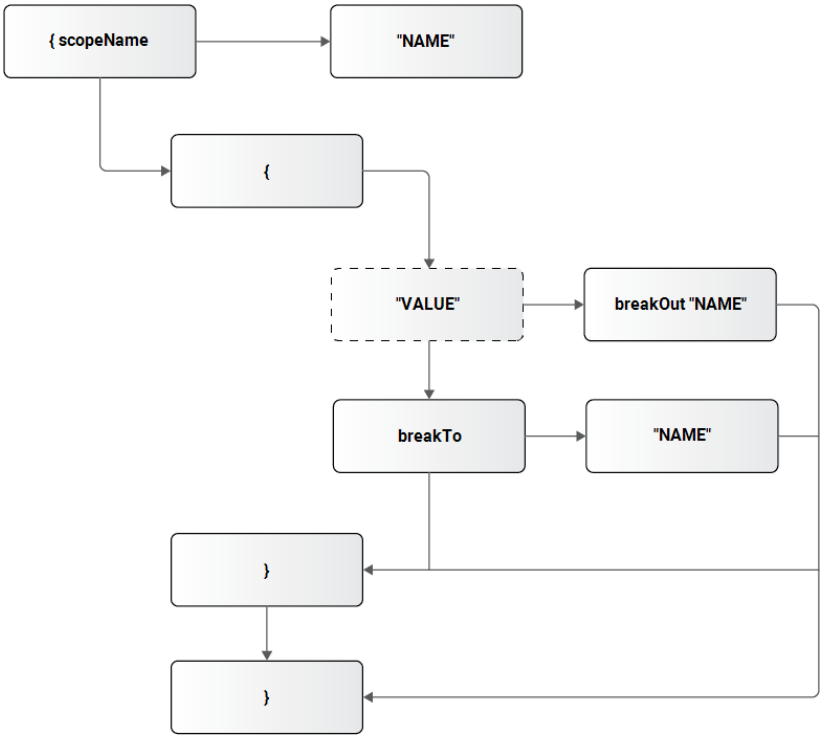
The iterative (loop) control structure workflow is:



- CONDITION BLOCK is a block of code evaluated as a condition.
- BLOCK is a block of code.
- BEFORE and AFTER are code blocks that execute before the for-Loop starts and after it finishes.
- VARIABLE is the variable that changes its value with each loop iteration.
- START is the VARIABLE value at the start of the for-Loop.
- FINISH is the VARIABLE value at the end of the for-Loop.
- STEP is the value by which the VARIABLE increases / decreases at the end of the for-Loop.
- The green circle indicates that the final BLOCK value is returned (nil is returned if no BLOCK is executed).
- A dotted line or dotted rectangle indicates that the part is optional.

6.1.4 Break Scopes

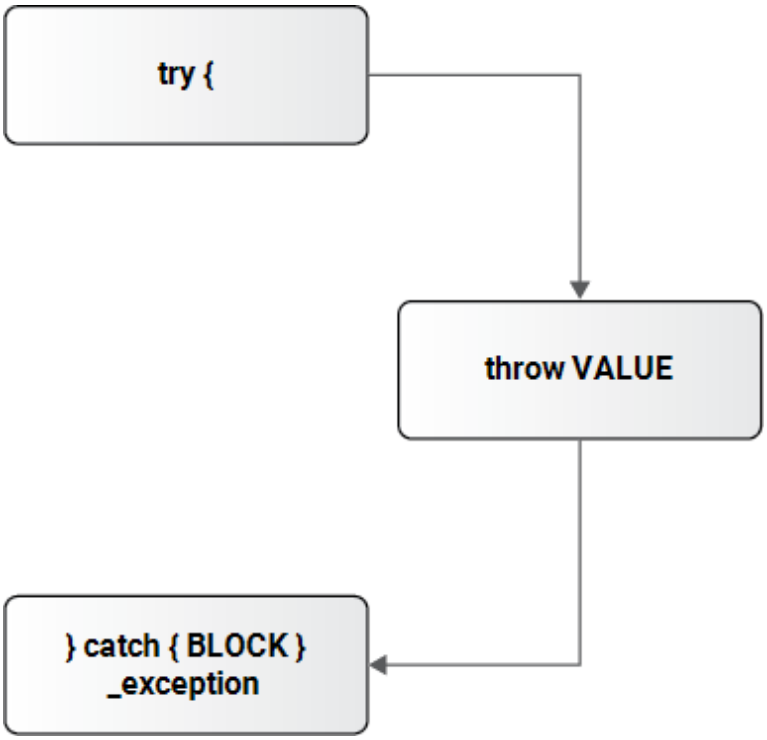
The relationship between code scopes and break statements is illustrated in the following workflow:



- NAME is the name of the code scope.
- A dotted rectangle indicates that the part is optional.

6.1.5 Exceptions

Exception handling has the following workflow:



- If no exception is thrown, the value of the try block is returned. Otherwise, the value of the catch block is returned.

6.1.6 Locality

The following commands are used to determine the locality of an object in multiplayer:

Command	Description	Syntax
isMultiplayer (https://sqf.bisimulations.com/display/SQF/isMultiplayer)	Checks if the machine is running a multiplayer session.	`isMultiplayer`
isServer (https://sqf.bisimulations.com/display/SQF/isServer)	Checks if the local machine is the multiplayer server.	`isServer`
local (https://sqf.bisimulations.com/display/SQF/local_(Command))	Checks if the object is local to the machine the command is executed on.	`local object`

6.1.7 Types

Variable types use the following common commands:

Command	Description	Syntax
nil (https://sqf.bisimulations.com/display/SQF/nil)	Undefines a variable by assignment.	`any = nil`
isNil (https://sqf.bisimulations.com/display/SQF/isNil)	Checks if a variable is undefined. Checks if the code result is undefined.	`isNil "variable"` `isNil code`
isNull (https://sqf.bisimulations.com/display/SQF/isNull)	Checks if the object is deleted.	`isNull object`
typeName (https://sqf.bisimulations.com/display/SQF/typeName)	Returns the expression type.	`typeName object`
typeOf (https://sqf.bisimulations.com/display/SQF/typeOf)	Returns the `CfgVehicles` class name of the object.	`typeOf object`

6.1.8 Numbers

The Number (on page 21) type has the following common commands:

Command	Description	Syntax
abs (https://sqf.bisimulations.com/display/SQF/abs)	Returns the absolute value of a real number.	`abs number`
acos (https://sqf.bisimulations.com/display/SQF/acos)	Returns the acosine of a cosine.	`acos number`
asin (https://sqf.bisimulations.com/display/SQF/asin)	Returns the asine of a sine.	`asine number`
atan (https://sqf.bisimulations.com/display/SQF/atan)	Returns the arctangent of a tangent.	`atan number`
atan2 (https://sqf.bisimulations.com/display/SQF/atan2)	Returns the arctangent of a vector (in degrees).	`number1 atan2 number2`
ceil (https://sqf.bisimulations.com/display/SQF/ceil)	Returns the ceiling value of a number.	`ceil number`
cos (https://sqf.bisimulations.com/display/SQF/cos)	Returns the cosine of an angle (in degrees).	`cos number`
deg (https://sqf.bisimulations.com/display/SQF/deg)	Converts from radians to degrees.	`deg radians`
exp (https://sqf.bisimulations.com/display/SQF/exp)	Returns the value of e raised to the power of x.	`exp x`
finite (https://sqf.bisimulations.com/display/SQF/finite)	Checks if a number is finite.	`finite number`
floor (https://sqf.bisimulations.com/display/SQF/floor)	Returns the floor value of a number.	`floor number`
ln (https://sqf.bisimulations.com/display/SQF/ln)	Returns a natural logarithm.	`ln number`
log (https://sqf.bisimulations.com/display/SQF/log)	Returns a base-10 logarithm.	`log number`
max (https://sqf.bisimulations.com/display/SQF/max)	Returns the maximum of two numbers.	`number1 max number2`
min (https://sqf.bisimulations.com/display/SQF/min)	Returns the minimum of two numbers.	`number1 min number2`

Command	Description	Syntax
mod (https://sqf.bisimulations.com/display/SQF/mod)	Returns the remainder of the division of two numbers.	`number1 mod number2`
parseNumber (https://sqf.bisimulations.com/display/SQF/parseNumber)	Converts a string into a real number.	`parseNumber string`
pi (https://sqf.bisimulations.com/display/SQF/pi)	Returns the value of pi (in radians).	`pi`
rad (https://sqf.bisimulations.com/display/SQF/rad)	Converts from degrees to radians.	`rad degrees`
random (https://sqf.bisimulations.com/display/SQF/random)	Returns a random number.	`random maximum`
round (https://sqf.bisimulations.com/display/SQF/round)	Rounds a number.	`round number`
sin (https://sqf.bisimulations.com/display/SQF/sin)	Returns the sinus of an angle (in degrees).	`sin number`
sqrt (https://sqf.bisimulations.com/display/SQF/sqrt)	Returns the square root.	`sqrt number`
tan (https://sqf.bisimulations.com/display/SQF/tan)	Returns the tangent of an angle (in degrees).	`tan number`

6.1.9 Strings

The String (on page 21) type has the following common commands:

Command	Description	Syntax
+, =	Concatenates two strings. Compares two strings.	`string1 + string2` `string1 == string2`
format (https://sqf.bisimulations.com/display/SQF/format)	Inserts values into a string of the specified format.	`format [format, values]`
loadFile (https://sqf.bisimulations.com/display/SQF/loadFile)	Returns the content of a given file.	`loadFile filename`
localize (https://sqf.bisimulations.com/display/SQF/localize)	Returns the international representation of a string (for example, a UI label) in VBS4.	`localize stringname`
select (https://sqf.bisimulations.com/display/SQF/select)	Returns an element of the array at the given index. Returns a configuration object at the given index from the configuration file.	`array select index` `config select index`
str (https://sqf.bisimulations.com/display/SQF/str)	Converts any variable to a string.	`str any`
toArray (https://sqf.bisimulations.com/display/SQF/toArray)	Converts a string to an array of numbers.	`toArray string`
toLower (https://sqf.bisimulations.com/display/SQF/toLower)	Converts a string to lowercase.	`toLower string`
toString (https://sqf.bisimulations.com/display/SQF/toString)	Converts an array of numbers to a string.	`toString array`
toUpper (https://sqf.bisimulations.com/display/SQF/toUpper)	Converts a string to uppercase.	`toUpper string`

6.1.10 Array

The Array (on page 23) type has the following common commands:

Command	Description	Syntax
+, -	Concatenates one array with another. Removes a sub-array from an array.	`array1 + array2` `array1 - array2`
count (https://sqf.bisimulations.com/display/SQF/count)	Returns the number of elements in an array.	`count array`
find (https://sqf.bisimulations.com/display/SQF/find)	Returns the first encountered array position with a given value.	`array find value`
in (https://sqf.bisimulations.com/display/SQF/in)	Checks if an array contains a certain value.	`value in array`
resize (https://sqf.bisimulations.com/display/SQF/resize)	Resizes an array to a new given size.	`array resize newsize`
select (https://sqf.bisimulations.com/display/SQF/select)	Returns an element of the array at the given index.	`array select index`
set (https://sqf.bisimulations.com/display/SQF/set)	Sets an element of the array at the given index to a certain value.	`array set [index, value]`

6.1.11 Configuration

The Configuration (Config (on page 153)) type has the following common commands:

Command	Description	Syntax
configClasses (https://sqf.bisimulations.com/display/SQF/configClasses)	Returns an array of configuration classes that match a given condition.	`condition configClasses config`
configName (https://sqf.bisimulations.com/display/SQF/configName)	Returns the name of a configuration entry.	`configName config`
getArray (https://sqf.bisimulations.com/display/SQF/getArray)	Extracts an array from a configuration entry.	`getArray config`
getNumber (https://sqf.bisimulations.com/display/SQF/getNumber)	Extracts a number from a configuration entry.	`getNumber config`
getText (https://sqf.bisimulations.com/display/SQF/getText)	Extracts a string from a configuration entry.	`getText config`
inheritsFrom (https://sqf.bisimulations.com/display/SQF/landAt)	Returns the parent of a given configuration entry.	`inheritsFrom config`
isArray (https://sqf.bisimulations.com/display/SQF/isArray)	Checks if a configuration entry is an array.	`isArray config`
isClass (https://sqf.bisimulations.com/display/SQF/isClass)	Checks if a configuration entry is defined in a given configuration file.	`isClass config`
isKindOf (https://sqf.bisimulations.com/display/SQF/isKindOf)	Checks if the object is a sub-type of a give type.	`object isKindOf type`

Command	Description	Syntax
isNumber (https://sqf.bisimulations.com/display/SQF/isNumber)	Checks if a configuration entry is a file.	`isNumber config`
isText (https://sqf.bisimulations.com/display/SQF/isText)	Checks if a configuration entry is a string.	`isText config`
missionConfigFile (https://sqf.bisimulations.com/display/SQF/missionConfigFile)	Returns the root of the classes defined in `description.ext` (see Scripting with description.ext (on page 187)).	`missionConfigFile`
select (https://sqf.bisimulations.com/display/SQF/select)	Returns a configuration object at the given index from the configuration file.	`config select index`

6.2 Common Command Categories

Here is a list of commands that recur frequently in many scripts, divided by use category:

Command Category	Sub-Categories and Commands	
Controlling AI	Placing Units in Vehicles	moveInDriver (https://sqf.bisimulations.com/display/SQF/moveInDriver), moveInGunner (https://sqf.bisimulations.com/display/SQF/moveInGunner), moveInCommander (https://sqf.bisimulations.com/display/SQF/moveInCommander), moveInTurret (https://sqf.bisimulations.com/display/SQF/moveInTurret), moveInCargo (https://sqf.bisimulations.com/display/SQF/moveInCargo)
	Behavior and Posture	setBehaviour (https://sqf.bisimulations.com/display/SQF/setBehaviour), setCombatMode (https://sqf.bisimulations.com/display/SQF/setCombatMode), setUnitPos (https://sqf.bisimulations.com/display/SQF/setUnitPos)
	Invulnerability	allowDamage (https://sqf.bisimulations.com/display/SQF/allowDamage)
	Weapons and Ammo	addWeapon (https://sqf.bisimulations.com/display/SQF/addWeapon), addMagazine (https://sqf.bisimulations.com/display/SQF/addMagazine)
	Disable parts of the AI	disableAI (https://sqf.bisimulations.com/display/SQF/disableAI)
How to Modify Vehicles	Disable Vehicles	setEngineDisabled (https://sqf.bisimulations.com/display/SQF/setEngineDisabled)
	Disabling Detection	disableGeo (https://sqf.bisimulations.com/display/SQF/disableGeo)
How to Modify Object Properties	Object Positions	getPos (https://sqf.bisimulations.com/display/SQF/getPos), setPos (https://sqf.bisimulations.com/display/SQF/setPos)
	Attaching Objects, 3D Positioning [x,y,z]	attachTo (https://sqf.bisimulations.com/display/SQF/attachTo)

Command Category	Sub-Categories and Commands	
Animations	Unit Animations	playMove (https://sqf.bisimulations.com/display/SQF/playMove), switchMove (https://sqf.bisimulations.com/display/SQF/switchMove), playGesture (https://sqf.bisimulations.com/display/SQF/playGesture)
	Vehicle and Object Animations	animate (https://sqf.bisimulations.com/display/SQF/animate)
Creating Units and Vehicles / Objects	N/A	createUnit (https://sqf.bisimulations.com/display/SQF/createUnit), createVehicle (https://sqf.bisimulations.com/display/SQF/createVehicle)
Deleting Created Units / Objects	N/A	deleteVehicle (https://sqf.bisimulations.com/display/SQF/deleteVehicle)
Creating New Sides and Groups	N/A	createCenter (https://sqf.bisimulations.com/display/SQF/createCenter), createGroup (https://sqf.bisimulations.com/display/SQF/createGroup)
File Operations	N/A	Append (on page 285), AppendLine (on page 286), CloseFile (on page 287), CreateDir (on page 288), Delete (on page 289), Exists (on page 290), Lookup (on page 291), ReadFile (on page 292), RemoveDir (on page 293), Write (on page 294)

6.3 VBS Gateway Script Commands

The following script commands are available to change VBS Gateway settings at runtime.

- Gateway_ApplySettings (https://sqf.bisimulations.com/display/SQF/Gateway_ApplySettings)

  Applies queued settings changes and updates all adapters that were changed.

- Gateway_CenterMotionDevice (https://sqf.bisimulations.com/display/SQF/Gateway_CenterMotionDevice)

  Centers the motion device by sending `IG_CenterMotionDevice` to the IGs.

- Gateway_ChangeSetting (https://sqf.bisimulations.com/display/SQF/Gateway_ChangeSetting)

  Queues a setting change for the given adapter setting.

- Gateway_GetSetting (https://sqf.bisimulations.com/display/SQF/Gateway_GetSetting)

  Retrieves the current value for the given adapter.

- Gateway_Enabled (https://sqf.bisimulations.com/display/SQF/Gateway_Enabled)

  Gets the state of the Gateway UI as set in the settings file.

- Gateway_EnableMotionTracking (https://sqf.bisimulations.com/display/SQF/Gateway_EnableMotionTracking)

  Enables or disables the motion device on the given IGs.

- Gateway_ShowUI (https://sqf.bisimulations.com/display/SQF/Gateway_ShowGUI)

  Shows the Gateway UI in VBS if true, or in the default web browser if false.

- Gateway_SendCommand (https://sqf.bisimulations.com/display/SQF/Gateway_SendCommand)

  Sends script commands as strings through the active adapters to connected clients. If the Script Datum ID matches in the settings files, the other Gateway clients execute the script.

- Gateway_SendString (https://sqf.bisimulations.com/display/SQF/Gateway_SendString)

  Sends strings through the active adapters to connected clients. Other Gateway clients log the message.

- Gateway_ViewAttachGroup (https://sqf.bisimulations.com/display/SQF/Gateway_ViewAttachGroup)

  Loads an XML file of view configurations, or parses an array of view configurations, and attaches them to the designated entity.

- Gateway_ViewGet (https://sqf.bisimulations.com/display/SQF/Gateway_ViewGet)

  Returns the configuration for the view according to the View Configuration array structure.

- Gateway_ViewModify (https://sqf.bisimulations.com/display/SQF/Gateway_ViewModify)

  Updates the view configuration according to the View Configuration array structure.

- Gateway_ViewClearGroup (https://sqf.bisimulations.com/display/SQF/Gateway_ViewClearGroup)

  Removes all views attached to the given entity.

- Gateway_ViewRemove (https://sqf.bisimulations.com/display/SQF/Gateway_ViewRemove)

  Removes the specified views, so they cannot be retrieved by `Gateway_ViewGet`.

- Gateway_ViewSetSensor (https://sqf.bisimulations.com/display/SQF/Gateway_ViewSetSensor)

  Sets IG views to display as various types of sensors.

TIP

If you run a Dedicated Server with VBS Gateway and want to view the output of VBS Gateway script commands in the Admin Client, you can execute script commands:

```
if (isServer) then {
  ServerResult = [
    "DIS", "VerticalOffset", 20
  ] call Gateway_Changesetting;
  publicvariable "ServerResult"
}
```

View the output (value of `ServerResult`) in the Watch field of the Developer Console (on page 138)(in the VBS4 Scripting Manual). The command `Gateway_ChangeSetting` can be replaced with the appropriate script command.

Alternatively, you can also call:

```
executeOnServer [
  "ServerResult = [] call Gateway_Enabled; publicVariable 'ServerResult'"
]
```

Similarly, the output can be viewed in the watch field. The command `Gateway_Enabled` can be replaced with the appropriate script command.

The VBS Scripting Reference is the primary resource on VBS scripting:

https://sqf.bisimulations.com/display/SQF/VBS+Scripting+Reference

Detailed explanations and example uses for Gateway script commands can be found at:

https://sqf.bisimulations.com/display/SQF/VBS+Gateway

6.4 Waypoint Functions and Parameters

Waypoints can be created, modified, and assigned using the following SQF functions:

- fn_vbsCon_waypointCreate (https://sqf.bisimulations.com/display/SQF/fn_vbsCon_waypointCreate) - Creates a waypoint.
- fn_vbsCon_waypointUpdate (https://sqf.bisimulations.com/display/SQF/fn_vbsCon_waypointUpdate) - Modifies an existing waypoint.
- fn_vbsCon_waypointAssign (https://sqf.bisimulations.com/display/SQF/fn_vbsCon_waypointAssign) - Assigns an existing waypoint to a unit, vehicle, group, or another waypoint.

WARNING

These SQF functions and their parameters are experimental and subject to change in future releases of VBS4.

Each of these SQF functions use the following waypoint parameters, which correspond to different Order behaviors (for the Order behavior class names - see Order Behavior Class Names (on page 282)).

For more information on the Order behaviors, see the respective Orders in the VBS Control AI Manual, unless specified otherwise.

Parameter (Data Type and Specific Values)	VBS4 Name / Description	Order Behaviors
`ACCURACY` **Data Type:** Number (on page 21)	Accuracy	Individual - Fire At Order Individual - Move Order
`AIRCRAFT_LOITER_DIRECTION` **Data Type:** String (on page 21) **Specific Values:** `"clockwise"`, `"counterClockwise"`	Direction	Loiter Order
`AIRCRAFT_LOITER_RADIUS` **Data Type:** Number (on page 21)	Loiter Radius	Loiter Order
`AZIMUTH` **Data Type:** Number (on page 21)	Azimuth	Defend Order
`BEHAVIOR_ENTITY` **Data Type:** String (on page 21)	Behavior Tree (Entity)	Custom (see Custom Behaviors)
`BEHAVIOR_GROUP` **Data Type:** String (on page 21)	Behavior Tree (Group)	Custom (see Custom Behaviors)

Parameter (Data Type and Specific Values)	VBS4 Name / Description	Order Behaviors
`BEHAVIOR_LINK` **Data Type:** Object (on page 21)	Assign Next Waypoint	All
`BEHAVIOR` (see Order Behavior Class Names (on page 282)) **Data Type:** String (on page 21)	Behavior	All
`BTSET` **Data Type:** String (on page 21)	BT Set Name	Custom (see Custom Behaviors)
`BTSETPATH` **Data Type:** String (on page 21)	BT Set Path	Custom (see Custom Behaviors)
`CODE_ON_COMPLETION` **Data Type:** Code (on page 153)	Code on Completion	All
`CONDITION_TO_COMPLETE` **Data Type:** Code (on page 153)	Condition to Complete	All
`CONVOY_REACTION_TO_CONTACT` **Data Type:** String (on page 21) **Specific Values:** `"driveIgnore"`, `"driveFire"`	On Visual Contact	Convoy Order
`CONVOY_REACTION_TO_FIRE` **Data Type:** String (on page 21) **Specific Values:** `"driveIgnore"`, `"driveFire"`, `"haltEngage"`	On Taking Fire	Convoy Order
`CONVOY_ROUTE_RESTRICTIONS` **Data Type:** String (on page 21) **Specific Values:** `"StayOnRoad"`, `"RespectLanes"`, `"PreferRoads"`, `"Unrestricted"`	Road Usage	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) Convoy Order Tactical Move Order UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`CONVOY_SPACING` **Data Type:** Number (on page 21)	Vehicle Spacing (m)	Convoy Order UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual)

Parameter (Data Type and Specific Values)	VBS4 Name / Description	Order Behaviors
`CONVOY_SPEED` **Data Type:** Number (on page 21)	Movement Speed (km/h)	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) Convoy Order Tactical Move Order UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`CREW_ROLE_CATEGORY` **Data Type:** String (on page 21) **Specific Values:** `"All"`, `"Crew"`, `"Cargo"`	Crew Role Category	Dismount Order Mount Order
`CYCLE` **Data Type:** Object (on page 21)	Create Cycle	All
`DEPLOYDSB_BRIDGE_LENGTH` **Data Type:** Object (on page 21) **Specific Values:** `"L_46"`, `"L_40"`, `"L_34"`, `"L_28"`, `"L_22"`	Bridge Length	Deploy DSB Order
`DESTINATION_ALTITUDE_MODE` **Data Type:** String (on page 21) **Specific Values:** `"AGL"`, `"ASL"`	Altitude Mode	Fly Order Loiter Order
`DESTINATION_ALTITUDE` **Data Type:** Number (on page 21)	Destination Altitude (m)	Fly Order Loiter Order
`ENTITY_TO_FOLLOW` **Data Type:** Object (on page 21)	Follow Entity	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) Convoy Order Tactical Move Order UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`FLIGHT_SPEED_LIMIT` **Data Type:** Number (on page 21)	Maximum Speed (km/h)	Fly Order Loiter Order

Parameter (Data Type and Specific Values)	VBS4 Name / Description	Order Behaviors
`INDIVIDUAL_REACTION_DIRECT_FIRE` **Data Type:** String (on page 21) **Specific Values:** `"openFire"`, `"holdFire"`	On Direct Fire	Individual - Fire At Order Individual - Move Order
`INDIVIDUAL_REACTION_SPOTTED` **Data Type:** String (on page 21) **Specific Values:** `"openFire"`, `"holdFire"`	On Enemy Spotted	Individual - Fire At Order Individual - Move Order
`INDIVIDUAL_SPEED` **Data Type:** String (on page 21) **Specific Values:** `"slowWalk"`, `"walk"`, `"run"`	Speed	Individual - Fire At Order Individual - Move Order
`INDIVIDUAL_STANCE` **Data Type:** String (on page 21) **Specific Values:** `"standing"`, `"crouched"`, `"prone"`	Stance	Individual - Fire At Order Individual - Move Order Tactical Move Order
`INDIVIDUAL_TARGET_SQF` **Data Type:** String (on page 21)	Target (SQF Code)	Individual - Fire At Order
`INDIVIDUAL_WAIT` **Data Type:** Number (on page 21)	Wait After Finishing (s)	Individual - Fire At Order Individual - Move Order
`INDIVIDUAL_WEAPON_BURST_DELAY_BETWEEN_BURSTS` **Data Type:** Number (on page 21)	Wait Between Bursts (s)	Individual - Fire At Order
`INDIVIDUAL_WEAPON_BURST_NUMBER_OF_BURSTS` **Data Type:** Number (on page 21)	Number of Bursts	Individual - Fire At Order
`INDIVIDUAL_WEAPON_BURST_ROUNDS_PER_BURST` **Data Type:** Number (on page 21)	Rounds per Burst	Individual - Fire At Order
`INDIVIDUAL_WEAPON_CLASS` **Data Type:** String (on page 21) **Specific Values:** `"primary"`, `"secondary"`	Weapon	Individual - Fire At Order
`INDIVIDUAL_WEAPON_FIRE_RATE` **Data Type:** String (on page 21) **Specific Values:** `"single"`, `"burst"`	Firing Mode	Individual - Fire At Order

Parameter (Data Type and Specific Values)	VBS4 Name / Description	Order Behaviors
`INDIVIDUAL_WEAPON_SINGLE_DELAY_BETWEEN_ROUNDS` **Data Type:** Number (on page 21)	Wait Between Rounds (s)	Individual - Fire At Order
`INDIVIDUAL_WEAPON_SINGLE_NUMBER_OF_ROUNDS` **Data Type:** Number (on page 21)	Rounds	Individual - Fire At Order
`INDIVIDUAL_WEAPON_STANCE` **Data Type:** String (on page 21) **Specific Values:** `"lowered"`, `"raised"`	Weapon Stance	Individual - Fire At Order Individual - Move Order
`LOITER_RADIUS` **Data Type:** Number (on page 21)	Loiter Radius	Animal Herd Movement (see Animal AI)
`LOITER_TIMEOUT` **Data Type:** Number (on page 21)	Loiter Timeout	Animal Herd Movement (see Animal AI)
`SIZE_X` **Data Type:** Number (on page 21)	Radius	Suppress Order
`SIZE_Y` **Data Type:** Number (on page 21)	Size (Up-Down)	Suppress Order
`SYNCED` **Data Type:** Object (on page 21)	Sync to Waypoint	All
`TACTICAL_MOVE_REACTION_DISTANCE` **Data Type:** Number (on page 21)	Reaction Distance	Tactical Move Order
`TARGET_GROUP` **Data Type:** Group (on page 22)	Select Target	Pursue Order
`TARGET_VEHICLE` **Data Type:** Object (on page 21)	Select Target	Mount Order
`UGV_AVOID_THREATS` **Data Type:** Boolean (on page 20)	Avoid Threats	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)

Parameter (Data Type and Specific Values)	VBS4 Name / Description	Order Behaviors
`UGV_DEFEND` **Data Type:** Boolean (on page 20)	Defend on Completion	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`UGV_DELAY` **Data Type:** Number (on page 21)	Delay (s)	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`UGV_ENGAGE_ENEMIES` **Data Type:** String (on page 21) **Specific Values:** `"openFire"`, `"holdFire"`, `"returnFire"`	Weapon Control Status	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`UGV_HALT_ON_DETECT` **Data Type:** Boolean (on page 20)	Halt on Detect	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`UGV_WP_REPORT` **Data Type:** Boolean (on page 20)	Send Waypoint Complete Report	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual) UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`VARNAME` **Data Type:** String (on page 21)	Variable Name	All

Parameter (Data Type and Specific Values)	VBS4 Name / Description	Order Behaviors
`WEAPON_CONTROL_STATUS` **Data Type:** String (on page 21) **Specific Values:** `"noChange"`, `"weaponsFree"`, `"holdFire"`	Weapon Control Status	Advance Order Assault Order Defend Order Dismount Order Fly Order Land Order Loiter Order Mount Order Pursue Order Riot (see Define Civilian Riot) Suppress Order Tactical Move Order

Order Behavior Class Names

The following Order behavior class names are available:

Class Name	Order Behavior
`Advance`	Advance Order
`Assault`	Assault Order
`autopilotDefend`	Autopilot - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`convoy`	Convoy Order
`Defend`	Defend Order
`Fly`	Fly Order
`Land`	Land Order
`Loiter`	Loiter Order
`Riot`	Riot (see Define Civilian Riot)
`Suppress`	Suppress Order
`TacticalMove`	Tactical Move Order
`Team_Dismount`	Dismount Order
`Team_Mount`	Mount Order

Class Name	Order Behavior
`Team_Pursue`	Pursue Order
`VBS_Custom`	Custom (see Custom Behaviors)
`vbsCon_animal`	Animal Herd Movement (see Animal AI)
`deployDSB`	Deploy DSB Order
`individual_fire`	Individual - Fire At Order
`individual_move`	Individual - Move Order
`ReturnToFormation`	Return to Formation Order
`umvDefend`	UGV Control - Defend (see AV Assign Waypoints in the VBS4 Trainee Manual)
`moveAggressive`	UGV Control (see AV Assign Waypoints in the VBS4 Trainee Manual)

EXAMPLE

Creating waypoints:

```
_firstWaypoint = [group player, "Advance", getPosASL2 player
vectorAdd [0,100,0]] call fn_vbsCon_waypointCreate;
_secondWaypoint = [_firstWaypoint, "Advance", getPosASL2 player vectorAdd
[0,200,0]] call fn_vbsCon_waypointCreate;
```

Modifying an existing waypoint:

```
[group player, ["VARNAME", "wp1", "WEAPON_CONTROL_STATUS",
"HoldFire"]] call fn_vbsCon_waypointUpdate;
```

Assigning an existing waypoint:

```
[waypoint1, group player] call fn_vbsCon_waypointAssign;
```

6.5 File Operations

VBS comes equipped with a plugin (`VBSPluginFileAccess.dll`) for file operations. This plugin can be found in:

`\VBS_Installation\plugins64\`

> **NOTE**
> The default folder for the file operations is `\VBS_Installation\`.

The following file operations are available:

- Append (on the next page)
- AppendLine (on page 286)
- CloseFile (on page 287)
- CreateDir (on page 288)
- Delete (on page 289)
- Exists (on page 290)
- Lookup (on page 291)
- ReadFile (on page 292)
- RemoveDir (on page 293)
- Write (on page 294)

6.5.1 Append

Adds data to the last line of the file, without a carriage return.

Syntax:

```
Append(fileName)@data
```

Parameter	Basic Data Types (on page 20)	Description
`fileName`	String	The file name.
`data`	Anything	The appended data.

Return Value:

- Nothing - Operation successful.
- String - Error message (the message can be any Operating-System message).

EXAMPLE

```
_status = pluginFunction ["VBSPluginFileAccess","Append(one.txt)@100"];
```

6.5.2 AppendLine

Adds data to the last line of the file, with a carriage return.

Syntax:

```
AppendLine(fileName)@data
```

Parameter	Basic Data Types (on page 20)	Description
`fileName`	String	The file name.
`data`	Anything	The appended data.

Return Value:

- Nothing - Operation successful.
- String - Error message (the message can be any Operating-System message).

EXAMPLE

```
_status = pluginFunction [
  "VBSPluginFileAccess","AppendLine(one.txt)@LineOne"
];
```

6.5.3 CloseFile

Closes file opened for reading with ReadFile (on page 292). Causes the file pointer to be reset to the beginning of the file.

> **NOTE**
> Closing a file that is already closed returns an error.

Syntax:

```
CloseFile(fileName)
```

Parameter	Basic Data Types (on page 20)	Description
`fileName`	String	The file name.

Return Value:

- Nothing - Operation successful.
- String - Error message (the message can be any Operating-System message).

> **EXAMPLE**

```
_status = pluginFunction ["VBSPluginFileAccess","CloseFile(one.txt)"];
```

6.5.4 CreateDir

Creates a folder.

Syntax:

```
CreateDir(folderName)
```

Parameter	Basic Data Types (on page 20)	Description
`folderName`	String	The folder name.

Return Value:

- Nothing - Operation successful.
- String - Error message (the message can be any Operating-System message).

EXAMPLE

```
_status = pluginFunction ["VBSPluginFileAccess","CreateDir(tmp)"];
```

6.5.5 Delete

Deletes a file.

Syntax:

```
Delete(fileName)
```

Parameter	Basic Data Types (on page 20)	Description
`fileName`	String	The file name.

Return Value:

- Nothing - Operation successful.
- String - Error message (the message can be any Operating-System message).

EXAMPLE

```
_status = pluginFunction ["VBSPluginFileAccess","Delete(one.txt)"];
```

6.5.6 Exists

Checks if a file exists.

Syntax:

```
Exists(fileName)
```

Parameter	Basic Data Types (on page 20)	Description
`fileName`	String	The file name.

Return Value:

- True - File exists.
- False - File does not exist.

EXAMPLE

```
_status = pluginFunction ["VBSPluginFileAccess","Exists(one.txt)"];
```

6.5.7 Lookup

Browses a file, finds the first line that contains the specified string, and reads 7 (space separated) numbers that follow the string.

Syntax:

```
Lookup(fileName, lookupString)
```

Parameter	Basic Data Types (on page 20)	Description
`fileName`	String	The file name.
`lookupString`	String	The string to look up.

Return Value:

A string containing the 7 read numbers delimited by commas.

- If the specified file or the look-up string is not found, an empty array is returned.
- Returns 0 for number entries that are not filled.
- Only numbers are returned.

EXAMPLE

```
_status = pluginFunction ["VBSPluginFileAccess","Lookup(one.txt,LineOne)"];
```

NOTE

If the file contains "LineOne 1 22 333", _status is "1,22,333,0,0,0,0".

6.5.8 ReadFile

Reads the current line of the specified file.

After each read, the file pointer is moved to the next line. In order to read the whole file, the command has to be executed repeatedly, until an empty array is returned, designating End of File (EOF).

> **NOTE**
>
> The ReadFile command is able to read lines that have single quotes in them ('), but lines with double quotes (") generate an error.
>
> In older versions of VBS (1.x, 2.x), reading files is very slow, and should only be used for smaller files or in situation which do not impede the simulation (for example, it takes about 10 seconds to read 100 lines).

Syntax:

```
ReadFile(fileName)
```

Parameter	Basic Data Types (on page 20)	Description
`fileName`	String	The file name.

Return Value:

- String - The line read.
- Nothing - If EOF is reached.
- String - Error message (the message can be any Operating-System message).

Example:

Reads the whole content of file `one.txt` and closes the file:

```
waitUntil {
  _line = pluginFunction ["VBSPluginFileAccess","ReadFile(one.txt)"];
  (count _line==0)
};
pluginFunction ["VBSPluginFileAccess","CloseFile(one.txt)"];
```

> **NOTE**
>
> Close the file with CloseFile (on page 287) after it is read.

6.5.9 RemoveDir

Removes a folder.

> **NOTE**
> Only empty folders can be removed.

Syntax:

```
RemoveDir(folderName)
```

Parameter	Basic Data Types (on page 20)	Description
`folderName`	String	The folder name.

Return Value:

- Nothing - Operation successful.
- String - Error message (the message can be any Operating-System message).

> **EXAMPLE**

```
_status = pluginFunction ["VBSPluginFileAccess","RemoveDir(c:\bi\tmp)"];
```

6.5.10 Write

Creates a file for writing.

> **NOTE**
> If the file already exists, it is not created but opened and its content is overwritten.

Syntax:

```
Write(fileName)@data
```

Parameter	Basic Data Types (on page 20)	Description
`fileName`	String	The file name.
`data`	Anything	The data to write to the file.

Return Value:

- Nothing - Operation successful.
- String - Error message (the message can be any Operating-System message).

> **EXAMPLE**

```
_status = pluginFunction ["VBSPluginFileAccess","Write(one.txt)@header"];
```

> **NOTE**
> If the first file entry should have a line carriage, write an empty string, and then add the data via AppendLine (on page 286).