WIREMOD

| Home | Forum | Blogs | What's New? |

Today's Posts   FAQ   Calendar   Community ▾   Forum Actions ▾   Quick Links ▾   Advanced Search

Forum ▸ Tool Help & Discussion ▸ Expression 2 Discussion & Help ▸ An Intermediate Guide to Expression 2

If this is your first visit, be sure to check out the **FAQ** by clicking the link above. You may have to **register** before you can post: click the register link above to proceed. To start viewing messages, select the forum that you want to visit from the selection below.

+ Reply to Thread

Results 1 to 10 of 102   Page 1 of 11   **1** 2 3 ... ► Last ▸▸

Thread: An Intermediate Guide to Expression 2

LinkBack ▾   Thread Tools ▾   Display ▾

04-18-2009   #1

Matte ○
Developer
WIREMOD WM DEVELOPER
Matte's Avatar
Join Date: Jan 2009
Location: Norway
Posts: 3,103

💡 **An Intermediate Guide to Expression 2**

I have been asked a few times (understatement of the year) if I could teach people the basics, bits and pieces of the Expression 2
Yes, I know a lot of people thinks the Expression 2 is hard, and I also thought it was impossible in the beginning. I am going to break it down for you guys, and hopefully make it easier. For the purpose of this guide I will just refer to the Expression 2 as E2.

This guide won't get updated as often as the guide on the Wiki. I suggest reading that, as this may get outdated from time to time.

Wire Expression2:Guide - GMod Wiki

Table of Contents

Basic Arithmetics
Addition
Subtraction
Multiplication
Division
Exponantion
Square Root
Cube Root
PI

Round Down
Round Up
Round
Random & RandInt
Abs
Int & Frac
Clamp
Inrange
Sign
Min & Max

Basic Statements
Conditional Statements

Intermediate Statements
else
elseif
| (or)
& (and)
! (not)

Advanced Statements
$ (Delta)
interval()
~ (Input-triggered execution?)
clk()

@trigger and @persist
@trigger
@persist

Basic Command Syntax

Self-Aware
print() and hint()
Execute Console Commands
Changing Colors

Vectors
Explaining Vectors
Making Vectors in Gmod
Using applyForce()

Angles
Angles in Garry's Mod
Using ApplyAngForce()

Ok, here we go.

Basic Arithmetics
Calculating math is what the E2 is most widely used for. Im gonna give you examples on how to calculate basic arithmetics
in the E2.

Addition
Addition is the same as plus. To add in the E2 simply use the plus sign(+).
Example:

Code:
```
N = X + Y
```

Subtraction
Subtraction means minus. To subtract in the E2 we use the minus sign(-).
Example:

Code:
```
N = X - Y
```

Multiplication
Multiplication is probably the most widely used operation in the E2. It uses the star sign (*).
Example:

Code:
```
N = X * Y
```

Division
Division is also a widely used mathematical operation in the E2. It uses the slash (/)
Example:

Code:
```
N = X / Y
```

Exponention
Exponention is not so widely used. It means how many times the argument is multiplied by itself.
Example:

Code:
```
N = X ^ 3
Could also be written
N = X*X*X
```

Square Root
The square root of a value is how many times it must be multiplied by itself to get the value. It uses the square root command(sqrt()). It is hard to explain, so I will give you some examples.

Code:
```
sqrt(9) = 3    because 3 * 3 = 9
sqrt(16) = 4     because 4 * 4 = 16
```

Cube Root
Cube root is basically the same as square root, execept it returns the number which must be multiplied by it self *three* times to get the main value. It uses the cube root command (cbrt()).
Example:

Code:

```
cbrt(27) = 3    because 3 * 3 * 3 = 27
cbrt(64) = 4     because 4 * 4 * 4 = 64
```

pi
Pi is the ratio between a circles diameter and its circumference. If a circle's diameter is 1, it's circumference is *pi*. It uses the pi function (pi()).
Pi is approximately equal to 3.14159265358979323846264.
Example:

Code:
```
Circumference = CircleDiameter * pi()
```

Round Down
The round down command (floor()), is used to floor a value. I.e floor(2.71828) would be 2.
Example:

Code:
```
floor(pi()) = 3
floor(e()) = 2

where
pi() = approx. 3.14
e() = approx. 2.71
```

Round Up
Round up (ceil()) is the same as floor, except that it rounds up.
Example:

Code:
```
ceil(pi()) = 4
ceil(e()) = 3

where
pi() = approx. 3.14
e() = approx. 2.71
```

Round
Round command (round()) will round the value to the nearest integer. 1.499 or less would equal 1, while 1.5 or larger would equal 2.
Example:

Code:
```
round(pi()) = 3
round(e()) = 3

where
pi() = approx. 3.14
e() = approx. 2.71
```

Random and RandInt
The random command (random(x,y)) will return a random value between argument 1 (here x) and argument 2 (here y).
Example:

Code:
```
random(0,1)
```

```
will get a random value between 0 and 1.
```

RandInt will get a random integer between two numbers. this means, while random(x,y) may return
4.62648323979853562951413, randint(x,y) will only return rounded integers.

Abs
Abs returns the magnitude of the argument, which means that if the number is less than 0, it will be multiplied by -1, so it
will always have a positive value.
Example:

Code:
```
abs(10) = 10
abs(-3) = 3
```

Int and frac
Int returns the integer part of an argument, and frac returns the fractional part. This means that if the number is 3.14, int
will return 3, and frac will return 0.14.
Example:

Code:
```
int(2.718) = 2
frac(2.718) = 0.718
```

Clamp
Clamp "clamps" a value between two other values. If a value is clamped, it will always be greater than or equal to the first
argument, and always less than or equal to the second.
Example:

Code:
```
clamp(I,1,10) is the same as
if(I<1){I = 1} if(I>10){I = 10}
```

Inrange
Inrange returns 1 if the argument entered is in the value range entered.
Example:

Code:
```
inrange(I,1,10) returns 1 if I is larger than 1, but less than 10.
```

Sign
Sign returns -1 if the argument is less than zero, 0 if it is zero, and 1 if it's larger than zero.
Example:

Code:
```
sign(-255) = -1
```

Min & Max
Min returns the argument of the two entered, with the lowest value.
Example:

Code:

```
min(-255,255) = -255
```

Max returns the argument of the two entered, with the greatest value.
Example:

Code:
```
max(-255,250) = 250
```

Basic Statements

The syntax of the E2 is actually pretty simple and also pretty simular to both E1 and C/C#/C++ . For the first examples I will post both the E1 syntax and the E2 syntax, so all of you guys that already know a bit of E1 can learn the basics of E2. To start off with, here's a legend for the different types of operators in a conditional statement (if-statement).

Code:
```
==      Is it equal to?
!=      Is it not equal to?(Is it anything alse, but?)
<       Is it less than?
>       Is it greater than?
<=      Is it less than or equal to?
>=      Is it greater than or equal to?
```

In this example we will make a number increment at the push of a button.
In E1 we would write it like this:

Code:
```
I@ Button
O@ Output
Line1: If Button == 1 -> Output +=1;
```

While in E2 we would have written:

Code:
```
@inputs Button
@outputs Output

if(Button){Output++}
```

I am going to break it down for you guys.

if(Button)
This is the same as "If" in E1, except you put the arguments inside a parentheses.
This function means "if Button is equal to one, then proceed to *then*". You may have noticed that I only wrote the name of the variable incide the parentheses. This is because when the E2 finds the name of a variable inside the "if()" paretheses, it will default to checking if that variable is *true*. It will check if the variable is equal to anything above zero. We could wery well have written if(Button==1), but it is longer and in E2 you only have to put a variable inside an "if()"-function, and it will ask itself "Does this variable equal anything above zero??"

{Output++}
Like the arrow (->) in E1, the **curly brackets** ({}) means *then* in E2.
In this example I have chosen the C++ way of incrementing a number. We could also have written:

Code:

```
{Output+=1}
```

Which then is exactly the same as:

Code:

```
{Output=Output+1}
```

The only thing (I wouldn't call it a *flaw*) with the ++ command is that you can't set how much to increment the variable by.
It will always increment the variable by the default 1.

Intermediate Statements

For more possibilities you can start using "else", "elseif" , "|" (or), "&" (and) and "!" (not).
To give you an example:

Code:

```
@inputs Button1 Button2
@outputs DoorLength Screen
interval(100)

if(Button1 | Button2 & DoorLength >40){DoorLength=0}

elseif(Button1 | Button2 & DoorLength <1){DoorLength=41}

else{Screen=1}
```

Normally I would not have spaces between the *or*, *and* and *greater/less than* functions. I just did it for the purpose of this example to give a better overview.

Here you can see, I have used both "|"(or) and "&"(and).
The first if-statement would return true if Button1 *or* Button2 is pressed, meanwhile, DoorLength has to be *greater than* 40.
Otherwise, it would return zero, and the expression would run to the elseif-statement. You can see the elseif statements this way: If the first if returns true, it will not run the elseif-part. And if the first if returns false, it will run the elseif-statement as if it was a normal if-statement. If the elseif-statement returns false, the expression will go on to the else-statement, and to what it's told by that statement.
Here, as the first if-statement, Button1 *or* Button2 must be pressed, however, DoorLength has to be *less than* 1. If none of these statements returned true, then the expression would wait until one of the if's were true.

To show you the use of "!" (not) I'm gonna make a new example.

Code:

```
@inputs Button
@outputs Screen
interval(100)

if(!Button){Screen=1}
else{Screen=0}
```

In this example we use the function "not". Like I said earlier, if you just input a variable in the if-parentheses, the if()-statement will check if that variable is equal to one. Hovewer in this example it will check if the variable (Button) is *not* equal to 1. In other words, it checks if the variable is anything but 1.

So in this example, the screen would show the number 1 until Button is pressed. Then the screen would show 0 until the button is released again.

Advanced Statements

To really benefit from the E2 to your contraptions you will need "interval()", "clk()", "$" (delta) and "~" (Input-triggered execution?)

$ (Delta)

Delta is an operator that can calculate the change of a variable since the last execution. This means that if A=1 at a given time, and A=2 at the next execution, the delta of A ($A) would equal 1. An example to clear things up:

Code:
```
Execution 1 - (In1 = 1): $In1 = 1
(In1 has currently changed to 1 from 0, which all variable are at the first run. $In1 is now 1)

Execution 2 - (In2 = 1): $In1 = 0
(In1 is still 1, therefore delta of In1 = 0)

Execution 3 - (In1 = 0): $In1 = -1
(In1 has now changed to 0 again. It has decremented by one value. $In1 is now -1)
```

I hope this explains what delta is in an easy way.

interval()
You may have noticed that many of my examples include a function called "interval".
Interval determins how many milliseconds the expression will wait until it reads the code again. Example:

Code:
```
@inputs Button
@outputs Screen
interval(100)

if(Button){Screen++}
```

In this example, the screen will get a number added every 100ms whenever the button is pressed. That equals 10 numbers per second.

~ (Input-triggered execution?)

The "~" sign is used to check wether the current execution was caused by an input changing. It is useful when you, for instance have an interval, but don't want your output to increase with the interval.
I will give you an example:

Code:
```
@inputs Button
@outputs Screen
```

```
interval(10)

if(~Button&Button){Screen++}
```

In this if-statement, it asks itself: "Has Button changed?" and "Is Button equal to 1?". Now the screen would only increment if the button is pressed, regardless of the interval.

clk()
As you probably already know, the E2 reads the code again if the interval tells it to, or if an input changes. Unlike the "~"-sign, which could be useful when ignoring interval(), the clk is useful for ignoring input changes. Let me give you an example.

Code:
```
@inputs Time Button
@outputs Screen1 Screen2
interval(1000)
Screen1=time*10
if(clk())
{
if(Button){Screen2++}
}
```

Here, imagining a timer is hooked up to the "Time"-input, Time is an input that is always changing. Since our interval is so large, we wouldnt want our second screen to add every time the "time"-input changes, so here is the smart thing about the clk()-function I added if(clk()), which means "If this is triggered by the interval". I use this for a lot of stuff, for instance hints and chat messages, which i will talk about later.

@trigger and @persist
You may have noticed that when clicking "New Expression", the expression will automatically make a template looking like this:
Code:
```
@name
@inputs
@outputs
@persist
@trigger all
```

Now, the @name, @inputs and @outputs are self-explanatory, however, the @persist and @trigger, are not. I am going to explaing it for you here.

@trigger
The @trigger-directive can enable or disable selected inputs from triggering executions. I'm going to use an example I have used earlier:
Code:
```
@inputs Time Button
@outputs Screen1 Screen2
interval(1000)
Screen1=time*10
if(clk())
{
if(Button){Screen2++}
}
```

In that example I used the "~"-operator to stop the screen from incrementing because time was an input that was constantly changing.
By using the @trigger-directive we could have had a lot shorter expression. It would look like this:

Code:

```
@inputs Time Button
@outputs Screen2
@trigger none
interval(1000)
if(Button){Screen2++}
```

By setting @trigger to none, we told the expression not to execute from input changes, but what if we had an input we wanted it to execute from at the same time as "Time" was constantly changing?

Code:

```
@inputs Time Button
@outputs Screen2
@trigger Button
interval(1000)
if(Button){Screen2++}
```

Then we would write this to the @trigger-directive. Now the code would only execute with the interval, or if Button has changed.
The @trigger-directive is set to "all" by default. This means the expression will trigger from all input changes.

@persist
The @persist-directive is for remembering a value inside the expression. If we wanted to increment a value internally (not using the value as an output), and we didnt use the @persist-directive we would get the error "Cannot perform operation on a temporary value". Let me give you an example:

Code:

```
@inputs Button Rate
@ouputs Screen
interval(100)
if(~Button&Button){TempVal++}
Screen=TempVal*Rate
```

This would give us an error. The solution to the problem? Add TempVal to the @persist-directive. The code would now look like this:

Code:

```
@inputs Button Rate
@ouputs Screen
@persist TempVal
interval(100)
if(~Button&Button){TempVal++}
Screen=TempVal*Rate
```

Basic Command Syntax

Say you had a target finder and you want to get the name of your target. Here I am gonna explain some of the most basic [ENTITY]-commands.

To start off with, I'm gonna make an example where you get the name of an entity for instance from a target finder.

Code:
```
@inputs Target:entity
@outputs Name:string
interval(10)

Name = Target:name()
```

In this example you can see the input was "Target:entity". Here you may understand that the input is of the type "entity". If it was a normal numeric input, we could have written "Input", but since the entity-commands only takes entity-data, we will need to define that the input contains entity-data. If we tried to wire a normal input to an entity output we would get the error "Data type mismatch, Input takes NORMAL, output gives ENTITY!"

You can also see that the Name-output has the data-type "string". Now, string may mean nothing to you, but it's actually pretty simple. Just like a normal output gives numbers, a string-ouputs will give letters, words and sentences. A string could, for instance, be "Hello, World!". Just remember, you can't wire a string, or entity input/output to a normal numeric input/output.

Now, you can see I have set "Name" equal to "Target:name()". This is because the "name()" command outputs the name of the entity-input it's attached to via the colon (:).

I'm gonna give you another example to hopefully clear things up a bit.

Code:
```
@outputs Owner:string

Owner=entity():owner():name()
```

In this example, you can see I have also used the "name()"-command, but I have also used two new commands, the "entity()" and the "owner()" command.

Like I said earlier, the "name()" command outputs the name of the entity it is attached to via the colon, but here the entity is the entity():owner().
The "entity()"-command points out the entity of the expression 2 chip, and the "owner()"-commands points out the owner of the entity its attached to via the colon.
So "entity():owner()" would be the owner of the E2. You.
The output "Name:string" would now contain your name in string format.

Now onto how to wire a string output from the E2 to a wired text screen.
The wired text screen has by default 6 inputs, Clk, Text, StringClk, String [STRING], StringNum and Value 1. What we are gonna do is wire everything except String[STRING] and Value 1 to a Constant Value of 1. Then wire the String[STRING] input to the Name[STRING] output of the E2. The text screen will now display the string you outputted from the E2.
Here are more commands you can use to get string(text) output:

Code:
```
Entity:name()              Gets the name of a player
Entity:steamID()             Gets the steam ID of the player
Entity:type()              Gets the class of an entity
Entity:model()              Gets the model of an entity
```

And here are some commands that return entities:

Code:
```
Entity:aimEntity()             Gets the entity that the entity is aiming at
entity(Number)                 Gets the entity associated with the id
entity()                       Gets the entity of the expression
```

```
Entity:owner()                    Gets the owner of the entity
```

Self-Aware

Self-Aware in an expression is commands the expression can physically do by itself. For instance, with the expression 1, you had to have an external thruster to influence the physics in Garry's Mod. With the E2, however, you can apply thrust from the chip itself, print messages in the chat, execute console commands and much more.

print() and hint()

To print messages into the chat, we must first use the "print()"-command, then specify what to print. The arguments the print()-command takes must be formatted as a string, i.e. text with quotes as in "Hello, World!". I'm gonna give you an example on how to print messages using the print()-command:

Code:
```
@inputs Print

if(Print)
{
print("Hello, World!")
}
```

This code would print the sentence "Hello, World!" in the chat. However, now that the string data-type is properly implemented in Garry's Mod, we could actually input what to print in the chat. Example:

Code:
```
@inputs Print Text:string

if(Print&Text!="")
{
print(Text)
}elseif(Text=="")
{
hint("You have not inputted a text to print!",7)
}
```

In this example, you can see I have told the expression to print the inputted text *if* a text is inputted. If no text is inputted I told the expression to *hint* the owner "You have not inputted a text to print" for seven seconds. hint() works in the same way as print() exept, it is just showing up for the owner, and you need to specify how many seconds the message should appear. Hints will show up on the right of your screen as little text bubbles.

Execute Console Commands

In the E2 you can also execute console commands. Although this will only work if wire_expression2_concmd is enabled, a command which is disabled by default.
To execute console commands we use the command "concmd(<command>)". Im gonna give you an example:

Code:
```
@inputs EnableCheats DisableCheats

if(EnableCheats)
{
concmd("sv_cheats 1")
}
if(DisableCheats)
{
concmd("sv_cheats 0")
}
```

This would enable cheats in singleplayer, or if you are an admin in multiplayer.

While the print()-command print blue messages in the chat, you can make yourself say things automatically, by using concmd("say <msg>").
An example:

Code:

```
@inputs Say Message:string

if(Say&Message!="")
{
concmd("say "+Message)
}elseif(Say&Message=="")
{
hint("You have not inputted text to say!,7")
}
```

In this example, you would say the message inputted, if it is *not equal* to "", which is nothing. If it was equal to "", it would hint you that you have not inputted a text to say.

Changing Colors
Something new about the E2 is that you can program color changes. To change the color of an entity, we use E:setColor(N,N,N), where the E stands for which entity to change color, and the N's are the RGB values. Now you may wonder what RGB is. RGB stands for Red, Green and Blue. This color method is *subtractive*, which means we subtract colors from black to get the color we want. If we wanted to get red, we would simply max the red value and leave the green and blue value to zero. It would look like this:

Code:

```
@inputs Button

if(Button){
entity():setColor(255,0,0)
}
```

Here's a list over different colors:

Code:

```
Black - (0,0,0)
White - (255,255,255) (Although, to make it visible I made it (180,180,180))
Red - (255,0,0)
Green - (0,255,0)
Blue - (0,0,255)
Yellow - (255,255,0)
Magenta - (255,0,255)
Cyan - (0,255,255)
Orange - (255,180,0)
```

But say if I wanted to make my prop invisible. Then we need to add another value after the RGB, called *alpha*. Aplha is the opacity (see-through-ness-ish) of the color. Alpha is defaulted at 255, which is maximum opacity. If we would want an invisible prop, we would need to set alpha to zero. It would look like this:

Code:

```
@inputs Button

if(Button){
entity():setColor(255,255,255,0)
```

```
}
```

As you can see I've set the color to white(not really nescessary), and set opactiy to minimum. Our E2 would now be invisible.

Vectors
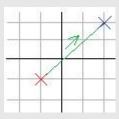
Explaining Vectors

To use vectors and applyForce (etc..) you will first need a basic understanding of what vectors are.
Vectors are basically a direction in a 2d(XY) or 3d(XYZ) space. To make a vector you will need two XY/XYZ co-ords. The first co-ordinate specify where the vector starts, and the second tells us where the vector is headed for. To create a vector example, I will use my mspaint skills to draw it up for you.

Say we had a 2D vector with the coordinates (-1,-1)(2,2). This would look like this when drawn up in a co-ordinate system:

As you can see, the red cross is the first XY(-1,-1) and the blue cross is the second (2,). The vector, however, is the *green line*, and it's direction is from point 1 to 2. You may notice that I left the Z co-ords to zero in both points. This is because it would be almost impossible to draw otherwise. This is another example that attemp to draw a 3D vector:
(Image taken from intmath.com)
In this example we have a vector from co-ordinate (0,0,0) to co-ordinate (2,3,5)

Now, that you hopefully understand a bit more about vectors, we move on how to use vectors within GMod.

Vectors in Gmod

When making a vector in Garry's Mod, we use the vec(X,Y,Z) command.
This command will always make the vector start at the local (0,0,0) co-ordinate, which in any case, is the E2, and it will make the vector end at the (X,Y,Z) you input.
Vectors are also used to store GPS locations. For instance, to store the owners position, we would use this code:

Code:
```
@outputs OwnerPos:vector

runOnTick(1)
OwnerPos=owner():pos()
```

Using applyForce()

The function vectors are probably most used for, is the applyForce()-function. This function is almost the same as a vector thruster, except it's built into the E2.
The arguments this function takes is what direction/vector to apply force against. To give you an example:

Code:
```
@name Vector Example

runOnTick(1)
applyForce(vec(0,0,1))
```

This example would constantly apply thrust against the vector (0,0,1). Remember the vector takes arguments (X,Y,Z), so (0,0,1) would be upwards. Hope this cleared up a bit.
Now onto how to make a E2 follow the owner (you).

To make it follow the owner, we must first get the owner's position. We get this by using the owner():pos()-command. One thing though. The owner's postiton is stored as GPS co-ordinates, which is co-ords according to the center of the map, not the E2. See it like this:
If the owner stands at the center of gm_flatgrass, his GPS position would be (0,0,0). However, his position according to the E2 will be different depending on where the E2 is on the map. If the E2 were 1000 units underneath him, his local position according to the E2 would be (0,0,1000), while the GPS position is still (0,0,0). To use GPS-positions within the E2, we will firstly need to convert them to the E2's local co-ordinates.

To convert from world to local co-ordinates, we first need the world position of the entity we are converting as local position (0,0,0), which in this case is the E2 chip. Then we need to subtract the E2's position from the target position. Example:

Code:
```
runOnTick(1)              #This will update the position every game tick.
TargetPos=owner():pos()   #The position the E2 will thrust towards.
CurrentPos=entity():pos() #Current position of the E2.
Vector=TargetPos-CurrentPos #The vector from the E2 to the target position.
```

As we got our vector, we're only missing the applyForce()-part. This is very easy, as it is just to insert the vector we got into the argument parenthaces.
Example:

Code:
```
runOnTick(1)              #This will update the position every game tick.
TargetPos=owner():pos()   #The position the E2 will thrust towards.
CurrentPos=entity():pos() #Current position of the E2.
Vector=TargetPos-CurrentPos #The vector from the E2 to the target position.
applyForce(Vector)        #Applies force to the inserted vector.
```

This E2, would follow it's owner. But as you probably will notice if you go into noclip, the expression is very wobbly. To get it less wobbly we just add a delta calculation to the applyForce()-line. Example

Code:
```
@persist Vector:vector
runOnTick(1)              #This will update the position every game tick.
TargetPos=owner():pos()   #The position the E2 will thrust towards.
CurrentPos=entity():pos() #Current position of the E2.
Vector=TargetPos-CurrentPos #The vector from the E2 to the target position.
applyForce($Vector*10+Vector)#Applies force to the delta of the vector*5 plus the vector.
```

As you can see the expression now applies force towards the delta of the vector plus the vector. You may also notice that I have added the Vector:vector to the @persist line. This is because you cannot calculate delta on a temporary variable, since the delta operator compares the change between the last execution and now, we need the E2 to remember the Vector variable.

Angles
Angles in Garry's Mod

The format of angles in garry's mod is pretty similar to the vector format. An angle is made up of three numbers. The first number represents the pitch in the angle, the second represents the yaw, and the third and last represents roll. Think of

these angles like an airplane.

The pitch is the angle of the nose/back of the airplane. To increase the pitch angle, we would raise the nose and lower the back.

The yaw is like the axis of turning. To increse/decrease the yaw, we would move the nose to the left or right and he back to the opposite direction.

The roll is pretty much self-explanatory. if this angle increases, the airplane will roll like your dog does on the floor. Like pitch, but sideways.

I reckon you have used the gyroscope before? The angles in thE E2 works like that. I'm gonna make an example on how to make a gyroscope in the E2.

Code:

```
@name Gyroscope
@outputs Pitch Yaw Roll Angle:angle
runOnTick(1)
Angle=angles()
Pitch=Angle:pitch()
Yaw=Angle:yaw()
Roll=Angle:roll()
```

This code will output the pitch, yaw and roll angles, but also an angle output containing the three angles formatted as an angle.

Using ApplyAngForce

ApplyAngForce is used to apply angular force to the entity specified. When getting an angle from an E2, the angles goes from 180 to -180. The angles of a stabilized E2 is (0,0,0). Same as applyForce shoudl be multiplied by mass, you should multiply your angle by inertia so no matter what prop is used it will still rotate at the same rate, multiply your angle by sqrt((E:inertia():length())/3).

Code:

```
@persist Ang:angle
runOnTick(1)                          #Execute every server tick
Ang = -entity():angles()     #The angle of the E2

applyAngForce(Ang)
```

This example will probably spass, but the basic principle is easy to understand. It applies force to the negative angles of the E2, which means that if the E2's angles are 0,0,0, the applyAngForce function would not apply any force at all. However, if we tilt the E2 10 degrees on the pitch axis, the function will apply force to that axis with a force of -10 (The pitch of the E2 negated).

To make it stop spassing, we must add another argument to the angle calculation.

Code:

```
runOnTick(1)                           #Execute every server tick
Ang = -entity():angles():setYaw(0)     #The angle of the E2, with yaw set to 0
AngV = entity():angVel()               #The angular velocity of the E2
applyAngForce(Ang*25  - AngV*2)
```

There. You may notice that I sat yaw to 0. This is because we don't care how the prop turns on the yaw axis. I added the angVel() line because it gives us the angular velocity of the entity. This works pretty much like delta.

But what if we got a prop that weighs much, much more? Here's another argument to add to the calculation. The inertia. You should multiply your angle by inertia so no matter what prop is used it will still rotate at the same rate, multiply your angle by sqrt((E:inertia():length())/3), E again being entity() if you are forcing the expression.

Code:
```
runOnTick(1)
Ang = -entity():angles():setYaw(0)
AngV = entity():angVel()
Inertia = shiftL(ang(entity():inertia()))
applyAngForce(-(Ang*25  + AngV*2)*Inertia)
```

Here we multiply the angles ang angular velocity by the inertia of the prop. Inertia is the angular resistance of the prop.

If we combine this with the last expression from the vector tutorial, we could make the E2 follow us, while it always stays level. Here:

Code:
```
@persist Ang:angle AngV:angle Vector:vector
runOnTick(1)

EE   = entity
Ang  = EE:angles():setYaw(0)
AngV = EE:angVel()
Inertia = shiftL(ang(entity():inertia()))
applyAngForce(-(Ang*25  + AngV*2)*Inertia)
TargetPos=owner():pos()+vec(0,0,120)
CurrentPos=EE:pos()
Vel=EE:vel()
Vector=TargetPos-CurrentPos-Vel
Mass = EE:mass()
applyForce(Vector*Mass)
```

Last edited by Matte; 08-22-2009 at 12:32 AM.

Share

"If anybody says he can think about quantum physics without getting giddy, that only shows he has not understood the first thing about them."
-- Niels Bohr

Reply With Quote

04-18-2009   #2

Solece ○

Wire Sofaking

WIREMOD   MEMBER

Solece's Avatar
Join Date: Jul 2008
Location: Pittsburgh, PA
Posts: 671

re: An Intermediate Guide to Expression 2

Very nice, glad to see someone is finally making one.

Share

> 🔁 **Originally Posted by Ehmmett**
>
> *Ehmmett <3: Too much effort to play a game to fap.*

💬 *Reply With Quote*

📄 04-18-2009   #3

Matte ○

Developer

WIREMOD ᵂᴹDEVELOPER

Matte's Avatar

Join Date:	Jan 2009
Location:	Norway
Posts:	3,103

📄 **re: An Intermediate Guide to Expression 2**

Thanks. I had so many people asking me how I did stuff on my servers that I decided to make a tutorial covering the E2.^_^

Share

"If anybody says he can think about quantum physics without getting giddy, that only shows he has not understood the first thing about them."
-- Niels Bohr

💬 *Reply With Quote*

📄 04-18-2009   #4

Nicolai1 ○

Banned

WIREMOD BANNED

Sgt-Pac

Join Date:	Nov 2008
Location:	Denmark.
Posts:	1,251

📄 **re: An Intermediate Guide to Expression 2**

Nice tutorial, I bet a lot of people could use this (I also tried to make one, failed miserably ^^)

Share

💬 *Reply With Quote*

📄 04-18-2009   #5

Matte ○

Developer

WIREMOD ᵂᴹDEVELOPER

Matte's Avatar

Join Date:	Jan 2009
Location:	Norway
Posts:	3,103

📄 **re: An Intermediate Guide to Expression 2**

Thanks. The tutorial is still WIP, so I will be adding content whenever possible.🙂

Share

"If anybody says he can think about quantum physics without getting giddy, that only shows he has not understood the first thing about them."
-- Niels Bohr

Reply With Quote

📄 04-18-2009   #6

Echo51 ○

Wire Sofaking

Join Date: Feb 2009
Location: Denmark
Posts: 1,492

📄 **re: An Intermediate Guide to Expression 2**

You need to cover triggers 😛

Share

Proud of my non-existant life...

> In russia, Expression 2 codes you!

> 📖 Originally Posted by **Snowden42** ▶
> *Nooblishious? That's a signature keeper.*

Reply With Quote

📄 04-18-2009   #7

Lyinginbedmon ○

billywitchdoctor.com

WIREMOD   MEMBER

Join Date: Mar 2009
Location: England
Posts: 2,656

📄 **re: An Intermediate Guide to Expression 2**

Frak that, we need a tutorial covering entities and the entity discovery functions.

Share

Reply With Quote

📄 04-18-2009   #8

Matte ○

Developer

WIREMOD   WM DEVELOPER

Matte's Avatar
Join Date: Jan 2009
Location: Norway
Posts: 3,103

📄 re: An Intermediate Guide to Expression 2

That will be my next "covering". Thanks!

EDIT: The @trigger and @persist-directive is now covered. I will look more into explaing Entity-Discovery and Self-Aware soon.

EDIT2: Added some basic arithmetic commands.

Last edited by Matte; 04-18-2009 at 11:48 AM.

Share

"If anybody says he can think about quantum physics without getting giddy, that only shows he has not understood the first thing about them."
-- Niels Bohr

💬 *Reply With Quote*

📄 04-20-2009
#9

woo482 ○
Wirererer

WIREMOD   MEMBER

woo482's Avatar
Join Date: Jul 2008
Posts: 283

📄 re: An Intermediate Guide to Expression 2

this tutorial helped me learn E2, thanks 🙂

Share

💬 *Reply With Quote*

📄 04-20-2009
#10

eduardo ○
Wirererer

eduardo's Avatar
Join Date: Jan 2009
Posts: 399

📄 re: An Intermediate Guide to Expression 2

HEZUS CHRISTEH FINALLY an actual $&@#ing human is making a decent tutorial rather than us having the crapass complicated as hell examples to run off of

improvement ideas: anything that has to do with vectors,angles,applying(anything),and how to write a code that pertains to said subjects

THANK YOU FOR WRITING IT IN HUMAN 😊

Share

💬 *Reply With Quote*

+ Reply to Thread

Page 1 of 11 **1** 2 3 ... ▶ Last ▶▶

« Previous Thread | Next Thread »

LinkBacks (?)

G3 Expression 2 Tutorial	02-23-2010, 10:53 AM

Refback This thread

Expression 2 chip that kills anyone that owner wants - Facepunch
Refback This thread

02-10-2010,

Wire Expression2:Guide - GMod Wiki
Refback This thread

02-03-2010,

want wire applyforce/variation help? - Facepunch
Refback This thread

02-03-2010,

Gmod Wire Keypad (Addon) Help! - Facepunch
Refback This thread

01-29-2010,

My First Expression 2 help - Facepunch
Refback This thread

01-28-2010,

G3 Expression 2 Tutorial
Refback This thread

01-28-2010,

New to E2 :/ - Facepunch
Refback This thread

01-27-2010,

Wire Expression2 - GMod Wiki
Refback This thread

01-27-2010,

Similar Threads

Wiremod Svn Guide
By Cyborgmatt in forum Installation and Malfunctions Support

Replies: **604**
Last Post: **09-17-2009,**

Guide
By Conmanx360 in forum Installation and Malfunctions Support

Replies: **4**
Last Post: **05-05-2008,**

Thrusters Only Guide
By Bernie in forum Gate Nostalgia (Old School Wiring) Discussion & Help

Replies: **13**
Last Post: **04-08-2008,**

Wiremod Guide?
By Tjuven in forum Installation and Malfunctions Support

Replies: **14**
Last Post: **03-28-2008,**

need a real guide
By sh410 in forum Gate Nostalgia (Old School Wiring) Discussion & Help

Replies: **8**
Last Post: **11-15-2007,**

Bookmarks

- Digg
- del.icio.us
- StumbleUpon
- Google

Posting Permissions

You may not post new threads
You may not post replies
You may not post attachments
You may not edit your posts

BB code is On
Smilies are On
[IMG] code is On
HTML code is Off

Trackbacks are On

Pingbacks are On
Refbacks are On

Forum Rules

-- Wiremod Reborn