

The Wayback Machine - http://web.archive.org/web/20160324212757/http://wiki.wiremod.com/wiki/Expression\_2

# Expression 2

From Wiremod Wiki  
Jump to: [navigation](#), [search](#)

## Contents

- [1 Features](#)
  - [1.1 Syntax](#)
    - [1.1.1 Variables](#)
    - [1.1.2 @trigger directive](#)
    - [1.1.3 @model directive](#)
    - [1.1.4 @autoupdate directive](#)
  - [1.2 Loops](#)
  - [1.3 Performance](#)
  - [1.4 Editor Shortcut Keys](#)
- [2 Tutorials](#)
- [3 Console Commands](#)
- [4 Datatypes](#)
  - [4.1 Number](#)
    - [4.1.1 Description](#)
    - [4.1.2 Commands](#)
  - [4.2 String](#)
    - [4.2.1 Description](#)
    - [4.2.2 Commands](#)
  - [4.3 Entity](#)
    - [4.3.1 Description](#)
    - [4.3.2 Commands](#)
  - [4.4 Vector](#)
    - [4.4.1 Description](#)
    - [4.4.2 2D Vector Commands](#)
    - [4.4.3 3D Vector Commands](#)
    - [4.4.4 4D Vector Commands](#)
    - [4.4.5 Common Vector Commands](#)
  - [4.5 Matrix](#)
    - [4.5.1 Description](#)
    - [4.5.2 2x2 Matrix Commands](#)
    - [4.5.3 2x2 Matrix Commands](#)
    - [4.5.4 3x3 Matrix Commands](#)
    - [4.5.5 4x4 Matrix Commands](#)
    - [4.5.6 Common Matrix Commands](#)
  - [4.6 Angle](#)
    - [4.6.1 Description](#)
    - [4.6.2 Commands](#)
  - [4.7 Table](#)
    - [4.7.1 Description](#)
    - [4.7.2 Related Examples](#)
    - [4.7.3 Commands](#)

- [4.8 Array](#)
  - [4.8.1 Description](#)
  - [4.8.2 Commands](#)
- [4.9 Bone](#)
  - [4.9.1 Description](#)
  - [4.9.2 Commands](#)
- [4.10 Wirelink](#)
  - [4.10.1 Description](#)
  - [4.10.2 Commands](#)
- [4.11 Complex](#)
  - [4.11.1 Description](#)
  - [4.11.2 Commands](#)
- [4.12 Quaternion](#)
  - [4.12.1 Description](#)
  - [4.12.2 Commands](#)
- [5 Basic extensions](#)
  - [5.1 Core](#)
    - [5.1.1 Description](#)
    - [5.1.2 Commands](#)
  - [5.2 Self-Aware](#)
    - [5.2.1 Description](#)
    - [5.2.2 Commands](#)
  - [5.3 Debug](#)
    - [5.3.1 Description](#)
    - [5.3.2 Commands](#)
  - [5.4 Timer](#)
    - [5.4.1 Description](#)
    - [5.4.2 Commands](#)
    - [5.4.3 Other](#)
  - [5.5 Unit Conversion](#)
    - [5.5.1 Description](#)
    - [5.5.2 Commands](#)
    - [5.5.3 Units](#)
  - [5.6 Server Information](#)
    - [5.6.1 Description](#)
    - [5.6.2 Commands](#)
  - [5.7 Constraint](#)
    - [5.7.1 Description](#)
    - [5.7.2 Commands](#)
  - [5.8 Chat](#)
    - [5.8.1 Description](#)
    - [5.8.2 Commands](#)
  - [5.9 Color](#)
    - [5.9.1 Description](#)
    - [5.9.2 Commands](#)
- [6 Advanced extensions](#)
  - [6.1 E2 Function System](#)
    - [6.1.1 Description](#)
  - [6.2 Callable strings](#)
    - [6.2.1 Description](#)
    - [6.2.2 Notable features](#)
  - [6.3 Entity Discovery](#)

- [6.3.1 Description](#)
  - [6.3.2 Commands](#)
- [6.4 Global Variables](#)
  - [6.4.1 Description](#)
  - [6.4.2 Commands](#)
- [6.5 Built-In Ranger](#)
  - [6.5.1 Description](#)
  - [6.5.2 Commands](#)
- [6.6 Sound Playback](#)
  - [6.6.1 Description](#)
  - [6.6.2 Commands](#)
- [6.7 NPC control](#)
  - [6.7.1 Description](#)
  - [6.7.2 Commands](#)
- [6.8 Signals](#)
  - [6.8.1 Description](#)
    - [6.8.1.1 Scope](#)
    - [6.8.1.2 Group](#)
  - [6.8.2 Commands](#)
- [6.9 Data Signals](#)
  - [6.9.1 Description](#)
    - [6.9.1.1 Scope](#)
    - [6.9.1.2 Group](#)
    - [6.9.1.3 Signal Names](#)
  - [6.9.2 Commands](#)
- [6.10 GLON](#)
  - [6.10.1 Description](#)
    - [6.10.1.1 NOTE](#)
  - [6.10.2 Commands](#)
- [6.11 vON](#)
  - [6.11.1 Description](#)
  - [6.11.2 Commands](#)
- [6.12 3D Holograms](#)
  - [6.12.1 Description](#)
  - [6.12.2 Console Variables](#)
  - [6.12.3 Commands](#)
- [6.13 File Functions](#)
  - [6.13.1 Description](#)
  - [6.13.2 Commands](#)
- [6.14 HTTP Functions](#)
  - [6.14.1 Description](#)
  - [6.14.2 Console Variables](#)
  - [6.14.3 Commands](#)
- [6.15 Bitwise](#)
  - [6.15.1 Description](#)
  - [6.15.2 Commands](#)
- [6.16 E2 Variable Scopes](#)
  - [6.16.1 Description](#)
  - [6.16.2 Example](#)
- [6.17 Include directive](#)
  - [6.17.1 Description](#)
  - [6.17.2 Example](#)

- [7 See Also](#)
- [8 Credits](#)

This wiki has been created so that everyone can contribute to the documentation, so please do. If what you are looking for isn't listed here, use [this board](#) at the [Wiremod forums](#).

## Features

### Syntax

The syntax in the Expression 2 will take some time to get used to, but will give you a lot of power over your expressions. Remember there is not just one way to code something, you can accomplish the same task in several ways.

In Expression 2 conditionals, numbers are considered false if they equal 0. Otherwise, they are considered true. Functions and methods which conceptually return "true" or "false" return 1 and 0 respectively.

Syntax	Example	Description
#	#This is a comment	Anything written after "#" on the same line will be treated as a comment (it will be ignored when the chip runs)
#[Comment]#	#[This is a multi-line comment]#	Anything between # brackets will be treated as a comment (it will be ignored when the chip runs)
if () {}	if (A) {B = C}	If the value (the condition) between the parentheses is true, then run the statements between the braces.
else {}	else {A = B}	<b>Must be preceded by if () {} or elseif () {}.</b> If the previous condition was <i>false</i> then run the statements between the braces.
elseif () {}	elseif (A) {B = C}	<b>Must be preceded by if () {} or elseif () {}.</b> If the previous condition was false <i>and</i> the value between the parentheses is true, then run the statements between the braces.
( ? : )	D = A ? B : C	If A is true then return B otherwise return C. Note: A ? : B is the same as A ? A : B
&	if (A & B) {C = 1}	Returns 1 if both A and B are true.
	if (A   B) {C = 1}	Returns 1 if A or B is true.
!	if (!A) {B = 1}	<b>Must precede a value.</b> Returns 1 if A is false (same as A == 0).
~	if (~Button & Button) {DO stuff}	<b>Must precede an input variable.</b> Returns 1 if the current execution was caused by a change in the input variable.
\$	Opitch = Pitch + \$Pitch * 3	<b>Must precede an input/output/persistent variable which has a subtraction operator.</b> Returns the current value of the variable minus the value of the variable at the end of the last execution. Note: Acceleration == \$Velocity only when intervals are one second apart.
		Can also be used to check if a variable has changed.
->	->Input or ->Output	If you use it on an input, it returns 1 if the input is wired to something. If you use it on an output, it returns the number of wires to the output.
#ifdef	#ifdef entity:setAlpha(number)	Checks if the given Function is usable on the Server.
#else	#else	<b>Must be preceded by #ifdef.</b> If the given Function is NOT usable on the Server then run the Code after it.
#endif	#endif	<b>Must be preceded by #ifdef or #else.</b> Closes the #ifdef Statement.

### Variables

In Expression 2, all variables must start with a capital letter. For instance:

Syntax	Description
@persist variable	This will <b>not</b> work, as the "variable" does not begin with a capital letter.
@persist Variable	This, however, will work, as "Variable" begins with a capital letter.

### @trigger directive

The trigger directive can selectively enable or disable inputs from triggering executions. Possible values are all/none, but also a list of inputs.

Syntax	Description
@trigger all	Default behavior, inputs always trigger executions
@trigger none	Will never trigger from inputs, only timers, etc
@trigger Time Button	Will only trigger if the inputs Time or Button has changed

@model directive

The model directive can be used to set the model of your expression chip. @model \*model path\*

@autoupdate directive

Using the autoupdate directive will enable auto updating. What this means is that whenever you paste a duplication of an E2 with autoupdate enabled, the E2 will check your files for a new version of that E2 and upload it. Note:

- Only works on saved E2s.
- To disable autoupdate, simply don't write @autoupdate
- If you for some reason need to get an old version of your E2 from a dupe, you will have to temporarily change the name of your E2 so that the autoupdate feature doesn't find the file. Then it'll silently fail and leave you with the old code.

Loops

E2 features three types of loop: while loops, for loops and foreach loops. These allow instructions to be repeated many times in one execution, rather than relying on multiple executions to perform repeat tasks. Be aware that loops that perform too many instructions during an execution will exceed the E2's op quota and cause it to stop running (see "Performance" below).

Syntax	Example	Description
while () {}	while (A) {B++}	Any instructions between the braces will repeat, as long as the condition between the parentheses is true. If the condition is false, the instructions will be skipped and the E2 will continue from the end of the loop. Note that the condition is only checked at the very start of each loop.
for () {}	for (I = A, B[, C]) {}	Adds a value C to a Variable over every iteration until it equals a value B. If a step C is not specified it will be set to one. Note that A and B are evaluated only once - if they are calculated from variables that the loop sets to new values, this will not change the number of iterations.
foreach () {}	foreach(K,V:type=Table) {}	Loops over each element of the specified type in Table. Assigns the key to K and the value to V. Elements that are not of the specified types are skipped. Elements can be added, removed, or modified, however elements that are added will not be processed in the current loop. <b>Will only loop through string indexes when using tables.</b>
continue	if (A) {continue}	<b>Can only be used within a while/for/foreach loop.</b> This will immediately return to the start of the loop, skipping any following instructions.
break	if (A) {break}	<b>Can only be used within a while/for/foreach loop.</b> This will immediately go to the end of the loop and exit it, skipping any following instructions.

Performance

Did some performance tests in Garry's Mod to see what it could churn out:

~3.000.000/s arithmetic operations (+-\*/) on numbers, 2000 ops, 200 executions in 0.13s

~850.000/s arithmetic operations (+-\*/) on vectors, 2000 ops, 100 executions in 0.23s

Although E2 can do a lot, it has quotas to prevent servers from lagging. If the amount of ops used in a tick exceeds the tickquota, then the expression shuts down. Every tick, the ops used minus the softquota is added to a counter, if that counter exceeds the hardquota, then the expression shuts down. If your expression is shutting down, then it is recommended you learn better coding practices, such as storing commonly used values in a variable.

Editor Shortcut Keys

Shortcut	Description
Ctrl-Space	Validate (and move cursor to error)
Ctrl-S	Save
Ctrl-Q	Close
Ctrl-Z	Undo
Ctrl-Y	Redo
Ctrl-X	Cut
Ctrl-C	Copy
Ctrl-V	Paste
Ctrl-A	Select all
Ctrl-I / Tab	Indent
Ctrl-O / Shift-Tab	Outdent
Ctrl-K	Comment the selected block
Ctrl-Shift-K	Uncomment the selected block
Ctrl-F	Find
Ctrl-H	Replace
Ctrl-Up	Scroll up
Ctrl-Down	Scroll down
Ctrl-Left	Jump one word left
Ctrl-Right	Jump one word right
Ctrl-Home	Go to first line
Ctrl-End	Go to last line

Tutorials

If you make a guide related to E2, please feel free to add it to this list!

- [ElementWire's YouTube channel](#)
- [UltimateWire's YouTube channel](#)
- [NickMBR's YouTube channel PT-BR](#)
- [Revan's Expression Gate 2 Guide - The Basics](#)
- [In-depth guide by Matte](#)
- [Player Targeting E2 in detail](#)
- [Expression 2 Guide](#)
- [Expression 2 Examples](#)
- [E2 And Plugs with Buttons](#)

Don't see the tutorial you need here? Check out the [tutorial list](#) at the fourms.

Still can't find the tutorial you need? [Ask for help.](#)

Console Commands

Command	Description

wire_expression2_model <model>	Manually changes the expression's model.
wire_expression2_reload	Reloads all E2 extensions, useful for debugging your own extensions. Keep in mind that client-side files will be taken from gmod's cache in multi-player mode.
wire_expression2_debug 0/1	Toggles debug mode, which shows info that might be useful for the developers. You need to do "wire_expression2_reload" after changing for this to have any effect.
wire_expression2_extension_enable <extension>	Enables the specified extension.
wire_expression2_extension_disable <extension>	Disables the specified extension.
wire_expression2_unlimited 0/1	Enables/disables performance limiting.

Note: Most E2 addons do not register themselves as extensions and thus cannot be turned on or off with the above commands. A notable exception is the [propcore](#) extension in the UWSVN, which is disabled by default.

Also, you must run wire\_expression2\_reload in order for enabling/disabling extensions to take effect.

## Datatypes

Expression2 uses several datatypes and to keep the wiki at a reasonable size we use a shorthand for those datatypes, here is a list of all the shorthands used and the datatype they represent.

Shorthand	Datatype
<b>N</b>	<a href="#">Number</a>
<b>V2</b> / <b>V</b> / <b>V4</b>	<a href="#">2D / 3D / 4D Vector</a>
<b>A</b>	<a href="#">Angle</a>
<b>S</b>	<a href="#">String</a>
<b>E</b>	<a href="#">Entity</a>
<b>R</b>	<a href="#">Array</a>
<b>T</b>	<a href="#">Table</a>
<b>RD</b>	<a href="#">Ranger Data</a>
<b>B</b>	<a href="#">Bone</a>
<b>M2</b> / <b>M</b> / <b>M4</b>	<a href="#">2x2 / 3x3 / 4x4 Matrix</a>
<b>WL</b>	<a href="#">Wirelink</a>
<b>C</b>	<a href="#">Complex number</a>
<b>Q</b>	<a href="#">Quaternion</a>

### Number

#### Description

Numbers, lots of them...

#### Commands

Function	Returns	Description
N + N	<b>N</b>	Addition
N - N	<b>N</b>	Subtraction
N * N	<b>N</b>	Multiplication
N / N	<b>N</b>	Division
N ^ N	<b>N</b>	Exponentiation (N to the power of N)

N % N	<div>N</div>	Modulo, returns the Remainder after Argument 1 has been divided by Argument 2. Note "-1 % 3 = 2"
mod(N, N)	<div>N</div>	Modulo, returns the Remainder after Argument 1 has been divided by Argument 2. Note "mod(-1, 3) = -1"
sqrt(N)	<div>N</div>	Returns the Square Root of the Argument
cbrt(N)	<div>N</div>	Returns the Cube Root of the Argument
root(N, N)	<div>N</div>	Returns the Nth Root of the first Argument
e()	<div>N</div>	Returns Euler's Constant
exp(N)	<div>N</div>	Returns e to the power of the Argument (same as e()^N but shorter and faster this way)
ln(N)	<div>N</div>	Returns the logarithm to base e of the Argument
log2(N)	<div>N</div>	Returns the logarithm to base 2 of the Argument
log10(N)	<div>N</div>	Returns the logarithm to base 10 of the Argument
log(N,N)	<div>N</div>	Returns the logarithm to base Argument 2 of Argument 1
abs(N)	<div>N</div>	Returns the Magnitude of the Argument
ceil(N)	<div>N</div>	Rounds the Argument up to the nearest Integer
ceil(N,N)	<div>N</div>	Rounds Argument 1 up to Argument 2's decimal precision
floor(N)	<div>N</div>	Rounds the Argument down to the nearest Integer
floor(N,N)	<div>N</div>	Rounds Argument 1 down to Argument 2's decimal precision
round(N)	<div>N</div>	Rounds the Argument to the nearest Integer
round(N,N)	<div>N</div>	Rounds Argument 1 to Argument 2's decimal precision
int(N)	<div>N</div>	Returns the Integer part of the Argument (same as floor(N))
frac(N)	<div>N</div>	Returns the Fractional part of the Argument (same as N - floor(N))
clamp(N,N,N)	<div>N</div>	If Arg1 <= Arg2 (min) returns Arg2; If Arg1 >= Arg3 (max) returns Arg3; otherwise returns Arg1.
inrange(N,N <sub>2</sub> ,N <sub>3</sub> )	<div>N</div>	Returns 1 if N is in the interval [N <sub>2</sub> ; N <sub>3</sub> ], 0 otherwise. This means it is equivalent to ((N <sub>2</sub> <= N) & (N <= N <sub>3</sub> ))
sign(N)	<div>N</div>	Returns the sign of argument (-1,0,1) [ <i>sign(N) = N / abs(N)</i> ]
min(N,N)	<div>N</div>	Returns the lowest value Argument
min(N,N,N)	<div>N</div>	Returns the lowest value Argument
min(N,N,N,N)	<div>N</div>	Returns the lowest value Argument
max(N,N)	<div>N</div>	Returns the highest value Argument
max(N,N,N)	<div>N</div>	Returns the highest value Argument
max(N,N,N,N)	<div>N</div>	Returns the highest value Argument
random()	<div>N</div>	Returns a random floating-point number between 0 and 1 [ <i>0 &lt;= x &lt; 1</i> ]
random(N)	<div>N</div>	Returns a random floating-point number between 0 and the specified value [ <i>0 &lt;= x &lt; a</i> ]
random(N,N)	<div>N</div>	Returns a random floating-point number between the specified interval [ <i>a &lt;= x &lt; b</i> ]
randint(N)	<div>N</div>	Returns a random integer from 1 to the specified value [ <i>1 &lt;= x &lt;= a</i> ]
randint(N,N)	<div>N</div>	Returns a random integer in the specified interval [ <i>a &lt;= x &lt;= b</i> ]
pi()	<div>N</div>	Returns the constant PI
toRad(N)	<div>N</div>	Converts Degree angles to Radian angles
toDeg(N)	<div>N</div>	Converts Radian angles to Degree angles
sin(N)	<div>N</div>	Returns the sine of N degrees
cos(N)	<div>N</div>	Returns the cosine of N degrees
tan(N)	<div>N</div>	Returns the tangent of N degrees
cot(N)	<div>N</div>	Returns the cotangent of N degrees
sec(N)	<div>N</div>	Returns the secant of N degrees



csc(N)	N	Returns the cosecant of N degrees
asin(N)	N	Returns the inverse sine of the argument, in degrees
acos(N)	N	Returns the inverse cosine of the argument, in degrees
atan(N)	N	Returns the inverse tangent of the argument, in degrees
atan(N,N)	N	Returns the inverse tangent of the arguments (arg1 / arg2), in degrees. This function accounts for positive/negative arguments, and arguments at or close to 0. Commonly known as atan2.
sinh(N)	N	Returns the hyperbolic sine of N
cosh(N)	N	Returns the hyperbolic cosine of N
tanh(N)	N	Returns the hyperbolic tangent of N
coth(N)	N	Returns the hyperbolic cotangent of N
sech(N)	N	Returns the hyperbolic secant of N
csch(N)	N	Returns the hyperbolic cosecant of N
sinr(N)	N	Returns the sine of N radians
cosr(N)	N	Returns the cosine of N radians
tanr(N)	N	Returns the tangent of N radians
cotr(N)	N	Returns the cotangent of N radians
secr(N)	N	Returns the secant of N radians
cscr(N)	N	Returns the cosecant of N radians
asinr(N)	N	Returns the inverse sine of the argument, in radians
acosr(N)	N	Returns the inverse cosine of the argument, in radians
atanr(N)	N	Returns the inverse tangent of the argument, in radians
atanr(N,N)	N	Returns the inverse tangent of the arguments (arg1 / arg2), in radians. This function accounts for positive/negative arguments, and arguments at or close to 0. Commonly known as atan2.
A = B	N	Assignment (set A equal to B)
A++	N	Assignment adding 1 (increases A by 1, same as "A += 1")
A--	N	Assignment subtracting 1 (decreases A by 1, same as "A -= 1")
A += B	N	Assignment using addition (increases A by B, same as "A = (A + B)")
A -= B	N	Assignment using subtraction (decreases A by B, same as "A = (A - B)")
A /= B	N	Assignment using division
A %= B	N	Assignment using modulo
A ^= B	N	Assignment using exponentiation
A == B	N	Returns 1 if A is equal to B
A != B	N	Returns 1 if A is not equal to B
A > B	N	Returns 1 if A is greater than B
A < B	N	Returns 1 if A is less than B
A >= B	N	Returns 1 if A is greater or equal to B
A <= B	N	Returns 1 if A is less than or equal to B

String

Description

String support allows you to manipulate text with E2. Text screens now have an input for strings.

Commands

Create a string by wrapping the text in quotation marks, for example; "text goes here". Equal (==) and Not equal (!=) operators are available, as is concatenation (+), for joining strings and numbers in any order. Concatenation returns a string. The first character of a string has the index 1. Negative indices are counted from the end of the string, with the last character being -1. Positive indices will be capped to the string's length.

Function	Returns	Description
S[N]	<div>S</div>	Returns Nth letter of the string, formatted as a string. Read-only.
S:index(N)	<div>S</div>	Returns Nth letter of the string, formatted as a string.
S:length()	<div>N</div>	Returns the length of the string.
S:upper()	<div>S</div>	All characters are made uppercase
S:lower()	<div>S</div>	All characters are made lowercase
S:sub(N)	<div>S</div>	Returns a substring, starting at the number argument and ending at the end of the string
S:sub(N,N)	<div>S</div>	Returns a substring, starting at the first number argument and ending at the second
S:left(N)	<div>S</div>	Returns N amount of characters starting from the leftmost character
S:right(N)	<div>S</div>	Returns N amount of characters starting from the rightmost character
S:find(S)	<div>N</div>	Returns the 1st occurrence of the string S, returns 0 if not found
S:find(S, N)	<div>N</div>	Returns the 1st occurrence of the string S starting at N and going to the end of the string, returns 0 if not found
S:findRE(S)	<div>N</div>	Returns the 1st occurrence of the string S using REGEX functions, returns 0 if not found
S:findRE(S, N)	<div>N</div>	Returns the 1st occurrence of the string S starting at N and going to the end of the string using REGEX functions, returns 0 if not found
S:explode(S)	<div>R</div>	Splits the string into an array, along the boundaries formed by the string S. See also <a href="#">String.Explode</a>
S:repeat(N)	<div>S</div>	Repeats the input string N times
S:trim()	<div>S</div>	Trims away spaces at the beginning and end of a string
S:trimLeft()	<div>S</div>	Trims away opening spaces on the string
S:trimRight()	<div>S</div>	Trims away spaces at the end of a string
S:replace(S,S)	<div>S</div>	Finds and replaces every occurrence of the first argument with the second argument
S:replaceRE(S,S)	<div>S</div>	Finds and replaces every occurrence of the first argument using REGEX with the second argument
S:reverse()	<div>S</div>	Returns a reversed version of <i>S</i>
S:number()	<div>N</div>	Parses a number from a string.
S:number(N)	<div>N</div>	Parses a number from a string. The argument given is the base. I.e. number(16) will parse hex.
tostring(N)	<div>S</div>	Formats a number as a string. (Numbers may be concatenated into a string without using this function)
N:tostring()	<div>S</div>	Formats a number as a string. (Numbers may be concatenated into a string without using this function)
tostring(N,N)	<div>S</div>	Formats a number as a string, using argument 2 as the base. i.e. using 16 for base would convert the number to hex.
N:tostring(N)	<div>S</div>	Formats a number as a string, using argument 2 as the base. i.e. using 16 for base would convert the number to hex.
tochar(N)	<div>S</div>	Returns a one-character string from its <a href="#">ASCII code</a> , where 32 = argument 1 = 255. An empty string is returned for numbers outside that range.
toByte(S)	<div>N</div>	Returns the ASCII code of the 1st character in the string
toByte(S,N)	<div>N</div>	Returns the ASCII code of the Nth character in the string
format(S,...)	<div>S</div>	Formats a values exactly like Lua's <a href="#">string.format</a> . Any number and type of parameter can be passed through the "...". Prints errors to the chat area.
S:match(S2)	<div>R</div>	runs <a href="#">string.match</a> ( <i>S</i> , <i>S2</i> ) and returns the sub-captures as an array.
S:match(S2,N)	<div>R</div>	runs <a href="#">string.match</a> ( <i>S</i> , <i>S2</i> , <i>N</i> ) and returns the sub-captures as an array.
S:matchFirst(S2)	<div>S</div>	runs <a href="#">string.match</a> ( <i>S</i> , <i>S2</i> ) and returns the first match or an empty string if the match failed.
S:matchFirst(S2,N)	<div>S</div>	runs <a href="#">string.match</a> ( <i>S</i> , <i>S2</i> , <i>N</i> ) and returns the first match or an empty string if the match failed.

Entity

Description





























These entity functions allow you to get information from, and directly manipulate, entities in the game world (such as props). Entities can be found using many methods, from target finders, entity markers and even the expression itself with entity() from selfaware.

Since the expression collects the data directly from the entity, it is much faster to handle calculations from within the E2 than having a beacon-sensor send its information to the gate.

A valid entity will return true in an if-statement. This is helpful for preventing LUA errors resulting from using entity commands on entities which have been destroyed.

Commands

The only operators available for entities are equal and not equal. In addition, if(Entity) will return true only if there is a valid entity.

Function	Returns	Description
entity(N)		Gets the entity associated with the ID. If no ID is specified, this gets the entity of the expression chip in which this line is written.
owner()		Gets the owner of the expression ( same as entity():owner() )
E:id()		Gets the numeric id of an entity
noentity()		Returns an invalid entity
E[S,<type>]=X		Saves the value to the entity with the specified index string
E[S,type]=X	<type>	Retrieves the requested data from the string
E:type()		Gets the class of an entity
E:model()		Gets the model of an entity
E:owner()		Gets the owner of an entity
E:name()		Gets the name of a player
E:steamID()		Gets the steam ID of the player
E:isSteamFriend(E)		Returns if the given Entity is a steam friend of the first Entity
E:steamFriends()		Returns an array with E's steam friends on the server E is playing on
players()		Returns an array containing all players on the server
playersAdmins()		Returns an array containing all admins on the server
playersSuperAdmins()		Returns an array containing all super admins on the server
E:pos()		Gets the position of the entity
E:eye()		Gets a players view direction else entity forward direction
E:eyeAngles()		Gets a players view direction
E:eyeTrace()		Performs a quick trace from the player's eye. Equivalent to rangerOffset(16384, E:shootPos(), E:eye()), but faster. Does not respect filters or ranger flags.
E:shootPos()		Returns a players shoot position
E:aimEntity()		Returns the entity that the entity is aiming at
E:aimBone()		Returns the bone the player is currently aiming at
E:aimPos()		Returns the point that the entity is looking at
E:aimNormal()		Returns a normalized directional vector perpendicular to the surface pointed at
E:frags()		Returns the number of kills the player has made
E:deaths()		Returns the number of times the player died
E:team()		Returns the team number a player is on
N:teamName()		Returns the name of the team associated with the team number
teamColor(N)		Returns the color of the team associated with the team number

E:forward()	V	Gets the forward direction of the entity <sup>2)</sup>
E:right()	V	Gets the right direction of the entity
E:up()	V	Gets the up direction of the entity
E:vel()	V	Gets the velocity of the entity
E:velL()	V	Gets the local velocity of the entity
E:boxCenter()	V	Gets the center of the entity's bounding box, as a local position vector
E:boxCenterW()	V	Same as E:toWorld(E:boxCenter()), but is slightly faster because Lua does the toWorld for you (This convenience function was added because you tend to use toWorld(boxCenter()) a lot when using applyForce & co.)
E:boxMax()	V	Gets the maximum local XYZ of the entity's bounding box (the "highest" corner), as a local position vector
E:boxMin()	V	Gets the minimum local XYZ of the entity's bounding box (the "lowest" corner), as a local position vector
E:boxSize()	V	Gets the dimensions of the entity's bounding box as a vector (length, width, height)
E:nearestPoint(V)	V	Returns the closest point on the edge of the entity's bounding box to the given vector.
E:aabbMax()	V	Gets the maximum local position of the entity's axis aligned bounding box.
E:aabbMin()	V	Gets the minimum local position of the entity's axis aligned bounding box.
E:aabbSize()	V	Gets the size of the entity's axis aligned bounding box.
E:toWorld(V)	V	Transforms from a vector local to <i>E</i> to a world vector.
E:toLocal(V)	V	Transforms from a world vector to a vector local to <i>E</i> .
E:toWorld(A)	A	Transforms from an angle local to <i>E</i> to a world angle.
E:toLocal(A)	A	Transforms from a world angle to an angle local to <i>E</i> .
E:toWorldAxis(V)	V	Transforms an axis local to <i>E</i> to a global axis.
E:toLocalAxis(V)	V	Transforms a world axis to an axis local to <i>E</i> .
E:angVel()	A	Gets the angular velocity of the entity
E:angVelVector()	V	Returns local rotation axis, velocity and direction given as the vector's direction, magnitude and sense
E:angles()	A	Gets the pitch, yaw and roll of the entity
E:radius()	N	Gets the size of the object (not precisely, but useful)
E:height()	N	Gets the height of a player or npc
E:bearing(V)	N	Gets the bearing from the entity to the vector
E:elevation(V)	N	Gets the elevation from the entity to the vector
E:heading(V)	A	Gets the elevation and bearing from the entity to the vector
E:health()	N	Gets the health of the entity
E:armor()	N	Gets the armor of the player
E:volume()	N	Gets the volume of the entity
E:mass()	N	Gets the mass of the entity
E:timeConnected()	N	Returns a players time connected to a server
E:massCenter()	V	Gets the Center of Mass of the entity
E:massCenterL()	V	Gets the center of mass as a local vector
E:setMass(N)		Sets the mass of the entity (between 0.001 and 50,000)
E:inertia()	V	Gets the principal components of the entity's inertia tensor in the form ( <i>I<sub>xx</sub></i> , <i>I<sub>yy</sub></i> , <i>I<sub>zz</sub></i> )
E:elasticity()	N	Gets the elasticity (bounciness) of the entity
E:friction()	N	Gets the friction of the entity
E:applyForce(V)		Applies force to the entity according to the given vector's direction and magnitude
E:applyOffsetForce(V,V)		Applies force to the entity according to the first vector from the location of the second

E:applyAngForce(A)		Applies torque to the entity according to the given local angle
E:applyTorque(V)		Applies torque according to the given local vector, representing the torque axis, magnitude and direction
E:isPlayer()	N	Is the entity a player?
E:isOnFire()	N	Is the entity on fire?
E:isWeapon()	N	Is the entity a weapon?
E:isNPC()	N	Is the entity a NPC?
E:isFrozen()	N	Is the entity frozen?
E:isVehicle()	N	Is the entity a vehicle?
E:inVehicle()	N	Is the player in a vehicle?
E:isWorld()	N	Is the entity the world?
E:isOnGround()	N	Is the player/NPC resting on something?
E:isUnderWater()	N	Is the entity under water?
E:isPlayerHolding()	N	Is the entity being held by a player?
E:isAdmin()	N	Is the player an admin?
E:isSuperAdmin()	N	Is the player a super admin?
E:isAlive()	N	Is the player or NPC alive?
E:isCrouch()	N	Is the player crouching?
E:inNoclip()	N	Is the player in noclip mode?
E:friends()	R	Returns an array of players on the prop protection friends list.
E:trusts(E2)	N	Is E2 on the prop protection friends list of E?
E:keyAttack1()	N	Is the player pressing their primary fire key?
E:keyAttack2()	N	Is the player pressing their secondary fire key?
E:keyUse()	N	Is the player pressing their use key?
E:keyReload()	N	Is the player pressing their reload key?
E:keyZoom()	N	Is the player pressing their zoom key?
E:keyWalk()	N	Is the player pressing their walk key?
E:keySprint()	N	Is the player pressing their sprint key?
E:keyDuck()	N	Is the player pressing their duck key?
E:keyTurnLeft()	N	Is the player pressing their turn left key?
E:keyTurnRight()	N	Is the player pressing their turn right key?
E:keyPressed( S )	N	Is the player pressing the specified key? List of keys here: <a href="http://wiki.garrysmod.com/page/Enums/KEY">http://wiki.garrysmod.com/page/Enums/KEY</a> You don't need to write "KEY_", and they're case insensitive.
E:isTyping()	N	Is the player typing a message in chat?
E:driver()	E	Returns the driver of the vehicle if there is one, nil otherwise
E:passenger()	E	Returns the passenger of the vehicle if there is one, in single seat pods this will return the driver.
E:vehicle()	E	Returns the entity of the vehicle that the specified player is in
E:lockPod(N)		1 locks and 0 unlocks vehicle
E:ejectPod()		Ejects player in vehicle
E:killPod()		Kills player in vehicle
E:weapon()	E	Returns the weapon that player E is currently holding
E:clip1()	N	Returns the amount of ammo in the primary clip of weapon E, -1 if there is no primary clip
E:clip2()	N	Returns the amount of ammo in the secondary clip of weapon E, -1 if there is no secondary clip <sup>1)</sup>



E:primaryAmmoType()	<div>S</div>	Returns the type of primary ammo of weapon E as a number in a string
E:secondaryAmmoType()	<div>S</div>	Returns the type of secondary ammo of weapon E as number in a string
E:ammoCount(S)	<div>N</div>	Returns the amount of stored ammo of type S on player E, excluding current clip
E:tool()	<div>S</div>	returns the name of the tool the player E is currently holding
E:removeTrails()		Removes the trail from <i>E</i>
E:setTrails(N,N,N,S,V,N)		StartSize, EndSize, Length, Material, Color (RGB), Alpha Adds a trail to <i>E</i> with the specified attributes.
E:setTrails(N,N,N,S,V,N,N,N)		StartSize, EndSize, Length, Material, Color (RGB), Alpha, AttachmentID, Additive Adds a trail to <i>E</i> with the specified attributes.
E:lookupAttachment(string attachmentName)	<div>N</div>	Returns <i>E</i> 's attachment ID associated with <i>attachmentName</i>
E:attachmentPos(attachmentID)	<div>V</div>	Returns <i>E</i> 's attachment position associated with <i>attachmentID</i>
E:attachmentAng(attachmentID)	<div>A</div>	Returns <i>E</i> 's attachment angle associated with <i>attachmentID</i>
E:attachmentPos(string attachmentName)	<div>V</div>	Same as <i>E:attachmentPos(E:lookupAttachment(attachmentName))</i>
E:attachmentAng(string attachmentName)	<div>A</div>	Same as <i>E:attachmentAng(E:lookupAttachment(attachmentName))</i>

- 1) This is not the stored amount, no known weapon has a secondary clip, the AR2 and smg only have a storage, not a clip
- 2) For **most** valid entities, E:

<lua>E:toLocal(E:pos()+E:forward()) == vec(1, 0, 0)

E:toLocal(E:pos()+E:right()) == vec(0,-1, 0)

E:toLocal(E:pos()+E:up()) == vec(0, 0, 1)</lua>

Some entities (vehicles for instance) have differing axes.

Attachment documentation at <http://www.wiremod.com/forum/wiremod-tutorials/14813-e2-entity-attachment-documentation.html>

## Vector

### Description

Vectors are now properly implemented in the Expression 2, which means that they are as easy to work with as numbers. For those that know what vectors are, and how to use them, this is a great tool for creating many things.

2D and 4D vectors are also supported by E2. These include all the standard functions of 3D vectors listed here. If you're doing 2D vector operations, you can now do things much more efficiently. 4D vectors work in conjunction with matrices, and can be used as homogeneous representations of 3D vectors.

Operational functions can be used between numbers and vectors, e.g. N\*V. Note that operations cannot be performed between two vectors of different size, for example multiplication between a 2D and a 3D vector.

### 2D Vector Commands

Functions specific to 2D vectors

Function	Returns	Description
vec2(N,N)	<div>V2</div>	Makes a 2D vector
vec2()	<div>V2</div>	Same as vec2(0,0)
vec2(V)	<div>V2</div>	Converts a 3D vector into a 2D vector (the z component is dropped)

vec2(V4)	<div>V2</div>	Converts a 4D vector into a 2D vector (the z and w components are dropped)
V2:cross(V2)	<div>N</div>	Gets the 2D vector cross product/wedge product
shift(V2)	<div>V2</div>	Swaps the vector's x,y components
V2:rotate(N)	<div>V2</div>	Rotates a vector by the argument (given in degrees)
V2:toAngle()	<div>N</div>	Returns the 2D angle of the vector (given in degrees, -180 to 180)
V:dehomogenized()	<div>V2</div>	Converts a 2D homogeneous vector (x,y,w) into a 2D cartesian vector

3D Vector Commands

Functions specific to 3D vectors

Function	Returns	Description
vec(N,N,N)	<div>V</div>	Makes a 3D vector
vec()	<div>V</div>	Same as vec(0,0,0)
vec(V2)	<div>V</div>	Converts a 2D vector into a 3D vector (the z component is set to 0)
vec(V2,N)	<div>V</div>	Converts a 2D vector into a 3D vector (the z component is set to the second argument)
vec(V4)	<div>V</div>	Converts a 4D vector into a 3D vector (the w component is dropped)
vec(A)	<div>V</div>	Changes an angle variable into a vector variable
randvec()	<div>V</div>	Returns a uniformly distributed, random, normalized direction vector.
randvec(N <sub>1</sub> ,N <sub>2</sub> )	<div>V</div>	Returns a random vector with its components between $N_1$ and $N_2$
randvec(V <sub>1</sub> ,V <sub>2</sub> )	<div>V</div>	Returns a random vector between $V_1$ and $V_2$
V:cross(V)	<div>V</div>	Gets the 3D vector cross product
shiftL(V)	<div>V</div>	Shifts the vector's components left: shiftL( x,y,z ) = ( y,z,x )
shiftR(V)	<div>V</div>	Shifts the vector's components right: shiftR( x,y,z ) = ( z,x,y )
V:rotate(A)	<div>V</div>	Gets the rotated vector
V:rotate(N,N,N)	<div>V</div>	Gets the rotated vector
V:toAngle()	<div>A</div>	Gets the angles of the vector
V4:dehomogenized()	<div>V</div>	Converts a 3D homogeneous vector (x,y,z,w) into a 3D cartesian vector
pointHasContent( V, S )	<div>N</div>	'S' can be a string containing the last half of the CONTENTS_ enums (ie without the "CONTENTS_"). Multiple CONTENTS types can be seperated by a comma. Check: <a href="#">Enumeration List:Contents</a> for a full list. Examples: "water,solid" or "empty,transparent". The function returns 1 if any one of the types are found in the vector point.
pointContents( V )	<div>S</div>	Returns a string with all the "content" types in the vector point, seperated by commas.
V:isInWorld()	<div>N</div>	Returns 1 if the position vector is within the world, 0 if not
toWorld(V,A,V,A)	<div>V</div>	These six toWorld and toLocal functions work like GLua's <a href="#">LocalToWorld</a> and <a href="#">WorldToLocal</a> , but return different things.
toWorldAng(V,A,V,A)	<div>A</div>	
toWorldPosAng(V,A,V,A)	<div>R</div>	
toLocal(V,A,V,A)	<div>V</div>	
toLocalAng(V,A,V,A)	<div>A</div>	
toLocalPosAng(V,A,V,A)	<div>R</div>	
bearing(V,A,V)	<div>N</div>	These functions work like e:bearing(v), e:elevation(v), but don't require an entity.
elevation(V,A,V)	<div>N</div>	Gets the bearing from the first position, at the specified angle, to the second position.
		Gets the elevation from the first position, at the specified angle, to the second position.

heading(V,A,V)

A

Gets the bearing and elevation from the first position, at the specified angle, to the second position.

4D Vector Commands

Functions specific to 4D vectors. From a mathematics standpoint these are treated as 4D Cartesian vectors, where the 4th component is referred to as "w".

Function	Returns	Description
vec4(N,N,N,N)	<div>V4</div>	Makes a 4D vector
vec4()	<div>V4</div>	Same as vec4(0,0,0,0)
vec4(V2)	<div>V4</div>	Converts a 2D vector into a 4D vector (the z and w components are set to 0)
vec4(V2,N,N)	<div>V4</div>	Converts a 2D vector into a 4D vector (the z and w components are set to the second and third arguments)
vec4(V2,V2)	<div>V4</div>	Creates a 4D vector from two 2D vectors
vec4(V)	<div>V4</div>	Converts a 3D vector into a 4D vector (the w component is set to 0)
vec4(V,N)	<div>V4</div>	Converts a 3D vector into a 4D vector (the w component is set to the second argument)
shiftL(V4)	<div>V4</div>	Shifts the vector's components left: shiftL( x,y,z,w ) = ( y,z,w,x )
shiftR(V4)	<div>V4</div>	Shifts the vector's components right: shiftR( x,y,z,w ) = ( w,x,y,z )

Common Vector Commands

Functions that apply to 2D and 3D vectors. They are written here in terms of 3D vectors, but apply to 2D and 4D vectors in the same way, also returning 2D or 4D vectors where applicable.

Function	Returns	Description
ceil(V)	<div>V</div>	Rounds XYZ up to the nearest integer
ceil(V,N)	<div>V</div>	Rounds XYZ up to argument 2's decimal precision
floor(V)	<div>V</div>	Rounds XYZ down to the nearest integer
floor(V,N)	<div>V</div>	Rounds XYZ down to argument 2's decimal precision
round(V)	<div>V</div>	Rounds XYZ to the nearest integer
round(V,N)	<div>V</div>	Rounds XYZ to argument 2's decimal precision
mod(V,N)	<div>V</div>	Returns the remainder after XYZ have been divided by argument 2
mod(V,V)	<div>V</div>	Returns the remainder after the components of vector 1 have been divided by the components of vector 2
clamp(V,V,V)	<div>V</div>	Clamps vector 1's XYZ between the XYZ of vector 2(min) and vector 3(max)
clamp(V,N,N)	<div>V</div>	Returns a vector in the same direction as vector 1, with length clamped between argument 2(min) and argument 3(max)
min(V,V)	<div>V</div>	Returns the vector with the smallest length
max(V,V)	<div>V</div>	Returns the vector with the greatest length
minVec(V,V)	<div>V</div>	Returns a vector combining the lowest value components of V1 and V2
maxVec(V,V)	<div>V</div>	Returns the vector combining the highest value components of V1 and V2
mix(V,V,N)	<div>V</div>	Returns the point N percent between the 2 given points. (N is equal to or between 0 and 1) Ex, mix(V1, V2, 0.5) will return the point in the middle of points V1 and V2.
positive(V)	<div>V</div>	Returns a vector containing the positive value of each vector component, equivalent to abs(N)
inrange(V,V <sub>min</sub> ,V <sub>max</sub> )	<div>V</div>	Returns 1 if each component of V is between (or is equal to) the components of V <sub>min</sub> and V <sub>max</sub>
toRad(V)	<div>V</div>	Converts the vector's magnitude from degrees to radians
toDeg(V)	<div>V</div>	Converts the vector's magnitude from radians to degrees
V:length()	<div>N</div>	Gets the length of the vector
V:length2()	<div>N</div>	Gets the squared length of the vector



V:distance(V)	N	Gets the distance between vectors
V:distance2(V)	N	Gets the squared distance between vectors
V:normalized()	V	Gets the normalized vector
V:dot(V)	N	Gets the vector dot (scalar) product
V:outerProduct(V)	M	Gets the outer product (tensor product) and returns a matrix (tensor)
V:x()	N	Gets the x component of the vector
V:y()	N	Gets the y component of the vector
V:z()	N	Gets the z component of the vector
V:w()	N	Gets the w component of the vector
V:setX(N)	V	Returns a copy of the vector with X replaced (use as Vec = Vec:setX(...))
V:setY(N)	V	Returns a copy of the vector with Y replaced (use as Vec = Vec:setY(...))
V:setZ(N)	V	Returns a copy of the vector with Z replaced (use as Vec = Vec:setZ(...))
V:setW(N)	V	Returns a copy of the vector with W replaced (use as Vec = Vec:setW(...))
V:toString()	S	Gets the vector nicely formatted as a string "[X,Y,Z]"

Matrix

Developed by: Jimlad

Description

2x2, 3x3 and 4x4 matrices are now supported in Expression 2. These are for more advanced manipulations involving vectors and numbers. As with vectors, for those with the relevant knowledge these can be very useful tools.

Basic operations supported:

- Matrix addition and subtraction
- Multiplication by scalars, vectors and matrices
- Division by a scalar
- Exponentiation (only integers between -1 and 2)
- Delta of a matrix (returns a matrix)

NOTES:

Similarly to vectors, 3x3 matrix commands are referred to using "**matrix**", whereas 2x2 and 4x4 matrix commands use "**matrix2**" and "**matrix4**"

The "set" and "swap" functions are like the 3D vector "set" functions; they *do not* affect the original matrix.

Remember that operations will only work on vectors/matrices of a similar size. You cannot, for example, multiply a 3x3 matrix by a 2D vector. Also, all vectors are treated as column vectors for the purposes of matrices, so M\*V will return a vector but V\*M is undefined.


























2x2 Matrix Commands

Functions specific to 2x2 matrices

2x2 Matrix Commands











Functions specific to 2x2 matrices













Function	Returns	Description
----------	---------	-------------

identity2()		Creates a 2x2 identity matrix
matrix2()		Creates a 2x2 zero matrix
matrix2(N,N,N,N)		Creates a matrix with values in order (i.j) of: (1,1), (1,2), (2,1), (2,2)
matrix2(V2,V2)		Creates a matrix with vectors by columns
rowMatrix2(V2,V2)		Creates a matrix with vectors by rows
matrix2(M)		Converts a 3x3 matrix into a 2x2 matrix - all (i,3) and (3,j) are omitted
matrix2(M4)		Converts a 4x4 matrix into a 2x2 matrix - all (i,3), (i,4), (3,j) and (4,j) are omitted
M2:swapRows()		Swaps rows
M2:swapColumns()		Swaps columns
M2:setRow(N,N,N)		Sets the values of a row. The first argument given specifies the row(j), the following arguments are the values 1j, 2j
M2:setRow(N,V2)		Sets the values of a row. The first argument given specifies the row, the vector contains the values to set
M2:setColumn(N,N,N)		Sets the values of a column. The first argument given specifies the column(i), the following arguments are the values i1, i2
M2:setColumn(N,V2)		Sets the values of a column. The first argument given specifies the column, the vector contains the values to set
Function	Returns	Description
identity2()		Creates a 2x2 identity matrix
matrix2()		Creates a 2x2 zero matrix
matrix2(N,N,N,N)		Creates a matrix with values in order (i.j) of: (1,1), (1,2), (2,1), (2,2)
matrix2(V2,V2)		Creates a matrix with vectors by columns
matrix2(M)		Converts a 3x3 matrix into a 2x2 matrix - all (i,3) and (3,j) are omitted
matrix2(M4)		Converts a 4x4 matrix into a 2x2 matrix - all (i,3), (i,4), (3,j) and (4,j) are omitted
M2:swapRows()		Swaps rows
M2:swapColumns()		Swaps columns
M2:setRow(N,N,N)		Sets the values of a row. The first argument given specifies the row(j), the following arguments are the values 1j, 2j
M2:setRow(N,V2)		Sets the values of a row. The first argument given specifies the row, the vector contains the values to set
M2:setColumn(N,N,N)		Sets the values of a column. The first argument given specifies the column(i), the following arguments are the values i1, i2
M2:setColumn(N,V2)		Sets the values of a column. The first argument given specifies the column, the vector contains the values to set

3x3 Matrix Commands

Functions specific to 3x3 matrices























Function	Returns	Description
identity()		Creates a 3x3 identity matrix
matrix()		Creates a 3x3 zero matrix
matrix(N <sub>1</sub> ,N <sub>2</sub> ... N <sub>9</sub> )		Creates a matrix with 9 values in the following order (i.j): (1,1), (1,2), (1,3), (2,1) etc.
matrix(V,V,V)		Creates a matrix with vectors by columns
rowMatrix(V,V,V)		Creates a matrix with vectors by rows
matrix(M2)		Converts a 2x2 matrix into a 3x3 matrix - all (i,3) and (3,j) are filled with 0's
matrix(M4)		Converts a 4x4 matrix into a 3x3 matrix - all (i,4) and (4,j) are omitted
M:swapRows(N,N)		Swaps the two rows specified
M:swapColumns(N,N)		Swaps the two columns specified
M:setRow(N,N,N,N)		Sets the values of a row. The first argument given specifies the row(j), the following arguments are the values 1j, 2j, 3j



M:setRow(N,V)		Sets the values of a row. The first argument given specifies the row, the vector contains the values to set
M:setColumn(N,N,N,N)		Sets the values of a column. The first argument given specifies the column(i), the following arguments are the values i1, i2, i3
M:setColumn(N,V)		Sets the values of a column. The first argument given specifies the column, the vector contains the values to set
M:setDiagonal(N,N,N)		Sets the elements of the leading diagonal
M:setDiagonal(V)		Sets the elements of the leading diagonal from the components of a vector
matrix(E)		Creates a reference frame matrix from an entity's local direction vectors by columns in the order ( x, y, z )
matrix(A)		Returns a 3x3 reference frame matrix as described by the angle <i>A</i> . Multiplying by this matrix will be the same as rotating by the given angle.
M:toAngle()		Returns an angle derived from a 3x3 rotation matrix
M:x()		Returns the local x direction vector from a 3x3 coordinate reference frame matrix ( same as M:column(1) )
M:y()		Returns the local y direction vector from a 3x3 coordinate reference frame matrix ( same as M:column(2) )
M:z()		Returns the local z direction vector from a 3x3 coordinate reference frame matrix ( same as M:column(3) )
mRotation(V,N)		Creates a 3x3 rotation matrix, where the vector is the axis of rotation, and the number is the angle (anti-clockwise) in degrees. Example*: to rotate a vector (7,8,9) by 50 degrees about the axis (1,1,0), you would write V = mRotation(vec(1,1,0), 50) * vec(7,8,9)

\* If you want to create a rotation matrix about the axes (1,0,0), (0,1,0) or (0,0,1), either use the V:rotate function, or construct a standard [rotation matrix](#).

4x4 Matrix Commands

Functions specific to 4x4 matrices











Function	Returns	Description
identity4()		Creates a 4x4 identity matrix
matrix4()		Creates a 4x4 zero matrix
matrix4(N <sub>1</sub> ,N <sub>2</sub> ... N <sub>16</sub> )		Creates a matrix with 16 values in the following order (i,j): (1,1), (1,2), (1,3), (1,4), (2,1) etc.
matrix4(V4,V4,V4,V4)		Creates a matrix with vectors by columns
rowMatrix4(V4,V4,V4,V4)		Creates a matrix with vectors by rows
matrix4(M2)		Converts a 2x2 matrix into a 4x4 matrix - all (i,3), (i,4), (3,j) and (4,j) are filled with 0's
matrix4(M2,M2,M2,M2)		Constructs a 4x4 matrix from four 2x2 matrices
matrix4(M)		Converts a 3x3 matrix into a 4x4 matrix - all (i,4) and (4,j) are filled with 0's
M4:swapRows(N,N)		Swaps the two rows specified
M4:swapColumns(N,N)		Swaps the two columns specified
M4:setRow(N,N,N,N,N)		Sets the values of a row. The first argument given specifies the row(j), the following arguments are the values 1j, 2j, 3j, 4j
M4:setRow(N,V4)		Sets the values of a row. The first argument given specifies the row, the vector contains the values to set
M4:setColumn(N,N,N,N,N)		Sets the values of a column. The first argument given specifies the column(i), the following arguments are the values i1, i2, i3, i4
M4:setColumn(N,V4)		Sets the values of a column. The first argument given specifies the column, the vector contains the values to set
M4:setDiagonal(N,N,N,N)		Sets the elements of the leading diagonal
M4:setDiagonal(V4)		Sets the elements of the leading diagonal from the components of a vector
matrix4(E)		Creates a 4x4 reference frame matrix from an entity's local direction vectors by columns in the order (x, y, z, pos), with the bottom row (0,0,0,1)
matrix4(A)		Returns a 4x4 reference frame matrix as described by the angle <i>A</i> . Multiplying by this matrix will be the same as rotating by the given angle.
matrix4(A,V)		Returns a 4x4 reference frame matrix as described by the angle <i>A</i> and the position <i>V</i> . Multiplying by this matrix will be the same as rotating by the given angle and offsetting by the given vector.
M4:x()		Returns the local x direction vector from a 4x4 coordinate reference frame matrix
M4:y()		Returns the local y direction vector from a 4x4 coordinate reference frame matrix
M4:z()		Returns the local z direction vector from a 4x4 coordinate reference frame matrix

M4:pos()		Returns the position vector from a 4x4 coordinate reference frame matrix
inverseA(M4)		Finds the matrix inverse of a standard 4x4 affine transformation matrix ( the type created by matrix4(E) ). This should only be used on matrices with a particular format, where the top left 3x3 specifies rotation, the rightmost 3-column specifies translation, and the bottom row is (0,0,0,1)

Common Matrix Commands

Functions that apply to 2x2, 3x3 and 4x4 matrices. They are written here in terms of 3x3 matrices, but apply to 2x2's and 4x4's in the same way.

Operations will only return vectors/matrices of similar sizes. For example, the row() function on a 2x2 matrix will return a 2D vector

Function	Returns	Description
M:row(N)		Returns the row as a vector
M:column(N)		Returns the column as a vector
M:element(N,N)		Returns the element with indices (i,j)
M:setElement(N,N,N)		Sets an element's value. The first two arguments specify the indices (i,j), the third argument is the value to set it to
M:swapElements(N,N,N,N)		Swaps two elements, specified by indices ( i <sub>1</sub> , j <sub>1</sub> , i <sub>2</sub> , j <sub>2</sub> )
diagonal(M)		Returns a vector comprising the elements along the leading diagonal
trace(M)		Returns the trace of a matrix
det(M)		Returns the determinant of a matrix (Does not work for 4x4 matrices)
transpose(M)		Returns the transpose of a matrix
adj(M)		Returns the adjugate of a matrix (Does not work for 4x4 matrices)












*NOTE:* To get the inverse of a matrix, simply raise the matrix to the power of -1. Use this sparingly as it can be computationally **expensive**! Remember that if your matrix is orthogonal (e.g. rotation matrices), the inverse is equal to the transpose, so use the transpose instead if you can. Inverse is not available for 4x4 matrices. Instead, see usage of the inverseA(M4) function.

Angle

Description

Like 3 different directions can be expressed as a Vector, the angles of Pitch, Yaw and Roll can be expressed as an angle Vector. This in the least has the advantage that when performing functions which use angles, such as vector rotation or creating vectors from angles, you don't have to write the Pitch, Yaw and Roll components, only the Angle.

Commands

Function	Returns	Description
ang(N,N,N)		Makes an angle
ang()		Same as ang(0,0,0)
ang(V)		Changes a vector variable into an angle variable
ceil(A)		Rounds PYR up to the nearest integer
ceil(A,N)		Rounds PYR up to argument 2's decimal precision
floor(A)		Rounds PYR down to the nearest integer
floor(A,N)		Rounds PYR down to argument 2's decimal precision
round(A)		Rounds PYR to the nearest integer
round(A,N)		Rounds PYR to argument 2's decimal precision
mod(A,N)		Returns the remainder after PYR have been divided by argument 2
mod(A,A)		Returns the remainder after the components of angle 1 have been divided by the components of angle 2

clamp(A,A,A)	<div>A</div>	Clamps angle 1's PYR between the PYR of angle 2(min) and angle 3(max)
clamp(A,N,N)	<div>A</div>	Clamps angle 1's PYR between argument 2(min) and argument 3(max)
mix(A,A,N)	<div>A</div>	Combines angle 1's PYR with angle 2's PYR by a proportion given by argument 3 (between 0 and 1)
shiftL(A)	<div>A</div>	Shifts the angle's components left: shiftL( p,y,r ) = ( y,r,p )
shiftR(A)	<div>A</div>	Shifts the angle's components right: shiftR( p,y,r ) = ( r,p,y )
inrange(A,A <sub>min</sub> ,A <sub>max</sub> )	<div>N</div>	Returns 1 if each component of A is between (or is equal to) the components of A <sub>min</sub> and A <sub>max</sub>
toRad(A)	<div>A</div>	Converts the angle's magnitude from degrees to radians
toDeg(A)	<div>A</div>	Converts the angle's magnitude from radians to degrees
angnorm(A)	<div>A</div>	Gets the normalized angle of an angle
angnorm(N)	<div>N</div>	Gets the normalized angle of a number
A:pitch()	<div>N</div>	Gets the pitch of the angle
A:yaw()	<div>N</div>	Gets the yaw of the angle
A:roll()	<div>N</div>	Gets the roll of the angle
A:setPitch(N)	<div>A</div>	Returns a copy of the angle with Pitch replaced (use as Ang = Ang:setPitch(...))
A:setYaw(N)	<div>A</div>	Returns a copy of the angle with Yaw replaced (use as Ang = Ang:setYaw(...))
A:setRoll(N)	<div>A</div>	Returns a copy of the angle with Roll replaced (use as Ang = Ang:setRoll(...))
A:toString()	<div>S</div>	Gets the angle nicely formatted as a string "[P,Y,R]"
A:forward()	<div>V</div>	Gets the forward vector of the <b>angle</b> (This can also be known as the direction vector).
A:right()	<div>V</div>	Gets the right vector of the <b>angle</b> .
A:up()	<div>V</div>	Gets the up vector of the <b>angle</b> .
A:rotateAroundAxis(V,N)	<div>A</div>	Returns the angle A rotated around vector V by N degrees.

Table

Description

Tables are a way to store variables. You can think of a table as a list of data, where each data is addressed with either a number or a string. Tables can contain any datatype, including other tables. There is, however, a max table depth (default 6)

Assigning one table variable to equal another will make them both refer to the same table. If you want to make a new copy of a table which will thereafter be set and retrieved from independently of the original table, you must use clone().

Related Examples

- [Table-RAM by Magos Mechanicus](#)

Commands

In the interest of brevity, some commands which have many variants are shown as a pattern. <type> may be substituted with the capitalized name of any supported datatype, and \* is the corresponding datatype symbol. For instance, T:push<type>(\*) can mean T:pushNumber(N), or T:pushString(S).

Function	Returns	Description
table(...)	<div>T</div>	Creates a table with the specified variables (the variables will have numerical indexes).
<b>Common</b> T:clear()		These functions affect variables indexed by both strings and numbers. Clears the table.



T:count()	<div>N</div>	Returns the number of variables in the table.
invert(T)	<div>T</div>	Inverts the table, creating a lookup table.
T:flip()	<div>T</div>	'Flips' the strings and numbers of the table (ignores other variables).
T:typeids()	<div>T</div>	Returns a table with the type IDs of the variables in the table.
T:remove(S/N)		Removes the variable with the specified index. If the key is a number, all sequential keys will be moved down to fill the gap.
T:remove*(S/N)	*	Removes the variable at the specified index, with the specified type, and returns it. If the key is a number, all sequential keys will be moved down to fill the gap.
T:clipToTypeid(S)	<div>T</div>	Removes all variables not of the specified type.
T:clipFromTypeid(S)	<div>T</div>	Removes all variables of the specified type.
T:clone()	<div>T</div>	Returns an independent copy of the table.
T:id()	<div>S</div>	Returns a unique identifier for the table. Useful for invert(T).
T:toString()	<div>S</div>	Returns the table as a human readable string.
T:add(T2)	<div>T</div>	Returns T with the contents of T2 added to the end. Pushes numerical indexes to the end of T, and only writes string indexes which don't exist on T.
T:merge(T2)	<div>T</div>	Merges T2 with T. Any variables with the same indexes are overwritten by T2's variables.
T:difference(T2)	<div>T</div>	Removes all variables with keys that exist in T2.
T:intersect(T2)	<div>T</div>	Removes all variables with keys which don't exist in T2.
T[S/N,type]	*	Retrieves the variable from the table at the specified index and with the specified type.
T[S/N,type] = X	*	Saves the variable in the table at the specified index and with the specified type.
<b>Number indexes</b>		
T:pop()		Removes the last variable.
T:pop*()	*	Removes and returns the last variable.
T:min()	<div>N</div>	Returns the smallest number.
T:max()	<div>N</div>	Returns the largest number.
T:minIndex()	<div>N</div>	Returns the index of the smallest number.
T:maxIndex()	<div>N</div>	Returns the index of the largest number.
T:typeidsArray()	<div>R</div>	Returns an array with the IDs of the variables with numerical indexes.
T:toArray()	<div>R</div>	Converts the table into an array (discards all string indexes, tables, and arrays).
T:concat()	<div>S</div>	Concatenates all values and returns the result. Behavior is undefined for vectors/angles/matrices and similar types.
T:concat(S)	<div>S</div>	Concatenates all values with the specified string in between each, and returns the result.
T:concat(N)	<div>S</div>	Concatenates all values, starting at index nr N, and returns the result.
T:concat(S,N)	<div>S</div>	Concatenates all values, starting at index nr N, and with string S in between each.
T:concat(N,N)	<div>S</div>	Concatenates all values, starting at index N1 and ending at N2.
T:concat(S,N,N)	<div>S</div>	Concatenates all values, with the specified string in between each. Starts at index N1, and ends at N2.
T:push*()	*	Adds the variable to the end of the table.
T:insert*(N,*)	*	Inserts the variable at the specified position. Moves all other indexes up one step to compensate. Maximum index possible is 2^31, minimum is 0.
T:unshift*()	*	Adds the data to the beginning of the table. Will move all other entries up one step to compensate.
T:shift()		Deletes the first element of the table; all other entries will move down one address
T:exists(N)	<div>N</div>	Returns 1 if any variable exists at the index N, else returns 0.
<b>String indexes</b>		
invert(R)	<div>T</div>	Inverts the array, creating a lookup table.
T:keys()	<div>R</div>	Returns an array with the keys of the table.
T:values()	<div>R</div>	Returns an array with the values of the table (tables and arrays, which arrays do not support, are discarded).
T:exists(S)	<div>N</div>	Returns 1 if any variable exists at the index S, else returns 0.

## Array

Thanks to: Erkle

### Description

Same as table, but with much less memory footprint and is numerically indexed instead. It is similar to E1's packet support. Arrays can contain any datatype except table and array.

The index 0 and even negative and non-integer indices can be used, but to get the most out of the array functions it is advisable to start at index 1 and not to leave any gaps.




If you don't follow these guidelines, push/pop/count might misbehave.

Arrays are limited to 1048576 elements by default.

### Commands

In the interest of brevity, some commands which have many variants are shown as a pattern. <type> may be substituted with the capitalized name of any supported datatype, and \* is the corresponding datatype symbol. For instance, R:push<type>(\*) can mean R:pushNumber(N), or R:pushString(S).

Function	Returns	Description
array()	<div>R</div>	Creates an empty array
array(...)	<div>R</div>	Constructs an array with the given values as elements. If you specify types that are not supported by the array data type, the behaviour is undefined.  Example: R = array( 5, 8, 8, "string", vec( 85,47,10 ) )
R:clone()	<div>R</div>	Creates an independant copy of an array
R:count()	<div>N</div>	Returns the number of used indexes in the array
R:sum()	<div>N</div>	Adds all numbers in the array together and returns result
R:concat()	<div>S</div>	Concatenates all values in the array and returns the result. Behavior is undefined for vectors/angles/matrices and similar types.
R:concat(S)	<div>S</div>	Concatenates all values in the array with the specified string in between each, and returns the result.
R:concat(N)	<div>S</div>	Concatenates all values in the array, starting at index nr N, and returns the result.
R:concat(S,N)	<div>S</div>	Concatenates all values in the array, starting at index nr N, and with string S in between each.
R:concat(N,N)	<div>S</div>	Concatenates all values in the array, starting at index N1 and ending at N2.
R:concat(S,N,N)	<div>S</div>	Concatenates all values in the array, with the specified string in between each. Starts at index N1, and ends at N2.
R:average()	<div>N</div>	Gives the average of all numbers in array
R:min()	<div>N</div>	Returns the smallest number in array
R:minIndex()	<div>N</div>	Returns the index of the smallest number in array
R:max()	<div>N</div>	Returns the largest number in array
R:maxIndex()	<div>N</div>	Returns the index of the largest number in array
R:<type>(N)	*	Deprecated. Use R[N,<type>] instead.
R:set<type>(N,*)		Deprecated. Use R[N,<type>]=X instead.
R[N,<type>]	<type>	Retrieves the array element indexed with the number. Returns the default value for the datatype if the element is nil.
R[N,<type>]=*		Saves Something into the N index of the array. Works exactly like a variable or persist.
R:push<type>(*)	*	Saves the data at the end of the array
R:pop<type>()		Deletes and returns the last entry in the array. Be sure not to use popNumber() on a vector or similar, as the data may be lost
R:pop()		Deletes the last entry in the array
R:unshift<type>(*)		Adds the data to the beginning of the array. Will move all other entries up one address

R:shift<type>()	*	Deletes and returns the first element of the array, moving other entries down one address to compensate.
R:shift()		Deletes the first element of the array; all other entries will move down one address
R:insert<type>(N,*)	*	Inserts the data into the specified index; all entries after this index will move up to compensate. Maximum index possible is 2^31, minimum is 0.
R:remove<type>(N)	*	Deletes and returns the specified entry, moving subsequent entries down to compensate
R:remove(N)		Deletes the specified entry, moving subsequent entries down to compensate
R:exists(N)		Returns 1 if any variable exists at the index N, else returns 0.
R:add(R2)		Adds the contents of R2 to R.
R:merge(R2)		Merges R2 with R (identical indexes are overwritten).

Bone






















Developed by: TomyLobo

Description

This extension gives E2 support for **bone** entities. A **bone** can be any part of any ragdoll (head, left arm, right leg, etc). You can get a bone's position, orientation, velocity, etc, much like with regular props (although some things are missing).

Array and table functions for bones are also provided.

Commands

Function	Returns	Description
E:bone(N)		Returns <i>E</i> 's <i>N</i> th bone
E:bones()		Returns an array containing all of <i>E</i> 's bones. This array's first element has the index 0!
E:boneCount()		Returns <i>E</i> 's number of bones
nobone()		Returns an invalid bone
E:aimBone()		Returns the bone the player is currently aiming at
B:entity()		Returns the entity <i>B</i> belongs to
B:index()		Returns <i>B</i> 's index in the entity it belongs to. Returns -1 if the bone is invalid or an error occured
B:pos()		Returns <i>B</i> 's position
B:forward()		Returns a vector describing <i>B</i> 's forward direction
B:right()		Returns a vector describing <i>B</i> 's right direction
B:up()		Returns a vector describing <i>B</i> 's up direction
B:vel()		Returns <i>B</i> 's velocity
B:velL()		Returns <i>B</i> 's velocity in local coordinates
B:toWorld(V)		Transforms <i>V</i> from local coordinates (as seen from <i>B</i> ) to world coordinates
B:toLocal(V)		Transforms <i>V</i> from world coordinates to local coordinates (as seen from <i>B</i> )
B:angVel()		Returns <i>B</i> 's angular velocity
B:angles()		Returns <i>B</i> 's pitch, yaw and roll angles
B:bearing(V)		Returns the bearing (yaw) from <i>B</i> to <i>V</i>
B:elevation(V)		Returns the elevation (pitch) from <i>B</i> to <i>V</i>
B:mass()		Returns <i>B</i> 's mass
B:massCenter()		Returns <i>B</i> 's Center of Mass



B:massCenterL()	<div>V</div>	Returns <i>B</i> 's Center of Mass in local coordinates
B:setMass(N)		Sets <i>B</i> 's mass (between 0.001 and 50,000)
B:inertia()	<div>V</div>	Gets the principal components of <i>B</i> 's inertia tensor in the form vec(Ixx, Iyy, Izz)
B:applyForce(V)		Applies force to <i>B</i> according to <i>V</i> 's direction and magnitude
B:applyOffsetForce(V,V2)		Applies force to <i>B</i> according to <i>V</i> from the location of <i>V2</i>
B:applyAngForce(A)		Applies torque to <i>B</i> according to <i>A</i>
B:applyTorque(V)		Applies torque to <i>B</i> according to the given local vector <i>V</i> , representing the torque axis, magnitude and direction
B:isFrozen()	<div>N</div>	Returns 1 if <i>B</i> is frozen, 0 otherwise

## Wirelink

### Description

Wirelinks are an alternative to normal wires that offer a number of advantages. Any number of inputs or outputs on a component can be manipulated with one Wirelink, and you can also use it to retrieve the entity of a wirelinked component. Since all Wirelinks are capable of two-way communication, wirelinks are not clear-cut inputs or outputs. As such, to avoid ambiguity wirelinks which the expression should be able to manipulate are always declared in the @inputs of the expression. To connect this input to another component, you must use the Wirelink tool on the component to create a new output on it of the type Wirelink, then wire the input to the output as normal.

### Commands

Equal and Not Equal operators are available. XWL here means the Wirelink input.

Function	Returns	Description
XWL:isHiSpeed()	<div>N</div>	Returns true if the linked component is high-speed capable.
XWL:entity()	<div>E</div>	Returns the entity of the linked component.
XWL:hasInput(S)	<div>N</div>	Returns true if the linked component has an input of the specified name.
XWL:hasOutput(S)	<div>N</div>	Returns true if the linked component has an output of the specified name.
XWL[S,<type>]	<type>	Retrieves the component's output of the specified name.
XWL[S,<type>]=X		Sets the component's input of the specified name equal to X.
<del>XWL:setNumber(S,N)</del>		Deprecated. Use XWL[S,number]=X instead.
<del>XWL:number(S)</del>	<div>N</div>	Deprecated. Use XWL[S,number] instead.
<del>XWL:setVector(S,V)</del>		Deprecated. Use XWL[S,vector]=X instead.
<del>XWL:vector(S)</del>	<div>V</div>	Deprecated. Use XWL[S,vector] instead.
<del>XWL:setString(S,S)</del>		Deprecated. Use XWL[S,string]=X instead.
<del>XWL:string(S)</del>	<div>S</div>	Deprecated. Use XWL[S,string] instead.
XWL:setXyz(V)		Sets the X/Y/Z to the corresponding values in the vector.
XWL:xyz()	<div>V</div>	Retrieves the X/Y/Z as the corresponding values in the vector.
<del>XWL:setEntity(S,E)</del>		Deprecated. Use XWL[S,entity]=X instead.
<del>XWL:entity(S)</del>	<div>E</div>	Deprecated. Use XWL[S,entity] instead.
<del>XWL:writeCell(N,N)</del>	<div>N</div>	Deprecated. Use XWL[N]=X instead.
<del>XWL:readCell(N)</del>	<div>N</div>	Deprecated. Use XWL[N] instead.
XWL[N]	<div>N</div>	Returns contents of the specified memory cell.
XWL[N]=X		Writes the value to the memory cell specified by the index.
XWL:writeString(S,N,N)		A helper function for using the <a href="#">Wired Console Screen</a> . The string will be written to the screen in white text on black background. The number arguments specify the starting position - X/Horizontal (0-29 recommended) and Y/vertical (0-17).
XWL:writeString(S,N,N,N)		As above, with an extra argument for the text colour. This is in the form of a 3-digit RGB code. 0 is black, while 999 is white, 900 is pure red and so on.

XWL:writeString(S,N,N,N,N)		As above, with an extra argument for background colour. 3-digit RGB again.
XWL:writeString(S,N,N,N,N,N)		As above, with an extra argument for flashing text. 0 or 1 is recommended.
XWL:writeString(N,S)	<div>N</div>	Writes a null-terminated string to the given address. Returns the next free address or 0 on failure.
XWL:readString(N)	<div>S</div>	Reads a null-terminated string from the given address. Returns an empty string on failure.
XWL:writeArray(N,R)	<div>N</div>	Writes an array's elements into a piece of memory. Strings and sub-tables (angles, vectors, matrices) are written as pointers to the actual data. Strings are written null-terminated.
XWL:writeTable(N,T)	<div>N</div>	Same as writeArray, except it uses the numerically indexed variables of the table instead.
XWL:inputs()	<div>R</div>	Returns an array of all the inputs that <i>XWL</i> has without their types. Returns an empty array if it has none
XWL:outputs()	<div>R</div>	Returns an array of all the outputs that <i>XWL</i> has without their types. Returns an empty array if it has none
XWL:inputType(S)	<div>S</div>	Returns the type of input that <i>S</i> is in lowercase. ( "NORMAL" is changed to "number" )
XWL:outputType(S)	<div>S</div>	Returns the type of output that <i>S</i> is in lowercase. ( "NORMAL" is changed to "number" )

Complex


















Developed by: Fizyk

Description

Complex numbers are an extension of real numbers to include roots of negative numbers as well. They support all basic operations, like addition, subtraction, multiplication, division and raising to a power. Also operations with real numbers are supported, like N+C etc. There are comparison operators == and !=, no < and > though, as those are undefined for complex numbers.

Commands

Function	Returns	Description
comp()	<div>C</div>	Returns complex zero
comp(N)	<div>C</div>	Converts a real number to complex (returns complex number with real part <i>N</i> and imaginary part 0)
comp(N,N <sub>2</sub> )	<div>C</div>	Returns <i>N</i> + <i>N</i> <sub>2</sub> *i
i()	<div>C</div>	Returns the imaginary unit i
i(N)	<div>C</div>	Returns <i>N</i> *i
abs(C)	<div>C</div>	Returns the absolute value of <i>C</i>
arg(C)	<div>C</div>	Returns the argument of <i>C</i>
conj(C)	<div>C</div>	Returns the conjugate of <i>C</i>
real(C)	<div>N</div>	Returns the real part of <i>C</i>
imag(C)	<div>N</div>	Returns the imaginary part of <i>C</i>
exp(C)	<div>C</div>	Raises Euler's constant e to the power of <i>C</i>
log(C)	<div>C</div>	Calculates the natural logarithm of <i>C</i>
log(C,C <sub>2</sub> )	<div>C</div>	Calculates the logarithm of <i>C</i> <sub>2</sub> to a complex base <i>C</i>
log(N,C)	<div>C</div>	Calculates the logarithm of <i>C</i> to a real base <i>N</i>
log2(C)	<div>C</div>	Calculates the logarithm of <i>C</i> to base 2
log10(C)	<div>C</div>	Calculates the logarithm of <i>C</i> to base 10
sqrt(C)	<div>C</div>	Calculates the square root of <i>C</i>
csqrt(N)	<div>C</div>	Calculates the complex square root of the real number <i>N</i>
sin(C)	<div>C</div>	Calculates the sine of <i>C</i>

cos( <i>C</i> )		Calculates the cosine of <i>C</i>
tan( <i>C</i> )		Calculates the tangent of <i>C</i>
cot( <i>C</i> )		Calculates the cotangent of <i>C</i>
sec( <i>C</i> )		Calculates the secant of <i>C</i>
csc( <i>C</i> )		Calculates the cosecant of <i>C</i>
asin( <i>C</i> )		Calculates the inverse sine of <i>C</i>
acos( <i>C</i> )		Calculates the inverse cosine of <i>C</i>
atan( <i>C</i> )		Calculates the inverse tangent of <i>C</i>
atan2( <i>C</i> )		Calculates the principle value of <i>C</i>
sinh( <i>C</i> )		Calculates the hyperbolic sine of <i>C</i>
cosh( <i>C</i> )		Calculates the hyperbolic cosine of <i>C</i>
tanh( <i>C</i> )		Calculates the hyperbolic tangent of <i>C</i>
coth( <i>C</i> )		Calculates the hyperbolic cotangent of <i>C</i>
sech( <i>C</i> )		Calculates the hyperbolic secant of <i>C</i>
csch( <i>C</i> )		Calculates the hyperbolic cosecant of <i>C</i>
toString( <i>C</i> )		Formats <i>C</i> as a string.
<i>C</i> .toString()		The same as toString( <i>C</i> ).

Quaternion

Developed by: Fizyk

Description

Quaternions are an extension of complex numbers. Instead of  $a+bi$ , they are of form  $a+bi+cj+dk$ , where  $a, b, c, d$  are real numbers, and **i, j, k** are imaginary units. The imaginary units can be used as a basis in a 3D space, allowing quaternions to represent rotations.

Like on real and complex numbers, on quaternions you can perform addition, subtraction, multiplication and division. Operations that take a quaternion and a real/complex number are also supported ( $N+Q, Q*C$ , etc.). Beware: quaternion multiplication isn't commutative!










**Note:** Because multiplication isn't commutative with quaternions, there are two ways of dividing them.  $Q1/Q2$  is the same as  $Q1*inv(Q2)$ , the second way is  $inv(Q2)*Q1$ .

The extension also supports multiplying quaternions by vectors for the purpose of rotations. If you want to rotate vector  $V$  using quaternion  $Q$ , use this code:

$V2 = vec(Q*V*inv(Q))$

A short guide on quaternions can be found here: [\[1\]](#)

Commands

Function	Returns	Description
quat()		Creates a zero quaternion
quat( <i>N</i> )		Creates a quaternion with real part equal to <i>N</i>
quat( <i>C</i> )		Creates a quaternion with real and "i" parts equal to <i>C</i>
quat( <i>V</i> )		Converts a vector to a quaternion (returns $V.x*i + V.y*j + V.z*k$ )
quat( <i>N</i> , <i>N</i> <sub>2</sub> , <i>N</i> <sub>3</sub> , <i>N</i> <sub>4</sub> )		Returns $N+N_2i+N_3j+N_4k$
quat( <i>A</i> )		Converts <i>A</i> to a quaternion
quat( <i>V</i> , <i>V</i> <sub>2</sub> )		Creates a quaternion given forward ( <i>V</i> ) and up ( <i>V</i> <sub>2</sub> ) vectors
quat( <i>E</i> )		Converts angle of <i>E</i> to a quaternion
qi()		Returns quaternion <b>i</b>

qi(N)	<span>Q</span>	Returns quaternion $N*i$
qj()	<span>Q</span>	Returns $j$
qj(N)	<span>Q</span>	Returns $N*j$
qk()	<span>Q</span>	Returns $k$
qk(N)	<span>Q</span>	Returns $N*k$
abs(Q)	<span>N</span>	Returns absolute value of $Q$
conj(Q)	<span>Q</span>	Returns the conjugate of $Q$
inv(Q)	<span>Q</span>	Returns the inverse of $Q$
Q:real()	<span>N</span>	Returns the real component of the quaternion
Q:i()	<span>N</span>	Returns the $i$ component of the quaternion
Q:j()	<span>N</span>	Returns the $j$ component of the quaternion
Q:k()	<span>N</span>	Returns the $k$ component of the quaternion
exp(Q)	<span>Q</span>	Raises Euler's constant $e$ to the power $Q$
log(Q)	<span>Q</span>	Calculates natural logarithm of $Q$
qMod(Q)	<span>Q</span>	Changes quaternion $Q$ so that the represented rotation is by an angle between 0 and 180 degrees (by coder0xff)
slerp(Q,Q2,N)	<span>Q</span>	Performs spherical linear interpolation between $Q$ and $Q_2$ . Returns $Q$ for $N=0$ , $Q_2$ for $N=1$
Q:forward()	<span>V</span>	Returns vector pointing forward for $Q$
Q:right()	<span>V</span>	Returns vector pointing right for $Q$
Q:up()	<span>V</span>	Returns vector pointing up for $Q$
qRotation(V,N)	<span>Q</span>	Returns quaternion for rotation about axis $V$ by angle $N$
qRotation(V)	<span>Q</span>	Construct a quaternion from the rotation vector $V$ . Vector direction is axis of rotation, magnitude is angle in degress (by coder0xff)
rotationAngle(Q)	<span>N</span>	Returns the angle of rotation in degrees (by coder0xff)
rotationAxis(Q)	<span>V</span>	Returns the axis of rotation (by coder0xff)
rotationVector(Q)	<span>V</span>	Returns the rotation vector - rotation axis where magnitude is the angle of rotation in degress (by coder0xff)
vec(Q)	<span>V</span>	Converts $Q$ to a vector by dropping the real component
matrix(Q)	<span>M</span>	Converts $Q$ to a transformation matrix
Q:toAngle()	<span>A</span>	Returns angle represented by $Q$
toString(Q)	<span>S</span>	Formats $Q$ as a string.

## Basic extensions

### Core

### Description

This is where things directly related to E2 are kept

### Commands

Function	Returns	Description
first()	<span>N</span>	Returns 1 if the expression was spawned or reset
duped()	<span>N</span>	Returns 1 if the expression was duplicated
dupefinished()	<span>N</span>	Returns 1 when the contraption has finished duping. (Only triggers on Adv Duplicator, not the normal duplicator)

inputClk()	N	Returns 1 if the expression was triggered by an input
reset()		Reset the expression itself as if it was just spawned, stops execution
exit()		Stops the execution of any code after it
runOnLast(N)		If <activate> != 0, the chip will run once when it is removed, setting the last() flag when it does.
last()	N	Returns 1 if it is being called on the last execution of the expression gate before it is removed or reset. This execution must be requested with the runOnLast(1) command.
removing()	N	Returns 1 if this is the last() execution and caused by the entity being removed.
ops()	N	Returns how many ops are used every execution on average
cpuUsage()	N	Returns the average cpu usage in seconds. Multiply it by 1000000 to get the same number you see in E2's overlay text.
opcounter()	N	Returns how many ops have been used so far in this execution plus the amount of hard quota used
minquota()	N	The ops left before soft quota is used up
maxquota()	N	The ops left before hard quota is exceeded and the expression shuts down
softQuota()	N	Returns the size of the soft quota
hardQuota()	N	Returns the size of the hard quota
perf()	N	If used as a while loop condition, stabilizes the expression around <maxexceed> hardquota used.
perf(N)	N	Same as perf(), where N is the percentage of the soft quota the e2 will stabilise at.

Self-Aware

Description

With entity() you can use Entity-Support to get all the data from the expression-entity. With concmd() you can execute console commands.

Also, the chip has the ability to force itself. Forces aren't dispersed over a certain amount of time, all forces applied to an object within a tick are added up and then applied to the object. Force commands are best used with runOnTick(N) because you won't end up applying more than 1 force per tick and it is easier to do things like defy gravity.

Commands

Function	Returns	Description
entity()	E	Gets the entity of the expression
concmd(S)	N	Takes a string and executes it in console. Returns 1 if it succeeded and 0 if it failed. The client must enable this in the console with "wire_expression2_concmd 1". "wire_expression2_concmd_whitelist" allows you to choose which commands can be used. <a href="#">[2]</a>
applyForce(V)		Applies force according to the vector given (Forces independently on each axis unlike a vector thruster)
applyOffsetForce(V,V)		Applies force to the expression according to the first vector from the location of the second
applyAngForce(A)		Applies torque to the expression according to the given local angle
applyTorque(V)		Applies torque to the expression according to the given local vector, representing the torque axis, magnitude and direction
selfDestruct()		Removes the expression
selfDestructAll()		Removes the expression and all constrained props
ioOutputEntities(S)	R	Returns an array of all entities wired to the output S.
ioInputEntity(S)	E	Returns the entity the input S is wired to.
ioSetOutput(S,*)	-	Trigger the output S of the E2 with the value *.
ioGetInput*(S)	*	Get the value of the input S of the E2.
E:getName()	S	Returns the name of the E2 E.
setName(S)	-	Sets the name of the E2.



		Checks if the value or variable was changed. Accepts any type except <b>table</b> and <b>array</b> .
changed(*)		It detects changes by checking whether it was called with a different parameter at the same point in the last execution. Multiple calls to changed() in the <code>_same_</code> execution are independent of each other. Note: Put changed(*) first in an IF AND statement or keep it out of conditional statements all together as changed(*) will silently fail if it was not called the previous execution.
select(N,*,...)	*	Returns the Nth value given after the index, *'s zero element otherwise. If you mix types, the behaviour is undefined.
hash()	N	Returns a numerical hash using the code of the E2 itself (Including comments).
hash(S)	N	Returns a numerical hash using the string specified.
hashNoComments()	N	Returns a numerical hash using the code of the E2 itself (Excluding comments).

Debug

Description

Contains various functions for displaying values to the user. print() and hint() allow you to display strings quickly on your screen.

Keep in mind that chat messages can be faked using E:printColorDriver(...) and E:printColorDriver(R). The game will display a warning message when first used on someone by a specific chip.

Commands

Function	Returns	Description
print(S)		Posts <i>S</i> to the chat area.
print(...)		Prints all arguments to the chat area, seperated by a tab. Automatically does toString for you (Can print arrays but not tables). Works just like lua's <a href="#">print</a> .
E:printDriver(S)	<div>N</div>	Posts a string to the chat of <i>E</i> 's driver. Returns 1 if the text was printed, 0 if not.
hint(S,N)		Displays a hint popup with message <i>S</i> for <i>N</i> seconds ( <i>N</i> being clamped between 0.7 and 7).
E:hintDriver(S,N)	<div>N</div>	Displays a hint popup to the driver of vehicle <i>E</i> , with message <i>S</i> for <i>N</i> seconds ( <i>N</i> being clamped between 0.7 and 7). Same return value as printDriver.
print(N,S)		Same as print( <i>S</i> ), but can make the text show up in different places. <i>N</i> can be one of the following: <code>_HUD_PRINTCENTER</code> , <code>_HUD_PRINTCONSOLE</code> , <code>_HUD_PRINTNOTIFY</code> , <code>_HUD_PRINTTALK</code> .
E:printDriver(N,S)	<div>N</div>	Same as <i>EE</i> :printDriver( <i>S</i> ), but can make the text show up in different places. <i>N</i> can be one of the following: <code>_HUD_PRINTCENTER</code> , <code>_HUD_PRINTCONSOLE</code> , <code>_HUD_PRINTNOTIFY</code> , <code>_HUD_PRINTTALK</code> .
printTable(T)		Prints a table like the lua function <a href="#">PrintTable</a> does, except to the chat area.
printTable(R)		Prints an array like the lua function <a href="#">PrintTable</a> does, except to the chat area.
printColor(...)		Works like <a href="#">chat.AddText</a> (...). Parameters can be any amount and combination of numbers, strings, player entities, color vectors (both 3D and 4D).
printColor(R)		Like printColor(...), except taking an array containing all the parameters.
E:printColorDriver(...)		Like printColor but prints to the driver of a specified vehicle.
E:printColorDriver(R)		Like printColorDriver but takes an array containing all the parameters.

Timer

Description

Timer functions are a way to trigger the expression to be run at a given time. These functions let the expression be run continuously without needing triggering from inputs.

Commands

Function	Returns	Description
----------	---------	-------------

runOnTick(N)		If set to 1, the expression will execute once every game tick. See <a href="#">Admin#Tick</a> and <a href="#">[[3]]</a> For more information on how often this is run.
tickClk()	<div>N</div>	Returns 1 if the current execution was caused by "runOnTick"
curtime()	<div>N</div>	Returns the current game time since server-start in seconds*
realtime()	<div>N</div>	Returns the current real time since server-start in seconds*
interval(N)		Sets a one-time timer with name "interval" and delay in milliseconds (minimum delay for timers is 10ms)
timer(S,N)		Sets a one-time timer with entered name and delay in milliseconds
stoptimer(S)		Stops a timer, can stop interval with stoptimer("interval")
clk()	<div>N</div>	Returns 1 if the current execution was caused by the interval timer
clk(S)	<div>N</div>	Returns 1 if the current execution was caused by the inserted name

\* Both curtime() and realtime() are given to 3 decimal places. Server lag will cause curtime() to slow down, but not realtime().

Other

Here are a few other commands which aren't exactly timer functions, but they are related to time.

Function	Returns	Description
curtime()	<div>N</div>	Returns the time in seconds since the server was started. This can slow down if the server lags.
realtime()	<div>N</div>	Returns the time in seconds since the server was started. This does not slow down if the server lags.
sysptime()	<div>N</div>	Returns a highly accurate time (also in seconds) since the server was started. Ideal for benchmarking.
date()	<div>T</div>	Returns the server's current time and date, formatted neatly in a table.
date(N)	<div>T</div>	Returns the specified unix time, formatted neatly in a table.
dateUTC()	<div>T</div>	Returns the server's current time and date, formatted neatly in a table, in UTC.
dateUTC(N)	<div>T</div>	Returns the specified unix time, formatted neatly in a table, in UTC.
time()	<div>N</div>	Returns the server's current unix time.
time(T)	<div>N</div>	Converts a table of data (with the same table structure as the date() function above returns; not all values are required) into unix time.

Unit Conversion

Description

All conversions are precise so it is recommended to round the result if it is going to be displayed (round()).

Commands

Function	Returns	Description
toUnit(S,N)	<div>N</div>	Converts default garrysmo units to specified units
fromUnit(S,N)	<div>N</div>	Converts specified units to default garrysmo units
convertUnit(S,S,N)	<div>N</div>	Converts between two units

Units

Length	Description
u	garrysmo units (default)
mm	millimeters

cm	centimeters
dm	decimeters
m	meters
km	kilometers
in	inches
ft	feet
yd	yards
mi	miles
nmi	nautical miles

Speed	Description
<b>u/s</b>	<b>garrysmo<u>d</u> units per second (default)</b>
u/x	garrysmo <u>d</u> units per time unit
m/s	meters per second
km/h	kilometers per hour
in/s	inches per second
mi/h	miles per hour
mph	miles per hour (more commonly used than mi/h)
knots	knots (correct term for nautical miles per hour)
mach	mach (times speed of sound)
mm/x	millimeters per time unit
cm/x	centimeters per time unit
dm/x	decimeters per time unit
m/x	meters per time unit
km/x	kilometers per time unit
in/x	inches per time unit
ft/x	feet per time unit
yd/x	yards per time unit
mi/x	miles per time unit
nmi/x	nautical miles per time unit
	<b>substitute x for s (per second), m (per minute) or h (per hour)</b>

Weight	Description
g	grams
<b>kg</b>	<b>kilograms (default)</b>
t	tons
oz	ounces
lb	pounds

Server Information

Developed by: Beer


Description



The following functions allow you to get various information about the server, such as the current map name, gamemode, etc.

Commands

Function	Returns	Description
map()	<div>S</div>	Returns the current map name
hostname()	<div>S</div>	Returns the Name of the server
isLan()	<div>N</div>	Returns 1 if lan mode is enabled
gamemode()	<div>S</div>	Returns the name of the current gamemode
gravity()	<div>N</div>	Returns gravity for players (in the negative z direction)
propGravity()	<div>V</div>	Returns gravity for props
airDensity()	<div>N</div>	Returns air density (affects how drag slows down props)
maxFrictionMass()	<div>N</div>	Returns how much friction influences props throughout the server
minFrictionMass()	<div>N</div>	Returns how much friction influences props throughout the server
speedLimit()	<div>N</div>	Returns the speed limit
angSpeedLimit()	<div>N</div>	Returns the angular speed limit
tickInterval()	<div>N</div>	Returns the time (in seconds) between each server tick.
E:ping()	<div>N</div>	Returns the latency for player <i>E</i>
isSinglePlayer()	<div>N</div>	Returns 1 if singleplayer, 0 if multiplayer
isDedicated()	<div>N</div>	Returns 1 if server is dedicated, 0 if it is not
numPlayers()	<div>N</div>	Returns the number of players currently in the server
maxPlayers()	<div>N</div>	Returns the max number of players allowed in the server
maxOfType(S)	<div>N</div>	Returns the maximum allowed of a certain type of entity, i.e. maxOfType("wire_thrusters"). Returns 0 if you enter an invalid parameter.
playerDamage()	<div>N</div>	Returns 1 if player vs player damage is enabled on the server
convar(S)	<div>S</div>	Returns a clientside convar's setting.
convarnum(S)	<div>N</div>	Returns a clientside convar's setting.
time(S)	<div>N</div>	Returns numerical time/date info from the server. Possible arguments: " <b>year</b> ", " <b>month</b> ", " <b>day</b> ", " <b>hour</b> ", " <b>min</b> ", " <b>sec</b> ", " <b>wday</b> " (weekday, Sunday is 1), " <b>yday</b> " (day of the year), and " <b>isdst</b> " (daylight saving flag 0/1)

 **Tip:** To get a list of all possible parameters for maxOfType(), open the console and type "find sbx\_max". If you need "sbx\_maxragdolls", you can simply pass "ragdolls" in the function.

Constraint










Developed by: ZeikJT

Description

The following functions get information about entities based on constraints

Commands

Function	Returns	Description
E:getConstraints()	<div>R</div>	Returns an <b>array</b> with all entities directly or indirectly constrained to <i>E</i> , except <i>E</i> itself.
E:hasConstraints()	<div>N</div>	Returns the number of the constraints <i>E</i> has
E:hasConstraints(S)	<div>N</div>	Returns the number of the constraints <i>E</i> has with the given constraint type (see the types list below)

E:isConstrained()		Returns 1 if <i>E</i> has constraints, 0 if not
E:isWeldedTo()		Returns the first entity <i>E</i> was welded to
E:isWeldedTo(N)		Returns the <i>N</i> th entity <i>E</i> was welded to
E:isConstrainedTo()		Returns the first entity <i>E</i> was constrained to
E:isConstrainedTo(N)		Returns the <i>N</i> th entity <i>E</i> was constrained to
E:isConstrainedTo(S)		Returns the first entity <i>E</i> was constrained to with the given constraint type (see the types list below)
E:isConstrainedTo(S, N)		Returns the <i>N</i> th entity <i>E</i> was constrained to with the given constraint type (see the types list below)
E:parent()		Returns the <b>entity</b> <i>E</i> is parented to.
E:parentBone()		Returns the <b>bone</b> <i>E</i> is parented to.

Constraint Types
AdvBallsocket
Axis
Ballsocket
Elastic
Hydraulic
Keepupright
Motor
Muscle
NoCollide
Pulley
Rope
Slider
Weld
Winch





Chat

Developed by: ZeikJT & Gwahir

Description

The following functions are for reading the chat log. This is similar to the text receiver.

Commands

Function	Returns	Description
runOnChat(N)		If <i>N</i> == 0, the chip will no longer run on chat events, otherwise it makes this chip execute when someone chats. Only needs to be called once, not in every execution.
chatClk()		Returns 1 if the chip is being executed because of a chat event. Returns 0 otherwise.
chatClk(E)		Returns 1 if the chip is being executed because of a chat event by player <i>E</i> . Returns 0 otherwise.
hideChat(N)		If <i>N</i> != 0, hide the chat message that is currently being processed.
lastSpoke()		Returns the last player to speak.
lastSaid()		Returns the last message in the chat log.

lastSaidWhen()	<div>N</div>	Returns the time the last message was sent.
lastSaidTeam()	<div>N</div>	Returns 1 if the last message was sent in the team chat, 0 otherwise.
E:lastSaid()	<div>S</div>	Returns what the player <i>E</i> last said.
E:lastSaidWhen()	<div>N</div>	Returns when the given player last said something.
E:lastSaidTeam()	<div>N</div>	Returns 1 if the last message was sent in the team chat, 0 otherwise.

Color

Developed by: Jimlad

Description

These commands allow E2 to find the color of an entity and change it. Changing color only works on entities you own.

Uses RGBA (Red, Green, Blue, Alpha) values, although when only RGB is specified, alpha will not be changed.

Note that color values have a range of 0 - 255, where (0,0,0,255) is black, and (255,255,255,255) is white.

Alpha is equivalent to opacity, where 0 is completely transparent and 255 is completely opaque.

Commands

Function	Returns	Description
E:getColor()	<div>V</div>	Returns the color of an entity as a vector (R,G,B)
E:getColor4()	<div>V4</div>	Returns the color of an entity as a 4D vector (R,G,B,A)
E:getAlpha()	<div>N</div>	Returns the alpha of an entity
E:getMaterial()	<div>S</div>	Returns the material of an entity
E:getSkin()	<div>N</div>	Gets <i>E</i> 's current skin number.
E:getSkinCount()	<div>N</div>	Gets <i>E</i> 's number of skins.
E:getBodygroups(N)	<div>N</div>	Gets the number of groups in the given GroupID.
E:setColor(N,N,N)		Changes the RGB color of an entity (leaves alpha alone)
E:setColor(N,N,N,N)		Changes the RGBA color of an entity
E:setColor(V)		Changes the RGB color of an entity (leaves alpha alone), using a vector with values (R,G,B)
E:setColor(V,N)		Changes the RGBA color of an entity, using a vector with values (R,G,B). The additional argument sets alpha
E:setColor(V4)		Changes the RGBA color of an entity, using a 4D vector with values (R,G,B,A)
E:setAlpha(N)		Changes the alpha of an entity
E:setMaterial(S)		Sets the material of an entity. E:setMaterial("") to reset material
E:setSkin(N)		Sets <i>E</i> 's skin number.
E:setBodygroup(N,N)		Sets <i>E</i> 's bodygroup id and subid.
hsv2rgb(V)	<div>V</div>	Converts <i>V</i> from the <a href="#">HSV color space</a> to the <a href="#">RGB color space</a>
hsv2rgb(N,N,N)	<div>V</div>	Converts <i>N,N,N</i> from the <a href="#">HSV color space</a> to the <a href="#">RGB color space</a>
rgb2hsv(V)	<div>V</div>	Converts <i>V</i> from the <a href="#">RGB color space</a> to the <a href="#">HSV color space</a>
hsl2rgb(V)	<div>V</div>	Converts <i>V</i> from the <a href="#">HSL color space</a> to the <a href="#">RGB color space</a>
hsl2rgb(N,N,N)	<div>V</div>	Converts <i>N,N,N</i> from the <a href="#">HSL color space</a> to the <a href="#">RGB color space</a>
rgb2digi(V,N)	<div>N</div>	Converts an RGB vector <i>V</i> to a number in digital screen format. <i>N</i> Specifies a mode, either 0, 2 or 3, corresponding to Digital Screen color modes.
rgb2digi(N,N <sub>2</sub> ,N <sub>3</sub> ,N <sub>4</sub> )	<div>N</div>	Converts the RGB color ( <i>N,N<sub>2</sub>,N<sub>3</sub></i> ) to a number in digital screen format. <i>N<sub>4</sub></i> Specifies a mode, either 0, 2 or 3, corresponding to Digital Screen color modes.

## Advanced extensions

### E2 Function System

**Developed by:** Rusketh

#### Description

This allows the user to create functions in their E2 code. Functions must be defined before they are used, which means they are typically at the top of the code. Functions are created at runtime so they should only be declaired once, additionally this also allows you overwrite existing custom functions as long as the return type is not changed. Existing predefined functions may not be overwritten or changed.

This example makes a function equivalent to the built-in entity(N) function. It takes one parameter, which is a number called EntityID, and returns an entity. Function names must start with a lowercase letter.

```
function entity doSomething(EntityID) {
    return entity(EntityID)
}
```

You can call it like any other function:

```
print(doSomething(1))
```

If a function doesn't return anything, the return type should be "void" or can be left blank.

```
function void notReturningAnything() {
    print("hi")
}
```

If a parameter is not a number, then you need to specify it's type using "ID:type", as in @persist/@inputs/@outputs directives.

```
function vector doSomethingElse(Number, Vector:vector) {
    return Vector * Number
}
```

Functions can also be defined as methods of another type. While inside the function body, the variable This is used to refer to the object on which this method is being called. The This variable does not obey usual scoping rules (see below).

```
function void entity:forceToward(Pos:vector, Mul) {
    This:applyForce(Mul*(Pos-This:pos()))
}
```

Functions will also run inside there own environment. See the 'E2 Variable Scopes' section for more information.

### Callable strings

**Developed by:** Divran

#### Description

This extension is a replacement for the idea of having functions as objects. Because making functions objects was such a massive undertaking, it was made possible to use strings instead.

The idea is to allow users to store references to functions in a table, or for use with callbacks, or other similar things.

Example use:

```
function doSomething(Str:string) {
    print("hello, " + Str)
}
```

```
SomeString = "doSomething"  
SomeString("world")
```

will print "hello, world"

You can also retrieve returned values from functions. To do so, you must specify the type that the function returns. Example:

```
function string doSomething() {  
    return "hello, world"  
}  
  
Str = "doSomething"  
ReturnValue = Str()[string]  
print(ReturnValue) #Will print "hello, world"
```

## Notable features

This extension does not act the way you'd expect in a few situations. For example, let's say you made this function

```
function table:doSomething(Index) {  
    print(This[Index,number])  
}
```

How would you call this function? The way you call it is slightly strange, but necessary due to the way this works. Here's how to call it:

```
Str = "doSomething"  
Str(table(1,2,3),2)) # Will print '2'
```

Now, there is a slight issue with this. If you make two similar functions, like so:

```
function table:doSomething(Index) {  
    print(This[Index,number])  
}  
function doSomething(This:table,Index) {  
    print(This[Index,number])  
}
```

If you do this, you will not be able to call the first of the two (table:doSomething(index)) because it will prioritize the second function.

I hope you find this useful!

## Entity Discovery

**Developed by:** Gwahir, TomyLobo

### Description

Use these to find and filter entities. The basic find functions will return how many entities were found but the actual entities are stored on the chip until they are accessed using find(), findResult(N), or findClosest(V)

There is a white list and a black list as well as functions for on the spot filtering and sorting. White and black lists are always in effect and will be used automatically when you request a new list of entities. Control of the lists is achieved through the find[Exclude, Allow, Include, Disallow][Player, Prop, Model, Class] functions. Exclude/Allow controls the blacklist while Include/Disallow controls the whitelist. If the same object is covered by both the whitelist and the blacklist, the blacklist takes priority.

In the case of names, classes and models, partial strings are acceptable.

Discovering entities is not cheap so suggested usage is to find what you're looking for and store it in a persitant variable to limit the number of queries you run. To prevent overuse of these features, two console variables have been included, wire\_exp2\_entFindRate and wire\_exp2\_playerFindRate. These are delays that control how often you can perform find queries. **This means that you cannot run find functions every tick with runOnTick(1)!** The ent variable is per chip, the player variable is for all chip owned by a specific player.

Commands

Function	Returns	Description
findUpdateRate()	<b>N</b>	Returns the minimum delay between entity find events on a chip
findPlayerUpdateRate()	<b>N</b>	Returns the minimum delay between entity find events per player
findCanQuery()	<b>N</b>	Returns 1 if find functions can be used, 0 otherwise.
findInSphere(V,N)	<b>N</b>	Finds entities in a sphere around V with a radius of N, returns the number found after filtering
findInCone(V,V,N,N)	<b>N</b>	Like findInSphere but with a <a href="#">[Spherical cone]</a> , arguments are for position, direction, length, and degrees.
findInBox(V,V)	<b>N</b>	Like findInSphere but with a globally aligned box, the arguments are the diagonal corners of the box
findByName(S)	<b>N</b>	Find all entities with the given name
findByModel(S)	<b>N</b>	Find all entities with the given model
findByClass(S)	<b>N</b>	Find all entities with the given class
findPlayerByName(S)	<b>E</b>	Returns the player with the given name, this is an exception to the rule
findExcludeEntities(R)		Exclude all entities from <i>R</i> from future finds
findExcludeEntity(E)		Exclude <i>E</i> from future finds
findExcludePlayer(E)		Exclude this player from future finds (put it on the entity blacklist)
findExcludePlayer(S)		Exclude this player from future finds (put it on the entity blacklist)
findExcludePlayerProps(E)		Exclude entities owned by this player from future finds
findExcludePlayerProps(S)		Exclude entities owned by this player from future finds
findExcludeModel(S)		Exclude entities with this model (or partial model name) from future finds
findExcludeClass(S)		Exclude entities with this class (or partial class name) from future finds
findAllowEntities(R)		Remove all entities from <i>R</i> from the blacklist
findAllowEntity(E)		Remove <i>E</i> from the blacklist
findAllowPlayer(E)		Remove this player from the entity blacklist
findAllowPlayer(S)		Remove this player from the entity blacklist
findAllowPlayerProps(E)		Remove entities owned by this player from the blacklist
findAllowPlayerProps(S)		Remove entities owned by this player from the blacklist
findAllowModel(S)		Remove entities with this model (or partial model name) from the blacklist
findAllowClass(S)		Remove entities with this class (or partial class name) from the blacklist
findIncludeEntities(R)		Include all entities from <i>R</i> in future finds, and remove others not in the whitelist
findIncludeEntity(E)		Include <i>E</i> in future finds, and remove others not in the whitelist
findIncludePlayer(E)		Include this player in future finds, and remove other entities not in the entity whitelist
findIncludePlayer(S)		Include this player in future finds, and remove other entities not in the entity whitelist
findIncludePlayerProps(E)		Include entities owned by this player from future finds, and remove others not in the whitelist
findIncludePlayerProps(S)		Include entities owned by this player from future finds, and remove others not in the whitelist
findIncludeModel(S)		Include entities with this model (or partial model name) in future finds, and remove others not in the whitelist
findIncludeClass(S)		Include entities with this class (or partial class name) in future finds, and remove others not in the whitelist
findDisallowEntities(R)		Remove all entities from <i>R</i> from the whitelist
findDisallowEntity(E)		Remove <i>E</i> from the whitelist
findDisallowPlayer(E)		Remove this player from the entity whitelist
findDisallowPlayer(S)		Remove this player from the entity whitelist
findDisallowPlayerProps(E)		Remove entities owned by this player from the whitelist
findDisallowPlayerProps(S)		Remove entities owned by this player from the whitelist

findDisallowModel(S)		Remove entities with this model (or partial model name) from the whitelist
findDisallowClass(S)		Remove entities with this class (or partial class name) from the whitelist
findClearBlackList()		Clear all entries from the entire blacklist
findClearBlackEntityList()		Clear all entries from the entity blacklist
findClearBlackPlayerPropList()		Clear all entries from the prop owner blacklist
findClearBlackModelList()		Clear all entries from the model blacklist
findClearBlackClassList()		Clear all entries from the class blacklist
findClearWhiteList()		Clear all entries from the entire whitelist
findClearWhiteEntityList()		Clear all entries from the player whitelist
findClearWhitePlayerPropList()		Clear all entries from the prop owner whitelist
findClearWhiteModelList()		Clear all entries from the model whitelist
findClearWhiteClassList()		Clear all entries from the class whitelist
findResult(N)	E	Returns the indexed entity from the previous find event (valid parameters are 1 to the number of entities found)
findClosest(V)	E	Returns the closest entity to the given point from the previous find event
findToArray()	R	Formats the query as an array, R[Index,entity] to get an entity.
findToTable()	T	Formats the query as a table, T[Index,entity] to get an entity.
find()	E	Equivalent to findResult(1)
findSortByDistance(V)	N	Sorts the entities from the last find event, index 1 is the closest to point V, returns the number of entities in the list
findClipToClass(S)	N	Filters the list of entities by removing all entities that are NOT of this class
findClipFromClass(S)	N	Filters the list of entities by removing all entities that are of this class
findClipToModel(S)	N	Filters the list of entities by removing all entities that do NOT have this model
findClipFromModel(S)	N	Filters the list of entities by removing all entities that do have this model
findClipToName(S)	N	Filters the list of entities by removing all entities that do NOT have this name
findClipFromName(S)	N	Filters the list of entities by removing all entities that do have this name
findClipToSphere(V,N)	N	Filters the list of entities by removing all entities NOT within the specified sphere (center, radius)
findClipFromSphere(V,N)	N	Filters the list of entities by removing all entities within the specified sphere (center, radius)
findClipToRegion(V,V)	N	Filters the list of entities by removing all entities NOT on the positive side of the defined plane. (Plane origin, vector perpendicular to the plane) You can define any convex hull using this.

Global Variables

Developed by: Divran (Original idea by ZeikJT)

Description

Global variables are a way to exchange data between two expression chips without the need for any wiring at all.

All variables in a non-shared table will automatically be removed if you disconnect from the server. Remember that variables in a shared table will not automatically be removed, so don't spam too many of them.

Shared means that any E2 can access the globals, not only your own. Non-shared allows only your own E2's to access your globals.

Commands

Function	Returns	Description
'New' syntax		The new syntax. Supports <i>any</i> type.



<b>Getting the gtable</b>		Get the gtable with which you use the regular table get and set syntax: "G[index,type]"
gTable(S)	GT	Returns a non-shared gtable with the group <i>S</i>
gTable(S,N)	GT	Returns a gtable with the group <i>S</i> . <i>N</i> determines whether or not it is shared. Remember that there are two tables: one which is shared and one which is not; values do not transition between the two.
gTableSafe(N)	GT	Returns a safe gtable which group is a numerical hash created from the code of the E2 itself. This is very useful in multiplayer games, as it makes it impossible to edit the code in order to cheat (by giving yourself infinite health or ammo etc) since the cheater will then be joining a different group (edited E2 means different hash).
		Note that the hash ignores comments, so editing the comments in your E2 won't change the group.
<b>Removing</b>		Remove variables & clear tables
GT:remove*(S)	*	Removes and returns the variable of the type * at the index <i>S</i>
GT:clear()	-	Clears the table <i>GT</i>
gRemoveAll*()	-	Removes all variables of the type * in your non-shared table.
gRemoveAll*(S)	-	Removes all variables of the type * in your non-shared table in group <i>S</i> .
gRemoveAll()	-	Resets the entire non-shared table (ie ALL your variables in every group)
<b>'Old' syntax</b>		The old syntax. Only supports strings, numbers, vectors, angles and entities. Use the first three letters when specifying type. These functions are NOT recommended. Use the new syntax instead.
<b>Group control</b>		Change group & share or stop sharing your variables
gSetGroup(S)	-	Sets the E2's current group. Does persist.
gGetGroup()	S	Gets the E2's current group.
gShare(N)	-	Sets wether or not you want to share the variables. (1/0) Remember that there are two tables for each group: one which is shared and one which is not; values do not transition between the two.
gGetShare()	N	Returns 1/0
gResetGroup()	-	Resets the group back to "default".
<b>Getting and setting</b>		Save & load variables
gSet*(S,*)	-	Sets a variable of the type * at index <i>S</i> in the current group.
gGet*(S)	*	Gets a variable of the type * from index <i>S</i> in the current group.
gSet*(N,*)	-	Exactly the same as "gSet*(N:toString(),*)"
gGet*(N)	-	Exactly the same as "gGet*(N:toString())"
<b>Removing</b>		Remove variables & clear tables
gDelete*(S)	*	Removes and returns the variable of the type * at the index <i>S</i> in the current group.
gDelete*(N)	*	Exactly the same as gDelete*(N:toString())
gDeleteAll*()	-	Exactly the same as gRemoveAll*(S) (Except it removes in the group set by gSetGroup instead of using the group as an argument) (Remember that this function is only for compatibility)

Built-In Ranger

Developed by: ZeikJT

Description
























The built-in ranger is based on Erkle's original ranger. There are however some new functionalities that can be found in the commands below. Keep in mind that if you want to apply an option you must set it before getting the ranger data. To make ranger settings (like filters, "ignore world", "hit water") persist, run rangerPersist(1).

This also introduces a new Variable type, the RD (defined as :ranger). It holds the data returned after a trace, you will need to use the trace data functions to retrieve useful data. These are to be used after you have done an actual trace.



I will add a simple example to showcase the syntax and functionality.

Commands

Function	Returns	Description
<b>Ranger Options</b>		To be used before an actual ranger trace
rangerPersist(N)		Passing 0 (the default) resets all ranger flags and filters every execution and after calling ranger/rangerOffset. Passing anything else will make the flags and filters persist until they're changed again.
rangerReset()		Resets all ranger flags and filters.
rangerFlags()		Returns the ranger flags as a string.
rangerFlags(S)		Sets the ranger flags. S can be any combination of I=ignore world, W=hit water, E=hit entities and Z=default to zero.
rangerHitWater(N)		Default is 0, if any other value is given it will hit water
rangerHitEntities(N)		Default is 1, if value is given as 0 it will ignore entities
rangerIgnoreWorld(N)		Default is 0, if any other value is given it will ignore world
rangerDefaultZero(N)		If given any value other than 0 it will default the distance data to zero when nothing is hit
rangerFilter(E)		Feed entities you don't want the trace to hit
rangerFilter(R)		Feed an array of entities you don't want the trace to hit
<b>Ranger Tracing</b>		Gathers data, if options are declared prior to this they will be used
ranger(N)		You input max range, it returns ranger data
ranger(N,N,N)		Same as above with added inputs for X and Y skew
rangerAngle(N,N,N)		You input the distance, x-angle and y-angle (both in degrees) it returns ranger data
rangerOffset(V,V)		You input two vector points, it returns ranger data
rangerOffset(N,V,V)		You input the range, a position vector, and a direction vector and it returns ranger data
<b>Ranger Hull Tracing</b>		Same as above, except the traces have a size. Unfortunately, the hulls cannot be rotated. (Made by Divran)
rangerHull(N,V)		Inputs: Distance, Hull BoxSize
rangerHull(N,V,V)		Input: Distance, Hull MinSize, Hull MaxSize
rangerHull(N,N,N,V)		Inputs: Distance, X Skew, Y Skew, Hull BoxSize
rangerHull(N,N,N,V,V)		Inputs: Distance, X Skew, Y Skew, Hull MinSize, Hull MaxSize
rangerHullAngle(N,N,N,V)		Inputs: Distance, X Angle, Y Angle, Hull BoxSize
rangerHullAngle(N,N,N,V,V)		Inputs: Distance, X Angle, Y Angle, Hull MinSize, Hull MaxSize
rangerOffsetHull(V,V,V)		Inputs: StartPos, EndPos, Hull BoxSize
rangerOffsetHull(V,V,V,V)		Inputs: StartPos, EndPos, Hull MinSize, Hull MaxSize
rangerOffsetHull(N,V,V,V)		Inputs: Distance, StartPos, Direction, Hull BoxSize
rangerOffsetHull(N,V,V,V,V)		Inputs: Distance, StartPos, Direction, Hull MinSize, Hull MaxSize
<b>Ranger Data Retrieval</b>		Accesses data stored in an RD container
RD:distance()		Outputs the distance from the rangerdata input, else depends on rangerDefault
RD:position()		Outputs the position of the input ranger data trace IF it hit anything, else returns (0,0,0)
RD:entity()		Returns the entity of the input ranger data trace IF it hit an entity, else returns nil
RD:hit()		Returns 1 if the input ranger data hit anything and 0 if it didn't
RD:hitNormal()		Outputs a normalized vector perpendicular to the surface the ranger is pointed at.
RD:pos()		Returns the hit position. The difference between this function and RD:position() is that if you start the trace inside a world brush, RD:position() will return the position at which the trace EXITS the world. RD:pos(), however, will continue on and return the hit position outside the wall you started the trace in.
RD:fraction()		Returns a number between 0-1 which represents the percentage of the distance between the start & hit position of the trace. StartPos + (EndPos-StartPos):normalized() * RD:fraction() * (EndPos-StartPos):Length() is equal to RD:pos()

RD:hitWorld()	<div>N</div>	Returns 1 if the trace hit the world, else 0.
RD:hitSky()	<div>N</div>	Returns 1 if the trace hit the sky, else 0.
RD:positionLeftSolid()	<div>V</div>	Returns the position at which the trace left a world brush, if it was started inside a world brush. Else return the trace's Start Position.
RD:fractionLeftSolid()	<div>N</div>	Same as RD:fraction() except it represents the distance between the start position and the LeftSolid position.
RD:startSolid()	<div>N</div>	Returns 1 if the trace was started inside a world brush, else 0.
RD:matType()	<div>S</div>	Returns the material type (ie "wood, metal, dirt, flesh", etc)
RD:hitGroup()	<div>S</div>	Returns the hit group (ie "chest, face, left arm, right leg", etc)
RD:toTable()	<div>T</div>	Converts the trace data into an E2-style table and returns it. Remember that this returns the raw data, so for matType and hitGroup, it is recommend that you use the functions instead of this table.

Sound Playback

Developed by: ZeikJT

Description

Allows Expression 2 to play sounds. NEW: Expression2 Now has a Sound Browser, Use it instead!

The Duration is in seconds. If the sound is meant to be looped, set the duration to zero. If a sound is not designed to be looped (i.e: actor talking), it won't loop. The path must contain slashes '/' and not backslashes '\'. The soundPlay functions can optionally play from an entity that you own.

Commands

Function	Returns	Description
(E:)soundPlay(N,N,S)		soundPlay(int Index, int Duration, string Path to File)
(E:)soundPlay(S,N,S)		soundPlay(string Index, int Duration, string Path to File)
(E:)soundPlay(N,N,S,N)		soundPlay(int Index, int Duration, string Path to File, int FadeTime)
(E:)soundPlay(S,N,S,N)		soundPlay(string Index, int Duration, string Path to File, int FadeTime)
soundStop(N)		Stops the sound stored at the integer index and removes the entry
soundStop(N,N)		Fades the sound stored at the first input's integer index in the second input's amount of seconds and removes the entry
soundStop(S)		Stops the sound stored at the string index and removes the entry
soundStop(S,N)		Fades the sound stored at the string index in the integer input's amount of seconds and removes the entry
soundPitch(N,N)		soundPitch(integer Index, integer Pitch) Default Pitch is 100, max is 255. Pitch is scaled linearly (like frequency), rather than by octave
soundPitch(S,N)		Same as above but takes a string index instead of an integer index
soundVolume(N,N)		soundVolume(integer Index, Volume), where Volume is a number between 0 and 1. Default Volume is 1
soundVolume(S,N)		Same as above but takes a string index instead of an integer index
soundPurge()		Clears the sound table and stops all sounds
soundDuration(S)	<div>N</div>	soundDuration(string Path to File) Returns the duration of the sound. Note: If the server hasn't got the file it returns 60




NPC control

Developed by: Bobsymalone

Description

These functions allow you to control NPCs. You can create secondary AI systems responding to wire by telling NPCs how to feel about certain things, where to go, etc. You can also equip them with weapons.

Commands

Function	Returns	Description
E:npcStop()		Stops any anything the NPC is doing, including things it decided to do by itself
E:npcGoWalk(V)		Tells the NPC to walk to position V
E:npcGoRun(V)		Tells the NPC to run to position V
E:npcFace(V)		This will rotate the NPC to face position V. This is purely aesthetic and can't be used to aim their weapon.
E:npcAttack()		Tells the NPC to use their melee attack.
E:npcShoot()		Tells the NPC to shoot their gun
E:npcGiveWeapon()		Gives the NPC an SMG
E:npcGiveWeapon(S)		Gives the NPC a weapon. Example: E:npcGiveWeapon("pistol"). Other arguments include "ar2", "crowbar", "357", "shotgun", "crossbow", "rpg", "frag", etc. Other such as the bugbait or slam may be buggy.
E:npcSetTarget(E)		Sets the npcs current target.
E:npcGetTarget()		Returns what the npc is currently targeting.
E:npcRelationship(E,S,N)		Will set the NPC's relationship to the specified entity to the S input, priority N. Priority is any number between 0 and 999. The relationship string can be either "like" "neutral" "hate" or "fear". Same goes for all other relationship functions.
E:npcRelationship(S,S,N)		Same as above, but sets relationship to an entire class specified by the first string. Example: "npc_manhack", "prop_physics".
E:npcRelationshipByOwner(E,S,N)		Sets the NPC's relationship to all currently existing NPCs owned by player E. Returns number of entities added to relationships.
E:npcDisp(E)		Returns the NPC's relationship to entity E.

Signals

Developed by: Gwahir, TomyLobo

Description

These functions allow you to remotely execute exp2 chips, provided that chip is set to receive the given signal

Scope

Signals are restricted to certain scopes (only you, anyone, only others) (0,1,2)

Simplified, true = anyone, false = only you.

Scopes are used to restrict both who can receive your signal and who's signal you can receive.

Scopes are always relative to the owner of the chip. So if player A sends to scope 1 and player B only receives from scope 0, he/she won't receive it, but player B will receive it with scopes 1 or 2

Group

Set the chip's group with signalSetGroup(S) before calling the related runOnSignal, sendSignal, or signalSetOnRemove function

The chip's signal group is always "default" at the start of every execution.

runOnSignal() will subscribe to the given signal within the current group, this applies to sent signals as well.

Any signal the chip receives will run the chip regardless of its current group (so long as it subscribed to the signal and group of the sent signal)

A chip will never run because of a signal it sent itself.

Signals are issued 10ms after the first unissued signal was sent.  
There can only ever be one unissued signal/group combination per receiver in each scope.

Commands

Function	Returns	Description
signalSetGroup(S)	<div>S</div>	Sets the E-2's current signal group to <i>S</i> , this is applied during runOnSignal, signalSend, and signalSetOnRemove calls, so call it first.
signalGetGroup()		Gets the E-2's current signal group
runOnSignal(S,N,N2)		If <i>N2</i> == 0 the chip will no longer run on this signal, otherwise it makes this chip execute when signal <i>S</i> is sent by someone in scope <i>N</i> .
signalClk()	<div>N</div>	Returns 1 if the chip was executed because of any signal, regardless of name, group or scope. Returns 0 otherwise.
signalClk(S)	<div>N</div>	Returns 1 if the chip was executed because the signal <i>S</i> was sent, regardless of group or scope. Returns 0 otherwise.
signalClk(S,N)	<div>N</div>	Returns 1 if the chip was executed because the signal <i>S</i> was sent to the scope <i>N</i> , regardless of group. Returns 0 otherwise.
signalClk(S,S2)	<div>N</div>	Returns 1 if the chip was executed because the signal <i>S2</i> was sent in the group <i>S</i> , regardless of scope. Returns 0 otherwise.
signalClk(S,S2,N)	<div>N</div>	Returns 1 if the chip was executed because the signal <i>S2</i> was sent in the group <i>S</i> to the scope <i>N</i> . Returns 0 otherwise.
signalName()	<div>S</div>	Returns the name of the received signal.
signalGroup()	<div>S</div>	Returns the group name of the received signal.
signalSender()	<div>E</div>	Returns the entity of the chip that sent the signal.
signalSenderId()	<div>N</div>	Returns the entity ID of the chip that sent the signal. Useful if the entity doesn't exist anymore.
signalSetOnRemove(S,N)		Sets the signal that the chip sends when it is removed from the world.
signalClearOnRemove()		Clears the signal that the chip sends when it is removed from the world.
signalSend(S,N)		Sends signal <i>S</i> to scope <i>N</i> . Additional calls to this function with the same signal will overwrite the old call until the signal is issued.
signalSendDirect(S,E)		Sends signal <i>S</i> to the given chip. Multiple calls for different chips do not overwrite each other.
signalSendToPlayer(S,E)		sends signal <i>S</i> to chips owned by the given player, multiple calls for different players do not overwrite each other

Data Signals

Developed by: Divran

Description

This extension allows you to transmit data and execute E2s remotely.

Remember: When sending a table or array, it only sends the table reference. This means that if you then edit the table back on the first E2, the table will also be edited on the second E2. To fix this, if needed, use the clone() function before or after sending.

Scope

As mentioned above, you can set the scope of the E2 itself in order to choose which signals it should allow. If you set the scope of the E2 itself, the following will happen:

- 0: Only allow signals from E2s you own.
- 1: Allow signals from E2s you own and from people in your prop protection friends list.
- 2: Allow signals from anyone.

You can also choose which scope to send a signal to when you call the send functions. If you choose the scope while calling the function, the following will happen:

- 0: Only send to your E2s.
- 1: Send to your E2s and the people who have you in their prop protection friends list.
- 2: Send to everyone.

The default scope is 0.

Group

When I said "send to everyone" above, I didn't mean every single E2 on the map. It sends to everyone in a specific group. You can change the group of the E2 at any time, and you can specify which group to send a signal to. The E2 is not in a group by default, and you must join a group in order to receive any non-direct signals.

Signal Names

Signal names serve no other purpose than to identify the signal so that the recieving E2 can know what to do with the data.

Commands

Note that \* can be replaced by any variable type.

Function	Returns	Description
<b>Sending</b>		
dsSendDirect(S,E,*)	N	Sends the data * directly to the E2 <i>E</i> , with the signal name <i>S</i> . Returns 1 if successful.
dsSendDirect(S,R,*)	N	Sends the data * directly to all E2s in the array <i>R</i> , with the signal name <i>S</i> . Returns 0 if any one of the signals were unsuccessful.
dsSend(S1,S2,*)	N	Sends the data * to all E2s in the same scope as the E2 and group <i>S2</i> , with the signal name <i>S1</i> . Returns 0 if any one of the signals were unsuccessful.
dsSend(S1,S2,N,*)	N	Sends the data * to all E2s in the group <i>S2</i> and the scope <i>N</i> , with the signal name <i>S1</i> . Returns 0 if any one of the signals were unsuccessful.
<b>Receiving</b>		
dsClk()	N	Returns 1 if the current execution was caused by ANY datasignal. Returns 0 otherwise.
dsClk(S)	N	Returns 1 if the current execution was caused by a signal with the signal name <i>S</i> . Returns 0 otherwise.
dsClkName()	S	Returns the name of the datasignal which caused the current execution. Returns an empty string if no datasignal caused the current execution.
dsGet*()	*	Returns the recieved data. Example: dsGetNumber(), dsGetString(), dsGetVector()
dsGetType()	S	Returns the type of the recieved data.
dsGetSender()	E	Returns the E2 which sent the signal.
dsGetGroup()	S	Returns the name of the group the signal was sent from.
<b>Grouping</b>		
dsJoinGroup(S)	-	Joins the group <i>S</i> . The E2 will now receive signals sent in this group, as well as any other groups the E2 is in.
dsLeaveGroup(S)	-	Leaves the group <i>S</i> . The E2 will no longer receive signals sent in this group.
dsClearGroups()	-	Leave all groups. The E2 will no longer receive any signals at all, except dsSendDirect signals.
dsGetGroups()	R	Returns an array with the names of all groups the E2 is currently in.
dsSetScope(N)	-	Sets the scope of the E2 to <i>N</i> . See above for what setting the scope does.
dsGetScope()	N	Returns the scope the E2 is currently in.
<b>Probing</b>		
dsProbe(S)	R	Returns an array of the E2s which would have recieved a signal if you had sent it to the group <i>S</i> and the E2s scope.
dsProbe(S,N)	R	Returns an array of the E2s which would have recieved a signal if you had sent it to the group <i>S</i> and the scope <i>N</i> .

GLON

Developed by: E2 extension by TomyLobo, anti-hack by !cake



Description

This extension allows you to serialize (=turn into a string) an array or table of values. Unsupported element types are:

- Wirelink (this is deliberate, to stop you hacking other people's wirelinks)
- Bone
- some entity types (Vehicle, NPC, Weapon)
- ranger data if the ranger was pointed at one of the entity types mentioned previously.

NOTE

glon was removed in gmod 13. While this extension will remain available in E2, it won't work unless the server owner manually reinstalls glon. Use [vON](#) instead.

Commands

Function	Returns	Description
glonEncode(R)		Encodes <i>R</i> into a string, using <a href="#">GLON</a> .
glonEncode(T)		Encodes <i>T</i> into a string, using <a href="#">GLON</a> .
glonDecode(S)		Decodes <i>S</i> into an array, using <a href="#">GLON</a> .
glonDecodeTable(S)		Decodes <i>S</i> into a table, using <a href="#">GLON</a> .

vON

**Developed by:** E2 extension by Divran, vON by Vercas, anti-hack by !cake

Description

This extension is a replacement for glon in gmod 13. It does the exact same thing.

Commands

Function	Returns	Description
vonEncode(R)		Encodes <i>R</i> into a string, using <a href="#">vON</a> .
vonEncode(T)		Encodes <i>T</i> into a string, using <a href="#">vON</a> .
vonDecode(S)		Decodes <i>S</i> into an array, using <a href="#">vON</a> .
vonDecodeTable(S)		Decodes <i>S</i> into a table, using <a href="#">vON</a> .

3D Holograms

**Developed by:** McLovin & ZeikJT

Description

Adds the ability to project 3D objects. These objects can't be interacted with like most props; the only way to manipulate them is to use these functions.

When using the holoCreate function, bear in mind that there is a delay associated with spawning holograms to avoid lagging servers. Avoid using holoCreate every execution. In general you should only ever use the holoCreate function once for each hologram in your code, for example by using the first() condition. Use the other functions like holoPos to update them thereafter.












Note that except for `wire_holograms_display_owners`, `wire_holograms_block_client` and `wire_holograms_unblock_client` all other console commands are useable by **admins only**!



Console Variables

Function	Returns	Description
wire_holograms_display_owners		Shows the owner of each hologram.
wire_holograms_block_client		Hide all holograms spawned by the specified player from the display_owners list.
wire_holograms_unblock_client		Un-hide/show all holograms spawned by the specified player on the display_owners list.
wire_holograms_remove_all		Removes all holograms on the map
wire_holograms_block		Input a name (or part of a name) to prevent a player from spawning holograms
wire_holograms_unblock		Input a name (or part of a name) to allow a player to spawn holograms again
wire_holograms_block_id		Input a SteamID to prevent a player from spawning holograms
wire_holograms_unblock_id		Input a SteamID to allow a player to spawn holograms again
wire_holograms_max		Defines the maximum number of hologams a player can have at once
wire_holograms_size_max		Defines the maximum size of holograms according to the holoScale function

Commands

Function	Returns	Description
holoEntity(N)		Returns the entity corresponding to the hologram given by the specified index.
holoIndex(E)		Returns the index of the given hologram entity.
holoCanCreate()		Returns 1 when holoCreate() will successfully create a new hologram until the Max limit is reached Replaces holoRemainingSpawns()
holoRemainingSpawns()		Returns how many more holos you can create.
holoCreate(N,V,V,A,V)		Index, Position, Scale, Angle, Color (RGB) Creates a new hologram entity
holoCreate(N,V,V,A)		Index, Position, Scale, Angle Creates a new hologram entity
holoCreate(N,V,V)		Index, Position, Scale Creates a new hologram entity
holoCreate(N,V)		Index, Position Creates a new hologram entity
holoCreate(N)		Index Creates a new hologram entity
holoDelete(N)		Index Removes the hologram with the specified index
holoDeleteAll()		Removes all holograms of the owner
holoScale(N,V)		Index, Scale Scales a hologram by a scale factor in each direction given by a vector
holoScale(N)		Index Returns the scale of the given hologram
holoScaleUnits(N,V)		Index, Scale Scales a hologram in each direction according to Garry's Mod units, given by a vector
holoScaleUnits(N)		Index Returns the scale of the given hologram
holoPos(N,V)		Index, Position Sets the position of a hologram

holoColor(N,V,N)		Index, Color, Alpha Changes the color and alpha of a hologram
holoColor(N,V)		Index, Color Changes the color of a hologram
holoAlpha(N,N)		Index, Alpha Changes the alpha of a hologram
holoShadow(N,N)		Index, Shadow Set to 0 to remove a hologram's shadow, 1 to display shadow
holoAng(N,A)		Index, Angle Sets the angles of a hologram
holoModel(N,S,N)		Index, Model, Skin Changes the model of a hologram (see the model list below) as well as the skin number
holoModel(N,S)		Index, Model Changes the model of a hologram (see the model list below)
<del>holoModelAny(N,S)</del>		<del>Index, Model</del> Use holoModel(N,S) with UWSVN installed and wire_holograms_modelany set to 1 (default 0)
holoSkin(N,N)		Index, Skin Changes the skin of a hologram
holoMaterial(N,S)		Index, Material Changes the material of a hologram
holoBodygroup(N,N,N)		Index, GroupID, SubID Sets the bodygroup of hologram
holoBodygroups(N,N)	N	Index, SubID Returns how many groups are in the given SubID.
holoDisableShading(N,N)		Index, Shading A value of 1 disables all shading on the hologram. The hologram will be entirely the same colour regardless of lighting conditions.
holoRenderFX(N,N)		Index, Render FX # Changes the RenderFX for a hologram
holoParent(N,N)		Index (current Holo), Index (Holo being parented to) Attaches a hologram to another hologram
holoParent(N,E)		Index, Entity Attaches a hologram to an entity
holoParentAttachment(N,E,S)		Index, Entity, AttachmentID Attaches a hologram to an attachmentID on an entity
holoUnparent(N)		Index Removes any parenting associations from a hologram
holoClipEnabled(N,N)		Index, Enabled. Enables / disables clipping for a hologram
holoClipEnabled(N,N,N)		Holo Index, Clip Index, Enabled. Enables / disables clipping for a hologram. Clip index is for use with multiple clipping planes
holoClip(N,V,V,N)		Index, Position, Direction, isGlobal. Defines where a hologram is clipped. isGlobal to choose between local and world vectors
holoClip(N,N,V,V,N)		Holo Index, Clip Index, Position, Direction, isGlobal. Defines where a hologram is clipped. isGlobal to choose between local and world vectors. Clip index is for use with multiple clipping planes
holoClipsAvailable()	N	Returns the maximum number of clipping planes allowed per hologram
holoVisible(N,E,N)		Index, Entity, Number Visible. Enables / disables hologram N's visibility to E.

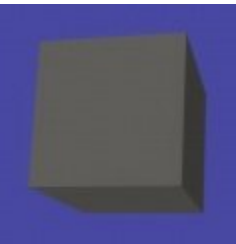
holoVisible(N,R,N)

Index, Array, Number Visible.  
Enables / disables hologram N's visibility to all players in R.

- List of holo models



cone



cube



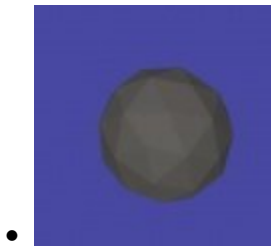
cylinder



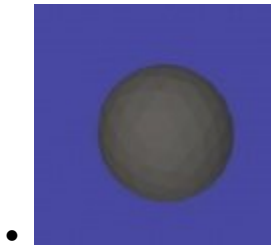
hexagon



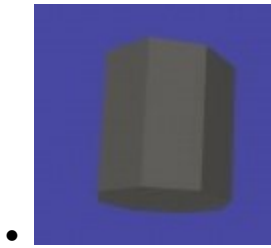
icosphere



icosphere2



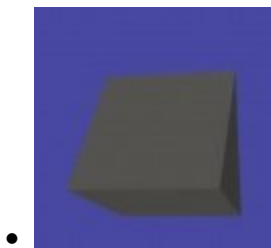
icosphere3



octagon



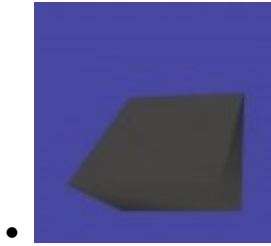
plane



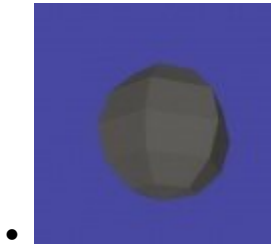
prism



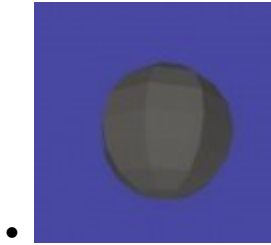
pyramid



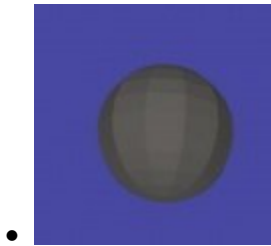
right\_prism



sphere



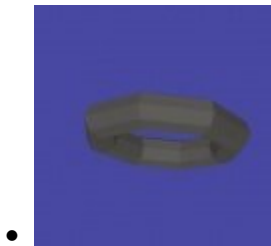
sphere2



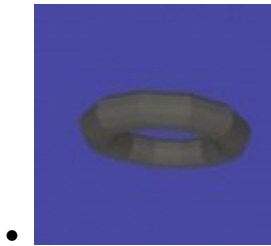
sphere3



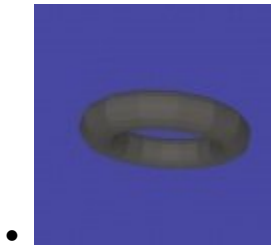
tetra



torus



torus2



torus3

- List of high quality holo models



hq\_cone

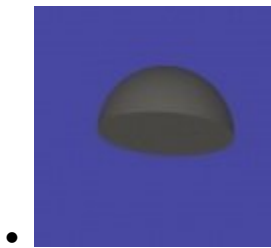


hq\_cubinder

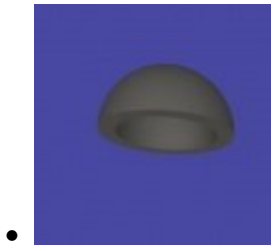




hq\_cylinder



hq\_dome



hq\_hdome



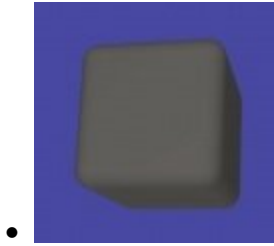
hq\_hdome\_thick



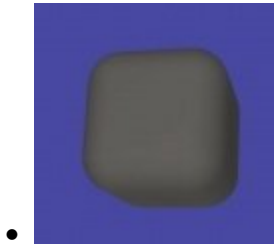
hq\_hdome\_thin



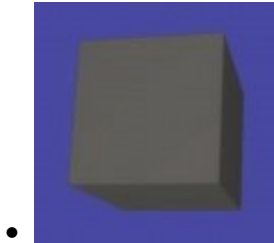
hq\_icosphere



hq\_rcube



hq\_rcube\_thick



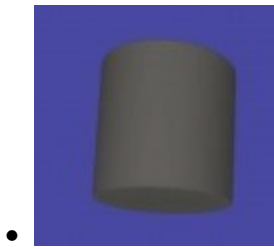
hq\_rcube\_thin



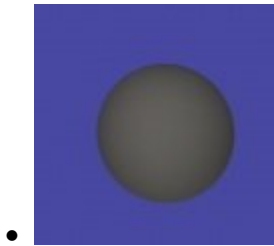
hq\_rcylinder



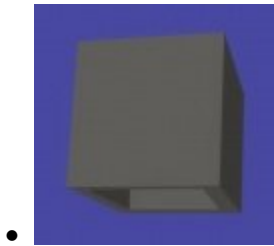
hq\_rcylinder\_thick



hq\_rcylinder\_thin



hq\_sphere



hq\_stube



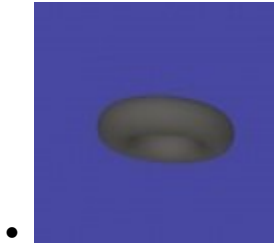
hq\_stube\_thick



hq\_stube\_thin



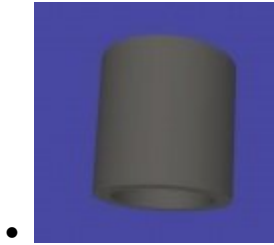
hq\_torus



hq\_torus\_thick



hq\_torus\_thin



hq\_tube



hq\_tube\_thick



hq\_tube\_thin

## File Functions

Developed by: McLovin

Description

The file functions are used to write text to files, stream files from client to server, and edit files.

**Important:** All the filenames must end in \*.txt!

Files are loaded from the "garrysmod/data/e2files" folder.

You can also load files from one of these other folders by prefixing the file name with ">special-folder-name/" (without quotes).

- e1shared = garrysmod/data/ExpressionGate/e2shared
- e2shared = garrysmod/data/Expression2/e2shared
- cpushared = garrysmod/data/CPUChip/e2shared
- gpushared = garrysmod/data/GPUChip/e2shared

*As of 25/10/2010, this is a new system for file access. If you have not updated Wiremod very recently, update it to use these new functions.*

**Warning:** Double quotation marks (") are replaced with single quotation marks (') in uploaded files. This is not intended behavior.

fileWrite() and fileAppend() actions have a 200ms cooldown.

Files cannot contain a null byte (0x00). Binary files lacking a null byte can be loaded and parsed, but there are no functions to assist with that in E2.

Commands

Function	Returns	Description
fileLoad(S)		Loads the file from the client and sends it to server (You must wait at least 10 seconds before uploading to server and there is a file size limit - see fileMaxSize).
fileCanLoad()	N	Returns whether a file can be uploaded right now.
fileLoaded()	N	Returns whether a file has been uploaded and can now be read.
fileLoading()	N	Returns whether a file is currently uploading.
fileStatus()	N	Returns the status of the upload in progress. Returns one of _FILE_UNKNOWN, _FILE_OK, _FILE_TIMEOUT, _FILE_404 or _FILE_TRANSFER_ERROR.
fileName()	S	Returns the name of the last uploaded file, or an empty string if there is no currently uploaded file.
fileRead()	S	Returns the contents of the last uploaded file, or an empty string if there is no currently uploaded file.
fileMaxSize()	N	Returns the maximum file size that can be uploaded or downloaded. Default is 100 KiB.
fileCanWrite()	N	Returns whether a file can be written or appended right now.
fileWrite(S,S)		First argument is filename, second is content. Sends a file to the client and saves it.
fileAppend(S,S)		First argument is filename, second is content. Sends the data to the client, and adds it to the end of an existing file.
runOnFile(N)		Specifies whether the E2 will run when a file finishes uploading.
fileClk()	N	Returns whether the E2 is running because a file finished uploading.
fileClk(S)	N	Returns whether the E2 is running because the specified file finished uploading.

HTTP Functions

Developed by: McLovin

Description

The HTTP functions are used to recieve data through HTTP. Only one request can be made at a time and a delay between requests is also in place to stop spamming of functions. The HTTP functions can be increadibly useful when used with an online http api. For a list of online apis visit [here](#).

Console Variables

Function	Default	Description
wire_expression2_http_delay	3	Allows you to set the delay between the last request and the new one, in seconds.
wire_expression2_http_timeout	15	Allows you to set the time until a connection times out.

Commands

Function	Returns	Description
HttpRequest(S)		Starts a new request.
HttpCanRequest()	<b>N</b>	Returns whether you can make a new request (delay has been met or previous request timed out).
HttpData()	<b>S</b>	Returns the data received from the last request.
HttpRequestUrl()	<b>S</b>	Returns the url of the last request.
HttpUrlEncode(S)	<b>S</b>	Returns formatted string to be placed in the url.
HttpUrlDecode(S)	<b>S</b>	Returns decoded url data.
runOnHTTP(N)		Sets whether to run the expression when a request finishes.
httpClk()	<b>N</b>	Returns whether the execution was run because of a completed request.

Bitwise

**Developed by:** asiekierka, TomyLobo (made xor), and Divran (added operators)

Description

Bitwise is carried out at a binary level. The individual bits of a number have operations taken against them. Wikipedia has a good explanation of each function: [http://en.wikipedia.org/wiki/Bitwise\\_operation](http://en.wikipedia.org/wiki/Bitwise_operation)

Tip: You can also use N:toString(2) and S:toNumber(2) to perform bitwise manipulation using the string functions.

Commands

Function	Returns	Description
bOr(N,N)	<b>N</b>	Performs bitwise OR against the two numbers (Operator: N    N)
bAnd(N,N)	<b>N</b>	Performs bitwise AND against the two numbers (Operator: N && N)
bXor(N,N)	<b>N</b>	Performs bitwise XOR against the two numbers (Operator: N ^^ N)
bShl(N,N)	<b>N</b>	Performs bitwise shift left on the first number by the amount of the second (Operator: N << N)
bShr(N,N)	<b>N</b>	Performs bitwise shift right on the first number by the amount of the second (Operator: N >> N)
bNot(N)	<b>N</b>	Performs a binary Not (Operator: Doesn't have one)
bNot(N,N)	<b>N</b>	Performs a binary Not. The second argument is the length of the number you wish to perform Not on in bits. (Operator: Doesn't have one)

E2 Variable Scopes

**Developed by:** Rusketh

Description

Variables in E2 can now be assigned to scopes, unlike previous versions of E2 where all variables were global.



A global variable is one that can be accessed from every point inside the E2 code, unlike local variables which can only be accessed from inside the block they are declared.

When your E2 code starts to get complex, local variables are a useful way to insure that only one block of code has access to its own set of variables. This prevents scripting errors when one block of code inadvertently modifies variables used by another block of code.

All variables defined as either an Input, an Output or a Peristant variable will always be in the global scope. Custom functions will run inside their own environment meaning they only have access to global variables and variables defined inside their own scopes.

Example

```
Variable = vec(20,30,40) #This is a Global variable

if ( Variable ) {
    local Variable = "Test" #This is a local variable
}
```

Include directive

Developed by: Divran, Rusketh

Description

This extension allows you to split up an E2 into several files and then include the code in those files in your main E2, which can help keep things more organized when your expressions gets more complex.

Syntax	Example	Description
#include ""	#include "myincludes/file"	The include line will be the location where the referenced E2 will be executed. Note that .txt is omitted.

Apart from global variables, everything in the included E2 is run in their own environment. Also, all of the directives, except for name and model, is merged with the main E2.

Remember to *always* use include inside an "if (first())" (or similar) statement.

When reading an E2 that uses the include directive, you get the option to download just the main E2 or everything. Keep in mind that if you want to update the E2, you have to save the expressions in the same location as where the E2 expects to find them. If you don't do this, you will get a file not found error.

Due to the nature of the include directive, it can only take a constant string, and not a variable.

Example

```
main.txt:

@name Include e2 main

#include "lib/functions"

includedFunction();

lib/functions.txt:

@name Include e2 example

function includedFunction(){
    print("Hello includes!")
}
```

See Also

- [Expression 2 Guide](#)
- [Expression 2 syntax highlighting for Notepad++](#)
- [E2Edit, a stand alone expression 2 editor. By itsbth](#)

## Credits

I would like to extend thanks to all of the following people who have made contributions to Expression 2 in one way or another, making it into what it is today.

**Shandolum, ZeikJT, Jimlad, Beer, Magos Mechanicus, Gwahir, chinoto, pl0x, Turck3, Ph3wl, Hunter234564, Fishface60, GUN, Bobsymalone, TomyLobo, Tolyzor, Jeremydeath, I am McLovin, Fizyk, Divran, Rusketh**

And of course all you others out there who use it, provide constructive feedback or help others become familiar with it!

Thank you! // Syranide

*P.S. I'm sorry if I forgot to mention someone!*

Retrieved from "[http://wiki.wiremod.com/w/index.php?title=Expression\\_2&oldid=924](http://wiki.wiremod.com/w/index.php?title=Expression_2&oldid=924)"

[Category](#):

- [Expression 2](#)

### Personal tools

- [Log in](#)

### Namespaces

- [Page](#)
- [Discussion](#)

### Variants

### Views

- [Read](#)
- [View source](#)
- [View history](#)

### Actions

### Search

Go

Search

### Navigation

- [Main page](#)
- [Wiremod.com](#)
- [Recent changes](#)
- [Random page](#)

### Quick links

- [Tools list](#)
- [Gates](#)
- [Expression 2](#)
- [UWSVN](#)

#### Toolbox

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)

#### Google AdSense

- This page was last modified on 25 February 2015, at 17:23.
- This page has been accessed 821,162 times.
- Content is available under [GNU Free Documentation License 1.3 or later](#).
- [Privacy policy](#)

- [About Wiremod Wiki](#)
- [Disclaimers](#)

