

The Wayback Machine - https://web.archive.org/web/20120910230924/http://www.wiremod.com/forum/expression-2-discussion-help/11041-beginners-guide-expression-2-a.html

WIREMOD

Sign in through STEAM

User Name

Password

Log in

Help

Register

☐ Remember Me?

What's New?

Articles

Forum

Blogs

Forum Home

New Posts

FAQ

Calendar

Community

Forum Actions

Quick Links

Advanced Search

Forum

Tool Help & Discussion

Expression 2 Discussion & Help

A Beginners 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.

If you'd like to receive bonuses in the Official Wiremod Gmod Server, Link your Steam account to you forum account [here!](#)

Results 1 to 10 of 255

Page 1 of 26

1

2

3

11

...

▶

Last ▶▶

Thread: A Beginners Guide to Expression 2

♥ 5 Likes

LinkBack

Thread Tools

Display

06-09-2009

#1

SpectreCat

Wire Sofaking

WIREMOD

MEMBER

SpectreCat's Avatar

Join Date: Mar 2008

Location: Sammamish, Washington

Posts: 505

A Beginners Guide to Expression 2

Spectre's Expression 2 Tutorial

A Beginners Guide to the Expression 2

In this tutorial I will be explaining the basics of Expression 2 for those of you confused or interested. I will attempt write this as simply as I can to avoid confusion. If you find something difficult to understand, feel free to post.

Remeber, as this is a general "All about Expression 2 tutorial," the purpose of this tutorial will not be to explain every command used by the Expression 2, but to give you enough knowledge to learn from yourself and others.

This tutorial includes:

The First 5 Lines

Basic Math

If Statements

Vectors

1 of 24

10/5/2024, 7:26 PM

Entities**Arrays****Strings****Find Commands****Applyforce****Chat Commands****Concmd (Console Commands)****Rangers****Console Screens****Holograms****Wirelink**

I will start the tutorial by explaining the first 5 lines of code that can be found in nearly every expression:

Code:

```
@name  
@inputs  
@outputs  
@persist  
@trigger all
```

"@name" is used to input what you want the expression will be called when others are viewing it.

"@inputs" this is where you write any inputs you would like to have on your expression. You may not have 2 of the same inputs, and they all must start with a capital letter. These will then appear as options to wire up later. For multiple inputs you would separate by a space.

"@outputs" is where you write any outputs that you would like to have on your expression. This is where you will wire other things to. You may not have 2 of the same outputs, and they all must start with a capital letter. For multiple outputs you would separate by a space.

"@persist" is a unique line that is only in the Expression 2. This is where you will put any thing you would like to retain between chip intervals. From personal experience, this will be used the most when you get one to more advanced wiring. You may not have 2 of the same inputs, and they all must start with a capital letter. For multiple inputs you would separate by a space.

"@trigger" Eh... Don't bother with this until you are more experienced with Expression 2. Basically for Console Screens trigger is your best friend. In a nutshell, trigger tells the expression what inputs or outputs update the chip. This is kind of hard to explain, so I may develop the explanation later.

Basic Math

A basic code would be

Code:

```
@name Add Expression
@inputs A B
@outputs Out
@persist
@trigger all

Out = A + B
```

In this simple Addition expression you are taking input "A" and adding it to input "B". Anything you would wire to the expression will receive the sum of the two.

You can do a lot of math with all most the same equation such as.

```
Out = A - B
Out = A / B
Out = A * B
Out = A ^ B
```

If statements

If statements are the most simple things to do in expression, but they will have the biggest impact on your code. The if statement could be viewed upon like a flood control valve. When the valve is open, all the data will come through, if it is closed, nothing will come through.

If statements are written as

```
if (A) {Out = B} else {Out = C}
```

this is to say, if "A" is 1, the output would be "B". However, if "A" is not 1, the output would be "C". Here is an example

Code:

```
@name Control
@inputs Button
@outputs Out
@persist
@trigger all

if(Button) {Out = 30} else {Out = 5}
```

In this example a Button would be wired to the expression. If the button was on, the expression would output 30, and if it was off the expression would output 5. This could be used for a door of some sort.

By taking out the "else" you would end up with the expression

```
if(Button) {Out = 30}
```

When you spawn this chip, out would be 0. When you press the button, it will output 30. However even if you turn the button off, the output would still be 30. This is because you are not telling it what to do if the button is off, so it retains its value. This would be used for a Toggle of some sort, and you can use another if statement to turn it back to 0.

In an If statement, you do not have to have a stable value. A good example for a statement with a changing value would be.

Code:

```
@name Speed
@inputs Speedometer Button
@outputs Out
@persist
@trigger all
```

```
if(Button) {Out = Speedometer} else {Out = 0}
```

This expression would output the Speedometer only if the button was pressed.

Now for a function called "Interval", this function will update the expression at a given number, 1000 being 1 second. Many functions require this to work, the ones that do usually involve retrieving information from the world. Intervals are also commonly used in time based expressions. While using Intervals, in some cases, you will have to use logic, but that comes later. Here is an example of a timer, when the timer hits 10 explode will equal one. We will use persists in this line, because we do not want to output the actual timer, just the explode variable.

A quick, but important, note before the tutorial:

"==" - Is Equal to

"!=" - Is not Equal to

This will be used in the following code

Code:

```
@name Timer
@inputs
@outputs Explode
@persist Time Count
@trigger all

interval(1000)
Time = Time + 1
if(Time == 10) {Explode = 1} else {Explode = 0}
```

By doing "Time = Time + 1" A loop has been created that will update with the interval. This can also be written as "Time += 1" or "Time++"

You can also reset your timer by adding the following:

Code:

```
@name Timer
@inputs
@outputs Explode
@persist Time Count
@trigger all

interval(1000)
Time = Time + 1
if(Time == 10) {Explode = 1, Time = 0} else {Explode = 0}
```

By adding a comma, I am telling the IF statement to do two things at one, first is to explode, second is to restart the timer. You can have as many as these as you want. There is also the option to use "|" this basically means "or". A good example would be

Code:

```
@name Or Test
@inputs Button Button2
@outputs Out
@persist
@trigger all

if(Button | Button2) {Out = 5} else {Out = 0}
```

This can be translated into, If Button is equal to one "or" Button2 is equal to one, Out will be 5.

One last thing to note: You can have an If statement, in an If statement. In this example I will make the timer only work if a button is on.

Code:

```
@name Timer
@inputs Button
@outputs Explode
@persist Time Count
@trigger all

interval(1000)

if(Button) {
    Time = Time + 1
    if(Time == 10) {Explode = 1} else {Explode = 0}
} else {Time = 0}
```

If statements allow the option to take up more then one line in a code, you can use as many lines as you would like. However as the code gets more complicated, you should label your code using the sign "#". What # does is tells the expression to ignore a line, so for example:

Code:

```
# out = 1
```

would do nothing at all.

Vectors

Vectors are generally used in the Coordinate situations. They are written as (X,Y,Z). Vectors are very important in expression, you will have to remember some rules though.

In the Inputs, or Outputs or Persists you would have to write the variable as V:vector. By doing this you are declaring V a vector, once you do this you can no longer use V for single numbers, angles, or entities. Another example:

Code:

```
@name GPS
@inputs GPS:vector Button
@outputs Out:vector
@persist
@trigger all

if(Button) {Out = GPS} else {Out = vec()}
```

For many of the vector's functions you will have to use a command called a function. To use a function you would typically write :(functionname) immediately after the variable.

You are not able to add or subtract a single number to or from a vector, you can achieve this however by creating a vector with a set number inside it.

An example would be

$(500,153,120) + 8$

This would not work; instead you would have to use a function to make the 8 a vector such as

$(500,153,120) + \text{vec}(8,8,8) = (508,161,128)$.

or

$(500,153,120) + \text{vec}(0,0,8) = (500,153,128)$.

The number does not have to be consistent, you can use any number you chose.

$(500,153,120) + \text{vec}(100,120,180) = (600,273,300)$

Once a vector is created, you are able to edit it with some of the following commands:

Code:

```
V:setX(#)  
V:setY(#)  
V:setZ(#)
```

if you would like to only use a single number from a vector the commands would be.

Code:

```
V:x()  
V:y()  
V:z()
```

Here is an example for using a GPS to find how high the GPS is.

Code:

```
@name Height  
@inputs GPS:vector  
@outputs Height  
@persist  
@trigger all  
  
Height = GPS:z()
```

Rotating vectors is another function in the expression2. Although it is not commonly used, it is worth knowing how to do. Basically... when you rotate an angle, you are rotating it relative to 0,0,0; for example

$\text{vec}(100,0,0):\text{rotate}(\text{ang}(180,0,0)) = \text{vec}(-100,0,0)$

$\text{vec}(100,0,0):\text{rotate}(\text{ang}(90,0,0)) = \text{vec}(0,0,0)$

This is generally used for HUD's and navigation. I do not think I explained this very well; so like always, I think the best example would be... an example.

** This code contains content that you have not learned yet, but you will learn this later in the tutorial. All you should pay attention to is Rotate. **

Do not worry about this yet but

Ang = your heads angle

Vec = your position

Code:

```
@name
@inputs
@outputs Rotate:vector
@persist Vec:vector Ang:angle
@trigger

runOnTick(1)

#Ignore ---
if(first()) {
  holoCreate(1)
}

Ang = owner():eye():toAngle()
Vec = owner():shootPos()

holoPos(1,Rotate)
#Ignore ---

#Rotate
Rotate = Vec+ (vec(100,0,0):rotate(Ang))
```

In this example vec(100,0,0) is how far away it is from you. Notice how I am only rotating vec(100,0,0). If i were to rotate Vec+vec(100,0,0) it would rotate the whole thing relative to the center of the world; obviously, this is not what you want.

Please Comment if you are confused about this section

Entities

Entities are used a lot in the expression 2 chip. They are used for players, robots, and other such things. An entity can be a person, a prop, a vehicle, an NPC or world values. There is a lot you can do with an entity, and a lot of information you can get from them. In the first 5 lines entities will be declared an entity by E:entity. You are able to find Entities using the Expression, however that section of the tutorial comes later, so use a target finder until then.

A simple command that I use a lot:

entity():isWeldedTo()

This command gives you the entity of the prop that you spawned it on.

Here is an expression using a target finder to grab the location of the player being targeted.

Code:

```
@name Vector
@inputs Player:entity
@outputs X Y Z
@persist
@trigger all

Location = Player:pos()
X = Location:x()
Y = Location:y()
Z = Location:z()
```

To wire this up, you would wire the Player input to the target finder. Usually the target finder will tell you what It is outputting. It will look something like 1 [ENTITY].

There is a HUGE amount of things you can do once you have an entity. I would strongly recommend checking the wiki for a

full list of the available commands.

[Wire Expression2 Wiki](#)

Arrays

Arrays are like tables that are filled with information, such as Entities, Strings, Numbers, and other stuff. An Array would be written on the first 5 lines as R:array. You can use it to store information for later use.

Arrays use the following commands:

Code:

```
R:setString( #, String )
R:setEntity( #, Entity )
R:setNumber( #, Number )
R:setVector( [ #, Vector )
```

The "#" stands for the index, basically what slot of the Array you want this information to be stored in.

An example of this would be

Code:

```
@name
@inputs
@outputs Out
@persist R:array
@trigger all

R:setNumber(5,500)
Out = R:number(5)
```

What I am doing in this expression is putting the information "500" in slot 5 of the array. Then I am calling the information out with R:number(5)

In the most recent update of wiremod, you have been given the option to create an array like this:

Code:

```
@persists Ary:array Blah:array Yargle:array

Ary[1,number]= 100
Blah[56,vector]= vec(100,100,100)
Yargle[1337,string]= "Roflsocks"
```

Strings

Strings are used mainly for console screens, chat bots, and find commands. Strings are written on the first 5 lines as S:string. You can write a string in the game by using quotes.

Here is an expression making me say an entities name using the target finder and S:print()

Code:

```
@name Test
@inputs Player:entity
@outputs
```



```
@persist Name:string
@trigger all

Name = Player:name()

if(Player:isAlive()) {
  print("Where is "+Name)
}
```

If you spawned this chip without the if statement, it would work, but you would not have time to wire it up. Adding `if(Player:isAlive())` means that the chip will only do this command, if the entity is alive.

When you spawn this chip it should say in blue letters on your chat box

"Where is Bob"

Assuming your target finder is targeting some guy named Bob.

You can cut parts of a string off with the commands `left()` and `right()`

Take this phrase for example

"THISISATEST"

If you did

"THISISATEST":`left(4)` = "THIS"

"THISISATEST":`right(4)` = "TEST"

You can also use a helpful function called `Explode`, where you break up a string into an array.

Using `S:explode(***)`

`***` = This is where you put the character you want to divide the string by.

For example

`Array = "This , Is , A , Test":explode(",")`

Would break this into

Code:

```
Array:string(1) = "This "
Array:string(2) = " Is "
Array:string(3) = " A "
Array:string(4) = " Test"
```

Find Commands

Find commands are used to find Players, Props, NPC's, Vehicles, and World things. They are actually very simple to use when you know what to do. For these, you must use an `interval()` otherwise you will only get the data at the time the chip is spawned.

The most basic find commands are

[findByModel\("String"\)](#) - When in Q menu right click on an Icon and click "Copy to clipboard"- paste it over "String"

[findByPlayerName\(String\)](#)

[findByClass\(String\)](#) - This includes "Player" "Npc_" "Prop" "Vehicle"

After your find command you need to get the entity using the commands `find()` or `findResult()`

`findResult(#)` - # is the number of entity you wish to get, such as players. If there were 3 players, `findResult(2)` would grab the second player

`find()` - Is exactly like `findResult(1)`

An example to get an entity would be

Code:

```
@name
@inputs
@outputs
@persist Player:entity
@trigger all

interval(10)
findByClass("Player")
Player = findResult(2)
```

This expression get the second player that joined the servers entity. You can now find out his position with `Player:pos()` or height with `Player:height()`

ApplyForce

ApplyForce is an amazing function. Its the equivalent to a vector thruster, however it uses the whole prop. You can only use this on props you own however. They are fairly simple to make, with a simple formula.

$V = (\text{Where you want it to go}) - (\text{Where it is now})$
`P:applyForce((V*25 - P:vel()) * P:mass())`

P being the prop and V being the vector.

Here is an expression that will fly over your head.

Code:

```
@name E2 Hat
@inputs
@outputs
@persist V:vector
@trigger all

interval(10)

V = owner():shootPos() + vec(0,0,50) - entity():pos()
entity():applyForce((V*25 - entity():vel()) * entity():mass())
```

`entity()` is the Expression 2 Chip and `ShootPos` is the height at which you fire your gun.

There is also `applyAngForce()` which will let you rotate your prop, however, it sucks and should not be used. Instead use `applyTorque()` which is much better, even though it is slightly more complicated.

Chat Commands

Chat commands are basically the text receiver. These commands involve the use of strings, usually not many.

Since there are not that many commands, I will list a few.

[runOnChat\(1\)](#) - This is exactly like interval, however it only updates when something is said in chat.

[lastSaid\(\)](#) - This will give you the string of what was said last in chat. Can also be written as

[Entity:lastSaid\(\)](#) - This will give you the last thing a certain player said.

[lastSpoke\(\)](#) - This will give you the entity of who said something last.

[lastTimeSaid\(\)](#) - This will give you the time the last thing was said, usually used in cases with intervals.

I will demonstrate a simple expression that will output a 1 if the word "boom" is used.

Code:

```
@name Explode
@inputs
@outputs Explode
@persist
@trigger all

runOnChat(1)

if(owner():lastSaid() == "boom") {Explode = 1} else {Explode = 0}
```

In this expression, if the owner of the chip said "Boom", Explode will equal one until something new is said in chat.

Concmd (Console Commands)

"Concmd", short for Console Commands, are very self explanatory; this function allows you to input a string into your console. This can be used for all kinds of things, such as chatbots and aimbots. These commands can only be used on the owner, however you can make someone copy your expression and it will work as well. This is client side, so it may be turned off using [wire_expression2_concmd 0](#) and many admins have this turned off.

If you have an interval, you must apply some logic to the expression. If not done correctly, the interval will constantly use the concmd and spam your console. This can be avoided by using "~" which looks for a change. Here is a simple expression that will make you say "Hi" when a button is pressed.

Code:

```
@name Concmd HI
@inputs Button
@outputs
@persist
@trigger all

if(Button & ~Button) {concmd("say Hi")}
```

I am assuming that you noticed the (Button & ~ Button), this is the logic I was speaking about earlier. Basically it says

If button is one, and there has been a change in the button, then concmd.

This is how you avoid console spam. With a regular button, the expression will constantly update as long as the button is equal to one. However the button can only change twice, and that is when you turn it on and off.

Because the concmd is using strings, the same principle applies, you can add to the string, cut the string, and what not. Here is an example

Code:

```
@name Concnd HI
@inputs Button
@outputs
@persist
@trigger all

if(Button & ~Button) {concmd("say Hi, I am "+owner():name())}
```

Remember to put a space after your last word, to avoid having yourself say
Hi, I amBob

Rangers

The internal ranger is much better then the wire ranger, only because you can do a lot more things with it. This is where the commands get a little more complicated however. First you will need to define it on the first 5 lines as R:ranger. Also, you will require an interval for rangers to function. Because it is a little more complicated, I think rangers deserve a basic command index.

ranger(range) - This will create a ranger that starts from the expression and goes directly up compared to the expression (For example, if the expression was tilted, the ranger will be tilted to).

ranger(range,x,y) - This will create a ranger that starts from the expression and goes to the local X, Y coordinate. This is to say, if you put ranger(500,50,0), the ranger will be on the right, because the X value is 50.

rangerOffset(Vector,Vector) - This is my favorite ranger function. It creates a ranger between two vectors.

Here are some options that you can add to the ranger, these should be put after the ranger.

rangerHitWater(N) - 1 / 0
rangerIgnoreWorld(N) - 1 / 0
rangerDefaultZero(N) - 1 / 0

These are the commands that you use to retrieve data from the ranger.

R:distance() - This will output the rangers distance.
R:position() - This will output where the ranger is hitting.
R:entity() - This will output the entity the ranger is hitting.
R:hit() - This outputs a 1 if the ranger is hitting something, or a 0 if it is not.

Although this may seem like a lot to remember, you will rarely be using them all at the same time. Here is a simple expression that will be like a trip mine, just spawn it on a wall.

Code:

```
@name Trip mine
@inputs
@outputs Explode
@persist R:ranger
@trigger all

interval(10)

rangerIgnoreWorld(1)
rangerDefaultZero(1)
R = ranger(500)

Explode = R:hit()
```

When adding options to an internal ranger, ex. rangerIgnoreWorld(1), always place them above the line where you are starting the ranger. So,

Code:

```
@name Trip mine
@inputs
@outputs Explode
@persist R:ranger
@trigger all

interval(10)
R = ranger(500)
rangerIgnoreWorld(1)
rangerDefaultZero(1)

Explode = R:hit()
```

would still work, but hit the world and never default to zero, because they are under the line where the ranger is defined.

Console Screens

Console screens are pretty easy to learn and a lot of fun to use from time to time. Usually once a person learns this, they go off and start making a giant expression using it. This will require you to make a new variable in the first five lines, S:wirelink. ***** Important: Please Please PLEASE, do not use an interval unless you actually NEED to when using this function. This could cause massive lag *****

First is setup.

Create a Console Screen, any size will do. Now take out a new tool called "Expression 2 - Wirelink" above your gate tool and click on your screen.

Now make an expression, on the back is always good for me. For inputs, write S:wirelink. Close out of the expression screen and wire S:wirelink from the expression, to the wirelink of the screen. Remember to wire Clk from the screen to a button or something. Also, you should wire the screens reset to another button. Now we are all set up.

There are many commands for the Console Screen, some more advanced then others, however this is a beginners tutorial so I will stick with the basics.

S:writeString("String",Xpos,Ypos,Color,BGColor,Flashing(1/0)

You do not have to fill out all this information, you can simply have S:writeString("String",Xpos,Ypos) and the text would be white with no background.

S:writeCell(2041,1) -- Resets your screen

S:writeCell(2042,Color) -- Adds a background color

Please be careful with blue commands, they could mess up your expression if you do not use them properly.

Ok before I begin to tell you about some of the commands, I would like to show you how the color system works. It's actually very easy, but you would have to figure it out first. The color system is like (255,255,255), except think that the max is 9 and you can have no commas. So for example red would be 900 (9,0,0) and green would be 90 (0,9,0) and blue would be 9 (0,0,9).

Here is a basic example of writing to the screen

Code:

```
@name
@inputs S:wirelink
@outputs
@persist
@trigger all
```

```
S:writeString("Hello World",1,1,9,90,0)
```

As basic as it can get, right? This expression will write "Hello World" in blue letters on a green background.

I will use a writeCell in this example to make the screens background white.

Code:

```
@name
@inputs S:wirelink
@outputs
@persist
@trigger all

S:writeCell(2042,999)
S:writeString("Hello World",1,1,9,90,0)
```

Now before you start playing with these commands, remember to use the screens reset after each update. Otherwise you will get copies of what you wrote. However you can also use the S:writeCell(2041,1) command to reset your screen. Here is an example of using this command.

For this expression it is not required for you to press the reset button yourself.

Code:

```
@name
@inputs S:wirelink Button
@outputs
@persist
@trigger all

S:writeCell(2042,999)
if(!Button) {
S:writeCell(2041,1)
S:writeString("Hello World",1,1,9,90,0)
} else {
S:writeCell(2041,1)
S:writeString("Goodby World",1,1,9,90,0)
}
```

Basically, when button = 0, the screen will say "Hello world", but when the button = 1, the screen will say "Goodby World." For this to work, you would have to reset the screen each time the text changes. This is because when you tell the screen to write a string, it stays there, until you reset it. Now if we didn't add the S:writeCell(2041,1) the two strings will overlap.

Obviously you can have more then one write string on the screen at one time. They have to be in different places however. Here is an example of a screen that will print your name.

Code:

```
@name
@inputs S:wirelink Button
@outputs
@persist
@trigger all

S:writeCell(2041,1)
S:writeCell(2042,999)
S:writeString("Hello,",1,1,9,90,0)
S:writeString(entity():owner():name(),1,2,9,90,0)
```

This expression will make the screen say, "Hello," and then your name under it. So for me it would say:

```
"Hello,  
<{BT}> Spectre {Cat}"
```

Now many people like to integrate keyboards with the console screen. Now this is a bit more advanced to do, but I don't think my tutorial would be complete without telling you how. For this expression, you will need an interval (Remember the lower you make it, the more laggy the screen will be).

Now the keyboard doesn't work how I wish it did, by outputting the letter you pressed. Instead it outputs a number, and each number represents a letter. I know there is a place that tells you what each number represents, and if anyone would be kind enough to post it, I will add it to tutorial.

Fortunately you do not have to worry about decoding it, because wiremod was kind enough to give us the command toChar() which decodes it for us. So to do this, add a "Wire keyboard" to the screen, I like to put it under the screen, because it looks nice. In the next expression wire the keyboard input into the memory of the keyboard.

Code:

```
@name  
@inputs S:wirelink Keyboard  
@outputs  
@persist String:string  
@trigger all  
  
if(first()) {  
S:writeCell(2041,1)  
S:writeCell(2042,999)  
}  
  
if(Keyboard & ~Keyboard) {String += toChar(Keyboard)}  
S:writeString(String,2,2,9,90,0)
```

You should notice that I have if(first()) this means that if the chip is spawned then it will run the if statement. I have this because since we have an interval, we do not want the screen resetting every 100 milliseconds. Also you might notice that we added logic again from before, this is to prevent the spamming of the letter you pressed. The reason this works is because when you press a letter, it does
String = String + Letter. {String += toChar(Keyboard)}

Most likely, you will start to make an expression, that is waaay over complicated using these commands... Like me :D

[CatOS Beta](#)

Holograms

I have come to know Holograms as one of the best parts about E2, and we have my buddy Mclovin to thank for that =) .

Holograms are basically props that are striped of all physics, so you see them, but you really can't do anything to them. They also have the unique feature to scale the holograms to anything you want (However there is a max). The commands to control there are actually pretty simple, so I've included them in this tutorial.

First off you need to know the basic functions:

Code:

```
holoCreate(Index,Vector,Scale,Angle,Color) - this is to create a hologram.  
holoCreate(Index,Vector,Scale,Angle)  
holoCreate(Index,Vector,Scale)  
holoCreate(Index,Vector)  
holoCreate(Index)  
  
holoPos(Index,Vector) - Sets the vector of a hologram  
holoColor(Index,Color,Alpha) - Sets the color of a hologram  
holoScale(Index,Vector) - Sets the scale of a hologram  
holoAngle(Index,Angle) - Sets the angle of a hologram  
holoModel(Index,String) - Sets the model of a hologram (Continue reading for help)
```

```
holoParent(Index,Index) - This is basically like weld with the holograms. Once you do this, wherever you move one holo  
holoDelete(Index)
```

These are the basic commands you will be using. Notice the Index, this is how you tell the command which hologram to control. So logically, you should set each hologram to have a different index.

HoloModel is a bit different then what you might think, there are 42 models you can currently use at revision 2506, these are

Code:

```
Low Quality  
"cone"  
"cube"  
"cylinder"  
"hexagon"  
"icosphere"  
"icosphere2"  
"icosphere3"  
"octagon"  
"plane"  
"prism"  
"right_prism"  
"sphere"  
"sphere2"  
"sphere3"  
"tetra"  
"torus"  
"torus2"  
"torus3"  
  
Hight Quality 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"
```

So say you wanted to make a holo Sphere, you would write `holoModel(1,"sphere")`. Simple as that.

How to scale it is pretty self explanatory, you put in the vector of how large you want it to be, for example:

`holoScale(1,vec(2,2,1))`

Would make a rectangle, 2 units length, 2 units width, and 1 unit high.

Now you would think to make a hologram you would just do, `holoCreate(1)`, and you would be correct, IF you did not have an interval. To do `holoCreate(1)` and have an interval would create massive lag, because you are creating a brand new hologram every interval. So we will have to approach this a different way.

Before we get too far, you will notice that I only create holos with `holoCreate(Index)` and not `holoCreate(Index,Vector,Scale,Angle,Color)`. This is because I find it easier to understand later. You do not need to do it my way however.

I hope you all remember the `if(first())` thing I mentioned earlier, this is what we use. Here is a quick example of how to make a hologram circle hover above your expression.

Code:

```
@name Holo Example  
@inputs  
@outputs  
@persist  
@trigger
```



```
interval(10)
if(first()) {
  holoCreate(1)
  holoModel(1,"sphere")
}

holoPos(1,entity():pos()+vec(0,0,50))
```

Now the reason I put holoModel() in the first() statement is because we really don't need to update the model every 10 milliseconds

You can create more then one holo at a time too. Like:

Code:

```
@name Holo Example
@inputs
@outputs
@persist
@trigger

if(first()) {
  holoCreate(1)
  holoCreate(2)
}

holoPos(1,entity():pos()+vec(0,0,10))
holoPos(2,entity():pos()+vec(0,0,20))
```

I said I would not talk about while() loops but I might just mention that it's out there. While I highly recommend that you do not use these loops until you are a bit more experienced with wiremod. Ill show you how to use one here.

Code:

```
@name Holo Example
@inputs
@outputs
@persist I
@trigger

if(first()) {
  while(I < 20) {I++
  holoCreate(I)
  }
}
```

To me, the best way to explain while() loops are that they are if statements, that have intervals built in.

In this statement I'm saying:

While I is less the 20, I++, holoCreate(I). This will create 20 holograms. If you don't understand this, don't worry, just stick with the basics.

Basically, anything you can do with applyForce or applyAngForce you can do with Holograms, without the formulas. For example, making a Triangle rotate over your head.

Code:

```
@name Holo Example
@inputs
@outputs
@persist
@trigger
```

```
interval(10)
if(first()) {
  holoCreate(1)
  holoModel(1,"pyramid")
}

holoPos(1,owner():shootPos()+vec(0,0,20))
if(I < 180) {I++}
if(I == 180) {I = -180}
holoAng(1,ang(0,I,0))
```

Here is something I made with holograms :D

[Cat's Hologram Chess](#)

Wirelink

Wirelink is a very useful tool if you know how to use it. Previously in the Console Screen part of this tutorial, you used wirelink to get text to your screen. This is not all the wire link does however. The best way to explain this, is that you can wire up more then 30 different things with one wirelink. This is commonly used in Advanced Pod controllers, but it can also be used in almost anything that has a wiremod output (Except other Expression).

Here is a short list of the basic functions that you can use.

[XWL:number\(S\)](#) - You can link this to anything that outputs a number, ex: 1 , 0

[XWL:vector\(S\)](#) - You can link this to anything that outputs a vector

[XWL:string\(S\)](#) - You can link this to anything that outputs a string

[XWL:entity\(S\)](#) - Same with entity...

The pod controller is really the best way to explain how to use this tool, basically because of its massive amount of outputs. First comes set up, spawn an Advanced Pod Controller and hook it up to a chair (If you do not know how to use an Advanced Pod Controller, then this is not the right tutorial for you). Now for the inputs on your expression, put P:wirelink. Take out the wirelink tool (Located in the tools section of your wiremod menu) and simply click on the Advanced Pod Controller. Now wire the input from the expression, to the newly created wirelink output on your advanced pod controller (Go slow when looking for the wirelink, or you will miss it and get angry).

In this code you will make the game display if you are hitting W, S, A, or D. Using the function [hint\("String",How long it will appear\)](#)

When setting up the function P:number(S) please remember that it is **case sensitive**.

Code:

```
@name Wirelink Example
@inputs P:wirelink
@outputs
@persist W A S D
@trigger all

runOnTick(1)
W = P:number("W")
A = P:number("A")
S = P:number("S")
D = P:number("D")

if(W & $W) {hint("W",2)}
if(A & $A) {hint("A",2)}
if(S & $S) {hint("S",2)}
if(D & $D) {hint("D",2)}
```

The reason we are using P:number is because W A S D on the advanced pod controllers output numbers (1/0). This is why

we would use P:number for buttons as well.
Pretty simple right =D

Now to make this a little more complicated I'm going to make the expression tell you WHO is pressing the button.

Code:

```
@name Wirelink Example
@inputs P:wirelink
@outputs
@persist W A S D
@trigger all

runOnTick(1)
W = P:number("W")
A = P:number("A")
S = P:number("S")
D = P:number("D")
Name = P:entity("Entity"):driver():name()

if(W & $W) {hint(Name+" Has Pressed W",2)}
if(A & $A) {hint(Name+" Has Pressed A",2)}
if(S & $S) {hint(Name+" Has Presses S",2)}
if(D & $D) {hint(Name+" Has Pressed D",2)}
```

For this I'm using P:entity() and getting the Entity that the Advanced pod controller is outputting (Which is outputting the vehicle it is linked to). Then I am using Driver, to get who is driving the vehicle, and then using Name to get the name of the driver. So now it will say

"SpectreCat Has Pressed A"

+Rep is appreciated :D

Please post for any suggestions or errors
[A Beginners Guide To Expression 2 Thread](#)

Last edited by SpectreCat; 4 Weeks Ago at 02:53 PM. Reason: fixed some inaccuracies and formatted example code a bit

Share

Reply With Quote

06-09-2009

#2

SpectreCat

Wire Sofaking

MEMBER

SpectreCat's Avatar

Join Date: Mar 2008

Location: Sammamish, Washington

Posts: 505

re: A Beginners Guide to Expression 2

First Tutorial Ever, so... be nice 😊

Share

New to the E2? Try my Tutorial:
[A Beginners Guide to Expression 2](#)



Please send a PM before you add me to friends

Originally Posted by **chinoto**

E2 is not complicated, but many of the people who use it do complicated things with it.

Reply With Quote

06-09-2009

#3

Mr. Slushy

Wire Noob

Mr. Slushy's Avatar

Join Date: Jun 2009

Posts: 2

re: A Beginners Guide to Expression 2

HAHA great first tutorial! I even picked up some stuff from that :P

Its sure to learn some of those beginners a bit of E2.

... just watch the smilies for functions.... i don't think expresson takes those XD

Share

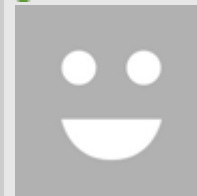
Reply With Quote

06-09-2009

#4

sambo

Wire Noob



Join Date: May 2009

Posts: 6

re: A Beginners Guide to Expression 2

I just thought I would point out, you put "trigget" instead of "trigger" in some of your examples. Good tutorial though

Share

Reply With Quote

06-09-2009

#5

re: A Beginners Guide to Expression 2

I see quite some flaws here and there but it's ok

Share

Nicolai1

Banned

WIREMOD BANNED



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

Reply With Quote

06-09-2009

#6

SpectreCat

Wire Sofaking

WIREMOD MEMBER



SpectreCat's Avatar
Join Date: Mar 2008
Location: Sammamish, Washington
Posts: 505

re: A Beginners Guide to Expression 2

Sorry about that, I made this at around 1am so I was not concentrating as best as I could. Now I am fixing some errors.

Share

New to the E2? Try my Tutorial:
A Beginners Guide to Expression 2



Please send a PM before you add me to friends

Originally Posted by chinoto

E2 is not complicated, but many of the people who use it do complicated things with it.

Reply With Quote

06-09-2009

#7

Drunkie

Ursus maritimus

WIREMOD SUPER MOD



re: A Beginners Guide to Expression 2

what is expression 2, did you need to learn me to so i can make cool stuffs like i see on garreysmood



Join Date: Feb 2009
Location: Canada
Posts: 6,255
Blog Entries: 1

Share

Reply With Quote

06-09-2009

#8

Masogir

Wirererer



Join Date: Mar 2008
Posts: 292

re: A Beginners Guide to Expression 2

Originally Posted by **Drunkie**

what is expression 2, did you need to learn me to so i can make cool stuffs like i see on garreysmood

i has already done learneded you boths.

Share

Reply With Quote

06-09-2009

#9

jmazouri

Wire Noob



Join Date: Apr 2009
Posts: 3

re: A Beginners Guide to Expression 2

hay dis tut showd me sum gud stuff i mad a bot tht goe arond and shot people lol thx spectr

Share

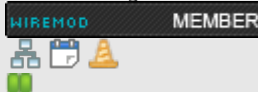
Reply With Quote

06-09-2009

#10

SpectreCat

Wire Sofaking



re: A Beginners Guide to Expression 2

Maaasooo 🤖

SpectreCat's Avatar
Join Date: Mar 2008
Location: Sammamish,
Washington
Posts: 505

I don't think you would be able to read the tutorial with that spelling Jman

Share

New to the E2? Try my Tutorial:
A Beginners Guide to Expression 2



Please send a PM before you add me to friends

Originally Posted by chinoto

E2 is not complicated, but many of the people who use it do complicated things with it.

Reply With Quote





Page 1 of 26 1 2 3 11 ... Last

« Previous Thread | Next Thread »

LinkBacks (?)	
Where to start with E2 Refbck This thread	11-01-2010, 01:35 PM
Where to start with E2 - Facepunch Refbck This thread	06-12-2010, 06:39 AM
Expression 2 chip that kills anyone that owner wants - Facepunch Refbck This thread	03-14-2010, 09:24 AM
???????? ?? Expression 2 - ?????? - Garry's Mod - ????? Garry's Mod Alliance Refbck This thread	03-09-2010, 10:12 AM
www.g-mod.ru Refbck This thread	03-06-2010, 06:17 AM
NIGMA : expression 2 setX Refbck This thread	03-06-2010, 12:42 AM
want wire applyforce/variation help? - Facepunch Refbck This thread	03-02-2010, 08:54 AM
G3 Expression 2 Tutorial Refbck This thread	02-23-2010, 10:52 AM
???????? ?? ?2 - ?????? ?? Garry's Mod - ?????? ????? - ??? ???? Refbck This thread	02-20-2010, 10:31 PM
Expression 2 ruined Gmod. - Page 2 - Facepunch Refbck This thread	02-10-2010, 02:34 PM
User talk:Cake4672 - GMod Wiki Refbck This thread	02-03-2010, 07:27 AM

New to EZ !/ - Facepunch Refbck This thread	01-29-2010, 11:17 AM
Gmod Wire Keypad (Addon) Help! - Facepunch Refbck This thread	01-29-2010, 08:23 AM
My First Expression 2 help - Facepunch Refbck This thread	01-28-2010, 09:06 PM
??????? ?????????? ?? Expression Chip 2 - garrysmod.ru Refbck This thread	01-27-2010, 02:03 PM
G3 Expression 2 Tutorial Refbck This thread	01-27-2010, 12:40 PM
Making a microwave using wiremod - GMod Wiki Refbck This thread	01-27-2010, 08:52 AM

Similar Threads	
An Intermediate Guide to Expression 2 By Matte in forum Expression 2 Discussion & Help	Replies: 106 Last Post: 07-20-2012, 09:21 AM
Key-Points For Beginners By Hitman271 in forum CPU, GPU, and Hi-speed Discussion & Help	Replies: 7 Last Post: 01-09-2010, 10:28 AM
Guide By Conmanx360 in forum Installation and Malfunctions Support	Replies: 4 Last Post: 05-05-2008, 09:12 PM
Thrusters Only Guide By Bernie in forum Gate Nostalgia (Old School Wiring) Discussion & Help	Replies: 13 Last Post: 04-08-2008, 09:03 AM
Wiremod Guide? By Tjuven in forum Installation and Malfunctions Support	Replies: 14 Last Post: 03-28-2008, 06:18 PM

Bookmarks
 Digg
 del.icio.us
 StumbleUpon
 Google
Facebook

Posting Permissions	
You may not post new threads	BB code is On
You may not post replies	Smilies are On
You may not post attachments	[IMG] code is On
You may not edit your posts	[VIDEO] code is On
	HTML code is Off
	Trackbacks are On
Pingbacks are On	
Refbacks are On	
	Forum Rules