Appendix A – Detailed TGE Scripting Reference

Credits

Joel Baxter – Major credit goes to Joel. Much of the data in this reference was verified against Joel's Resource: 'TGE scripting language reference' (qid=2212). Additionally, Joel has made other excellent resources that you should check out if you are going to expand/customize the scripting features of TGE for your game.

Conventions

Throughout this document, for succinctness, I will refer to the Torque Game Engine Scripting language simply as Torque Script.

Syntax/Rules

Torque Script is a typeless script that is very simliar in syntax to C/C++. You will find that most C/C++ operators function as expected in Torque Script. In addition to providing the strengths of C/C++, Torque Script provides:

- Auto creation and destruction of local/global variables and their storage. (see exceptions below).
- String catenation, comparison, and auto-string-constant creation (see 'Literals' below).

Variables

Literals

Numeric:

123	(decimal)
1.234	(floating point)
1e-3	(scientific notation)
0xabc	(hex)

String:

“abcd”	(string)
‘abcd’	(tagged string)
a1234	(auto-string constant; any alpha followed by alpha or numberic)

Special String Constants:

TAB	(tab)
SPC	(space)
NL	(newline)

Escape Sequences (must be enclosed in double-quotes):

\n	(newline)
\t	(tab)
\c0 … \c9	(colorize subsequent output in console)
\\	(backslash)

Boolean:

The boolean type is present but auto-converts to a numeric (true→1; false→0).

Array:

Torque Script does not support array literals. Don’t take this to mean it doesn’t support arrays, just array literals.

Vector:

Torque Script supports a special type of literal called the ‘vector’ literal. You may encounter this special literal in scripts and assume that it is a string. It takes the general form: “numeric numeric numeric” where the numeric can be any legal numeric type.

Operators in TGE

Operator	Name	Example	Explanation
		Variable Operators	
$	global	$a	$a is a global variable.
%	local	%b	%b is a local variable.
		Arithmetic Operators	
*	multiplication	$a * $b	Multiply $a and $b.
/	division	$a / $b	Divide $a by $b.
%	modulo	$a % $b	Remainder of $a divided by $b.
+	addition	$a + $b	Add $a and $b.
-	subtraction	$a - $b	Subtract $b from $a.
++	auto-increment (post-fix only)	$a++	Increment $a after use. *Note: ++$a is illegal.*
- -	auto-decrement (post-fix only)	$b--	Decrement $b after use. *Note: --$b is illegal.*
		Relational and Logical Operators	
<	Less than	$a < $b	1 if $a is less than % b 0 otherwise.
>	More than	$a > $b	1 if $a is greater than % b 0 otherwise.
<=	Less than or Equal to	$a <= $b	1 if $a is less than or equal to % b 0 otherwise.
>=	More than or Equal to	$a >= $b	1 if $a is greater than or equal to % b 0 otherwise.
==	Equal to	$a == $b	1 if $a is equal to % b 0 otherwise.
!=	Not equal to	$a != $b	1 if $a is not equal to % b 0 otherwise.
!	Logical NOT	!$a	1 if $a is 0 0 otherwise.
&&	Logical AND	$a && $b	1 if $a and $b are both non-zero 0 otherwise.
||	Logical OR		1 if either $a or $b is non-zero 0 otherwise.
		Bitwise Operators	
~	Bitwise complement	~$a	flip bits 1 to 0 and 0 to 1. (i.e. ~10b == 01b)
&	Bitwise AND	$a & $b	composite of elements where bits in same position are 1. (i.e. 1b & 1b == 1b)
|	Bitwise OR	$a | $b	composite of elements where bits 1 in either of the two elements. (i.e. 100b & 001b == 101b)
^	Bitwise XOR	$a ^ $b	composite of elements where bits in same position are opposite. (i.e. 100b & 101b == 001b)
<<	Left Shift	$a << 3	element shifted left by 3 and padded with zeros. (i.e. 11b << 3d == 11000b)
>>	Right Shift	$a >> 3	element shifted right by 3 and padded with zeros. (i.e. 11010b >> 3d == 00011b)

© Hall Of Worlds, LLC. All rights reserved.

Operators in TGE (cont'd)

Operator	Name	Example	Explanation
		Assignment Operators	
=	Assignment	$a = $b;	Assign value of $b to $a.
op=	Compound Assignment	$a ***op***= $b;	Equivalent to $a = $a ***op*** $b. ***op*** can be any of: *** / % + - & | ^ << >>**
		String Operators/Constants	
@	String catenation	$c @ $d	Concatenates strings $c and $d into a single string. *Numeric literals/variables convert to strings.*
NL	New Line	$c NL $d	Same as catenation example with new-line between $c and $d.
TAB	Tab	$c TAB $d	Same as catenation example with tab between $c and $d.
SPC	Space	$c SPC $d	Same as catenation example with space between $c and $d.
$=	String equal to	$c $= $d	1 if $c equal to $d .
!$=	String not equal to	$c !$= $d	1 if $c not equal to $d.
		Miscellaneous Operators	
? :	Conditional	x ? y : z	Substitute y if x equal to 1, else substitute z.
[]	Array element	$a[5]	Sixth element or array $a
.	Member/Method selection		
( )	Grouping		
{ }	Blocking		
,	Listing		
::	Namespace		
" "	String constant (normal)		
' '	String constant (tagged)		

Keywords

break	case	continue	datablock	default
else	false	for	function	if
new	or	package	parent	return
switch	switch$	true	while	

Note: Although keywords are not reserved, it is considered bad practice to use variables that have the same spelling as a keyword.

Note2: All examples in this section can be cut and pasted directly into the console unless otherwise noted.

---

break

Purpose
Use *break* to exit the innermost *for* or *while* loop. *break* can also be used to exit a switch or switch$ statement.

Console Example
```
==> %count = 0; while(1) { echo(%count++); if (%count > 2) break; }
1
2
3
```

See Also
case, if, switch, switch$, while

---

case

Purpose
Used to label cases in a *switch* or *switch$* statement.

Example
See *switch* and *switch$* examples.

See Also
break, switch, switch$

continue

Purpose
The *continue* keyword causes the script to skip the remainder of the innermost loop in which it appears;

Console Example
==> %count = 0; while(%count++ < 8) { if (%count > 2) continue; echo(%count); }
1
2

See Also
for, while

datablock

Purpose
A short statement explaining the purpose and use of this function call

Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

x

default

Purpose
This labels the default case in a *switch* or *switch$* statement. i.e. the case that is executed if no other cases matche the *switch*/*switch$* value.

Note: The *default* keyword must be listed after all *case* keywords. It is a syntax error to place it before subsequent *case* keywords.

Example
See *switch* and *switch$* examples.

See Also
break, switch, switch$

else

Purpose
The *else* keyword is used with the *if* keyword to control the flow of a script. The general form of the well known if-then-else construct is as follows,

```
if (expression) { statement(s); } else { alternate statement(s); }
```

Where the alternate statement(s) are executed if the expression evaluates to 0.

Console Example
See *if* example.

See Also
if

false

Purpose
The *false* keyword is used for boolean comparison and evalulates to 0.

Console Example
==> if(false == 0) { echo(“false evaulates to” SPC 0); }
false evaluates to 0

See Also
if, true

Purpose
The *for* keyword is a looping construct whose general form is:

```
for (expression0; expression1; expression2) {
   statements(s);
}
```

Where:

- expression0 is usually of the form: variable = value
- expression1 is usually of the form: variable *compare op* value
- expression2 is usally of the form: variable *op* OR variable *op* value

, and the loop continues to execute statement(s) until expression0 evaluates to false (i.e. 0).

Note: Unlike C and C++ expression0, expression1, and expression2 are all required. If you absolutely need expression0 or expression2 to be empty just insert a 0.
Note2: Composite expressions of the form (sub_expression0,sub_expression1, … sub_expressionN) are illegal.

Console Example

```
==> for(%value = 0; %value < 3; %value++) { echo(%value);}
0
1
2
==> %value = 0; for(0; %value < 3; 0) { echo(%value); %value ++;}
0
1
2
```

See Also
break, continue

© Hall Of Worlds, LLC. All rights reserved.

function

Purpose
A short statement explaining the purpose and use of this function call

Script Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

if

Purpose
The *if* keyword is used with or without the *else* keyword to control the flow of a script. The general form of the well known if-then-else construct is as follows,

```
if (expression) {
   statement(s);
} else {
   alternate statement(s);
}
```

Where the statement(s) are executed if the expression evaluates to a non-zero value.

Console Example
==> if(0) { echo(“hello”); } else { echo(“goodbye”); }
goodbye
==> if(5) { echo(“hello”); } else { echo(“goodbye”); }
hello

See Also
else

new

Purpose
A short statement explaining the purpose and use of this function call

Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

or

Purpose
TBD may not be correct

Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

x

package

Purpose
A short statement explaining the purpose and use of this function call

Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

parent

Purpose
A short statement explaining the purpose and use of this function call

Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

x

return

Purpose
The *return* keyword is used to return a value from a *function*

Console Example
==> function equal_to(%arg0, %arg1) { return (%arg0 == %arg1); }
==> echo(equal_to(10,11));
0
==> echo(equal_to(11,11));
1

See Also
function

switch

Purpose
The *switch* keyword is used to control the flow of a script. The general form of a switch statement is as follows:

```
switch (expression) {
case value0:
   statement(s);
   break;
case value1:
   statement(s);
   break;
. . .
case valueN:
   statement(s);
   break;
default:
   statement(s);
}
```

Where expression is evaluated and the subsequently compared to the following case values. If a case matches the evaluated expression, the statement(s) associated with that case are executed. If no values match and a default statement exists, the statement(s) in the default case will be executed.

switch is used ONLY for expressions that evaluate to a numeric value.

Note: Unlike C/C++, the break statements in switches are superfluous. Torque Script will only execute matching cases and NOT automatically execute all subsequent cases. This is proven in the example below.

Example
==> switch(%tmp = 1) { case 0: echo(0); case 1: echo(1); default: echo("proof"); }
1

See Also
break, case, default, switch$

switch$

Purpose
The *switch$* keyword is used to control the flow of a script. The general form of a switch statement is as follows:

```
switch (expression) {
case string_value0:
   statement(s);
   break;
case string_value1:
   statement(s);
   break;
. . .
case string_valueN:
   statement(s);
   break;
default:
   statement(s);
}
```

Where expression is evaluated and subsequently compared to the following case values. If a case string_value matches the evaluated expression, the statement(s) associated with that case are executed. If no values match and a default statement exists, the statement(s) in the default case will be executed.

switch$ is used ONLY for expressions that evaluate to a string value.

Note: Unlike C/C++, the break statements in switches are superfluous. Torque Script will only execute matching cases and NOT automatically execute all subsequent cases.

Example
==> switch$(%tmp = “hi”) { case “bye”: echo(“bye”); case “hi”: echo(“hi”); }
hi

See Also
break, case, default, switch

true

Purpose
The *true* keyword is used for boolean comparison and evalulates to 1.

Console Example
==> if(true == 1) { echo("true evaulates to" SPC 1); }
true evaluates to 1

See Also
if, true

x

while

Purpose
The *while* keyword is a looping construct whose general form is:

```
while (expression) {
   statements(s);
}
```

Where expression is usually of the form: variable *compare op* value, and the loop continues to execute statement(s) until expression evaluates to false (i.e. 0).

Console Example
==> %val=5; while(%val) { echo(%val--); }
4
3
2
1
0

See Also
Mention related function names.

x

function_name

Purpose
A short statement explaining the purpose and use of this function call

Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

x

---

function_name

Purpose
A short statement explaining the purpose and use of this function call

Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

x

---

function_name

Purpose
A short statement explaining the purpose and use of this function call

Example
Give a clear and concise example program section containing this function.

See Also
Mention related function names.

x

Scope and Visibility
Discuss namespaces here.

Functions and Methods
All

Packages
All

Arithmetics Ops
All

Conversion Ops
All

Arrays and Other Structures
All

Regular expressions
All

Search and Replace Functions
All

File Operations
All

I/O
All

System Interaction
All

Networking
All

Special
All

The Console
All