The VCE manual because the in game one is broken. Credit to Lugnut for the raw VCE manual.

Table Of Contents

Introduction

Welcome to the VCE Handbook! My name is Duckster, and I'll be your guide.

First off, if you're new to the VCE, I suggest you start at the first beginner tab, and continue step by step until you reach the end. If you have prior experience, feel free to jump around in the handbook. If you reach a section where you don't know how to do something, you might be too far in.

For those who actually are about the time and effort that went into this mod, feel free to read the History and Credits sections.

History
Since the beginning of Blockland, players have always wanted to design their own role-playing game, design advanced event contraptions, or even simple contraptions that would be impossible with the default selection of events. These desires is what fueled this mod into creation. Zack0Wack0 is the man for the job. He wished to create a mod the revolutionized the eventing world, and so he did. When events were first developed in Blockland, Zack0Wack0 teamed up with one of his scripting idols, Chrono. Chrono created the first version of the mod and sent it to Zack0Wack0. Zack0Wack0 then hosted the mod, and the crowd went wild.

Eventually, Chrono grew tired of working on the mod. Zack0Wack0 then carried the project on through the fourth version. Zack0Wack0 has tagged Truce along as a sidekick, and he created some extensions to the mod. After that, the mod remained unchanged, as the people of Blockland loved it as it was.

A while after Zack0Wack0's downtime, small talk had begun about a "Variable Replacers" system. Clockturn had messaged Zack0Wack0, and told him that he had a variable replacer system. This evolved into the fifth installment of the mod. The entire core of the mod was complete rewritten, optimized, and soon released among a few beta testers. Many people loved it, but some could not understand it. That is when this manual was born. The event mod continued to become more advanced with nifty features as new ideas tumbled into Zack0Wack0's hands.

I long time later, in the eleventh version of the mod, Zack0Wack0 summoned Boom to help out with the manual. Since the fifth version, he understood the mod very well, and has mastered every aspect of the mod. Because Boom can speak to the common people in lamens terms, he was put up to the task of rewriting the manual for better

understanding. A lot of time has been put into this mod, and it has evolved greatly since its first version. Hopefully everyone will continue to enjoy it. Have fun!

Credits

- Zack0Wack0: Creator; Documentation; Scripter; GUIs
- Clockturn: Scripter
- Chrono: Scripter
- Truce: Scripter; GUIs
- Lilboarder: Testing; Documentation
- Jaydee: Testing
- Boom: Manual Writer
- All of the awesome people who use this mod :3

Beginner

Welcome to the first part of the VCE Handbook. Before you go in depth in the VCE and play around with variables, it would be a good idea to know exactly what a variable is.

A variable is simply something that changes. Its opposite, a constant, does not change. An example of a constant would be the symbol for Pi (3.141...). Pi never changes. A variable on the other hand, does. A good example of a variable is money. Let's say that you have $40. Therefore, we could have a variable called "Cash" and set it to 40. Now, let's say you buy an item for $10. You now have $30. How did you know that? You simply subtracted $10 from the original $40 to get $30. In the VCE, you would modified the "Cash" variable by subtracting 10 from it. Pretty simple, huh?

Now that you know the basic concept of variables, it's time to break down the formatting of the VCE. The biggest part of the VCE is the ability to use IF statements. An IF statement in the VCE looks like this: "onActivate -> Self -> VCE_ifVariable -> [Cash]>= [40] [ ]". In the VCE, certain things go in certain places. In the first box, you have the variable being compared. In the second box, you have what it is being compared to, and inbetween the two, you have two comparison symbols telling the VCE how it's being compared. You will learn about the last box in the

section called "Multiple IF Statements". Until then, only use one IF statement per brick. For those who can't tell, ">=" means greater than or equal to. Here's what this IF statement is saying in english, "if "cash" is greater than or equal to 40, ...". This statement will either come back true or false.

By using the inputs that come with the VCE, you can make things happen based on whether or not the IF statement was true or false. For the IF statements, you will use onVariableTrue, for when the statement is true, and onVariableFalse, for when the statement is false. Before we continue, allow me to show you all of the comparison symbols and their meaning:

== : Equal to
!= : Not equal to
>= : Greater than or equal to
<= : Less than or equal to
> : Greater than
<: Less than
~= : Similar or equal to

Now you know the basic format behind the VCE. I think you are ready to start making your own variables!

Variables
Welcome to the Variable section in the VCE Handbook. Before you continue reading this section, be sure that you understand the concept of a variable. In the VCE, you can make your own variables. These variables can be used for keep track of money, stats, scores, you name it. To make a variable, you can simply use the VCE_modVariable output event. Here's what that will look like: "onActivate -> Self -> VCE_modVariable {Brick} -> [Cash] add [1]". As you can see, there are four parameters in this event. There's the target of where the variable is located, the variable you are modifying, how you are modifying it, and by how much you are modifying it. In the example, I am adding "1" to the variable "Cash". The "Cash" variable is on the "Brick" target. There are many ways to modify a variable. Here's a list of them, and what they do:

Set : Sets the variable to the specified number

Add : Adds the specified number to the variable
Subtract : Subtracts the specified number from the variable
Multiply : Multiplies the variable by the specified number
Divide : Divides the variable by the specified number
Floor : Rounds the variable down. Specified number unused.
Ceil : Rounds the variable up. Specified number unused.
Power : Sets the variable as the base, and the specified number as the exponent
squareRoot : Finds the square root of the variable. Specified number unused.
Percent : Takes the specified number, divides it by the variable, and converts to percent.
randomNum : Sets the variable as the minimum, the specified number as the maximum, and randomly picks an integer between.
getWord : Finds the Nth word in the variable. N is set to the specified number.
strLow : Makes all characters in the variable lowercase. Specified number unused.
strUp : Makes all characters in the variable uppercase. Specified number unused.
strChar : Finds all characters and symbols that proceed the specified number/character.
strLen : Counts up the number of characters in the variable. Specified number unused.

As you can see, there are many ways to modify a variable. Now that you know how to modify a variable, it's time for you to learn about variable replacers. The fancy term "Variable Replacer" simply means a word that replaces the variable in such a way that the VCE can interpret it. Variable replacers can be used in evented prints, specified numbers in modification, and IF statements. If you have not learned IF statements, please go back and read the previous section.

Now, a variable replacer works in three parts. You have the term "var", which simply specifies that the string is a variable. Next, you have the target. The target can range anywhere from the brick, player, client, vehicle, or minigame. There is also global variables which can only be used in variable replacers, and also a Named Brick target feature. You simply use "nb_(brick name)" as the target.

Lastly, you have the name of the variable. Each section is seperated by a colon. The string starts with a "<", and

ends with a ">". Let's use the example, "Cash". I know that "Cash" was set to the brick target. Therefore, my variable replacer will look like this: <var:brick:cash>. Variable replacers are not case-sensitive. Also, you can substitute the target of the variable replacer like so: "br" for brick, "pl" for player, "cl" for client, "ve" for vehicle, "mg" for minigame, and "gl" for global. It's very simple.

Values
Welcome to the Value section in the VCE Handbook. Values are very similiar to variables. In fact, they are variables, but values are special variables. When using the output VCE_ifValue, you will see a setup very similar to VCE_ifVariable. Instead of using the name of the variable in the first box, you can only use variable replacers. If you do not know what variable replacers are, please read the previous section.

So, what makes a variable special? Simple. A special variable, or value, is simply a variable that you cannot modify. For example, there is a value that checks if the player is an admin. It's a good variable to have, but you don't want people modifying it to become an admin. Therefore, it is a value. Here is a full list of values' variable replacers and what they correspond to:

<var:brick:datablock> : The name of the brick's datablock
<var:brick:colorID> : The color ID of the brick
<var:brick:printcount> : The brick's print count
<var:brick:printname> : The name of the brick's print
<var:brick:name> : The name of the brick
<var:brick:colorFxID> : The color effect ID of the brick
<var:brick:printID> : The print ID of the brick
<var:brick:shapeFxID> : The brick's shape effect ID
<var:brick:ownerName> : The owner of the brick's name
<var:brick:ownerBL_ID> : The owner of the brick's Blockland ID

<var:client:bl_id> The client's Blockland ID
<var:client:name> : The client's name
<var:client:clanPrefix> : The client's clan tag shown before their name

<var:client:clanSuffix> : The client's clan tag shown after their name
<var:client:score> : The client's score
<var:client:lastmsg> : The client's last message
<var:client:lastteammsg> : The client's last team message
<var:client:isAdmin> : Checks if the client is any kind of admin (1 = True, 0 = False [Boolean])
<var:client:isSuperAdmin> : Checks if the client is a super admin (Boolean)
<var:client:isHost> : Checks if the client is the host (Boolean)
<var:client:brickcount> : The number of bricks planted by the client

<var:player:damage> : The amount of damage that has been done to the player
<var:player:health> : The amount of health the player has left
<var:player:maxHealth> : The most health the player can have
<var:player:velx> : The player's velocity on the global X axis
<var:player:velY> : The player's velocity on the global Y axis
<var:player:velZ> The player's velocity on the global Z axis
<var:player:crouching> : Checks if the player is crouching (Boolean)
<var:player:jumping> : Checks if the player is jumping (Boolean)
<var:player:jetting> : Checks if the player is jetting (Boolean)
<var:player:firing> : Checks if the player is firing/clicking (Boolean)
<var:player:sitting> : Checks if the player is sitting (Boolean)
<var:player:datablock> : The player's datablock name
<var:player:currentItem> : The UI name of the player's current item
<var:player:item*> : The UI name of the *th item in the player's inventory (1-5)
<var:player:energy> : The amount of energy the player has

<var:vehicle:datablock> : The name of the vehicle's datablock
<var:vehicle:health> : The amount of health the vehicle has
<var:vehicle:maxhealth> : The maximum amount of health the vehicle can have

<var:global:date> : The current date
<var:global:time> : The current time
<var:global:simtime> The current simulated time
<var:global:macintosh> : Checks if the user is on a Macintosh Computer (Boolean)
<var:global:windows> : Checks if the user is on a Windows Computer
<var:global:serverName> : The name of the server
<var:global:port> : The number of the port the user is accessing the server on
<var:global:maxPlayerCount> : The maximum number of players the server can have
<var:global:playerCount> : The number of players on the server
<var:global:brickcount> : The total number of bricks on the server

As you can see, there is a very long list of values that can be used. The targets of values may be shortened too. You are not limited to just using values in VCE_ifValue. Any variable replacer will work. Now that you know all of the values, why not go try some out?

Saving/Loading
Welcome to the Saving and Loading Variables section in the VCE Handbook. Saving and Loading variables is probably the easiest part of the VCE. Since you should understand variables very well by now, this section will go quickly for you.

First off, you can only save player and client variables. Good news is, you can reload them later. This means that if a user leaves the servers and looses his player and client tied variables, he or she can reload them through the VCE as long as the server has been up while they were gone.

To save variables, you use the VCE_saveVariable command on the Self target. To load variables, you use the VCE_loadVariable command that is also on the self target. The layout is pretty straight forward, so you should not get lost. Just simply type in the name of the variable you wish to save or load. To load more than one, you can do this: "Var1, Var2, Var3, Var4". See how each variable is followed by a comma and a space? If you're trying to load a lot of variables, spaces can add to the character count, disallowing you to keep adding variables that you want to save or load. You can actually just use commas to seperate the variables like so: "Var1,Var2,Var3,Var4". Be

warned, if you start using one format of seperation, do not switch. This would be incorrect: "Var1, Var2, Var3,Var4" See how there is no space between Var3 and Var4? That needs to be changed or the variables will not load correctly. Now that you can save and load your variables, the limits of the VCE just improved.

Retro Check
Welcome to the Retro Check section in the VCE Handbook. As advanced as Retro Check may seem, I can assure you, it is not. It is placed after the Values section because I wanted you to learn the new way of using values rather than this. Retro Check is simply for the users of the VCE who liked the old way of checking variables better. Here's a list of Retro Check options, and what they relate to:

ifPlayerName : The player's name (<var:cl:name>)
ifPlayerID : The player's ID (<var:cl:BL_ID>)
ifAdmin : If the player is any kind of admin (Boolean) (<var:cl:isAdmin>)
ifPlayerEnergy : The amount of energy the player has (<var:pl:energy>)
ifPlayerDamage : The player's damage (<var:pl:damage>)
ifPlayerScore : The player's score
ifLastPlayerMsg : The last message from the player (<var:cl:lastmsg>)
ifBrickName : The name of the brick (<var:br:name>)
ifRandomDice : A random number between 1 and 6

VCE_retroCheck works just like any other value, it can't be changed through the VCE, however, it can't be displayed in an evented print unless you use its variable replacer correspondant. This concludes the Retro Check section of the VCE handbook.

Advanced
Welcome to the Advanced section in the VCE Handbook. The following chapters discuss slightly more advanced features of the VCE. I recommend getting used to the basics beforehand. If you feel lost, that's okay, it just means that you are not yet ready for the advanced portion of the VCE.

Multiple IF Statements

Welcome to the Multiple IF Statements section in the VCE Handbook. In the VCE, having multiple IF statements is really easy. Remember the last box on the VCE_ifVariable output? That can be used for multiple IF statements. Before we dive into using more than one IF statement, you need to know what that box does. Let's say you want two IF statements on one brick. Well, you can't go about doing this, because one IF statement executes all of the onVariableTrue/False events. With the last box on the right, you can control which events are executed. Let's say your first IF statement corresponds to the three events following it. The number of those events (which can be found on the far left) just happen to be events 3, 4, and 5. If you also had onVariableTrue/False events, they would be triggered too. But, if you type in "3 5" in the rightmost box, that IF statement will only check for onVariableTrue/False events with numbers 3 through 5. Simple, right? Using the last textbox will allow you to have multiple IF statements. As you can see, using multiple IF statements on one brick is really easy. Just for your information, events without an onVariableTrue/False input will not be run through an IF statement. They will simply run through their normal inputs. Now that you know how to use multiple IF statements, go try it out!

Functions

Welcome to the Functions section in the VCE Handbook. Functions are probably the most complex feature in the VCE. Functions come in three parts: calling functions within a brick, calling a function from another brick, and stating arguments while calling a function from another brick. Obviously, we'll start off at square one. The only required knowledge you need to know outside of the VCE for this is Relays. Relays are basically a way of controlling inputs and outputs. If you do not understand the concept of relays, I suggest you learn it.

Now, onto part one: calling functions within a brick. So, what's a function? It is quite similar to a relay. Functions, however, have names, meaning more than one per brick can be used. Before you can call a function, you must state that function. You do so with the VCE_stateFunction output on the Self target. VCE_stateFunction has two boxes. A large one, and a small one. The first box is where you put the name of the function. The second box is where you specify which onVariableFunction input events belong to that function. Let's say you've stated the function "Check". You can now call that function with the VCE_callFunction output which is located under the Self target. VCE_callFunction comes with two boxes, this time both of them are large and of equal size. The first box is where you put the name of the function you wish to call. If you put "Check" in the first box, you'll call the function "Check" on the desired input. The second box brings us to part two of functions: calling functions from other bricks. Let's say that

the function "Check" is on a brick named "Test". You can call the function "Check" on the named brick "Test" by putting "Check" in the first box, as it is the name of the function, and "Test" in the second box. This will call the function "Check" on all bricks named "Test". It's pretty simple.

Now to expose the second box's double feature in the VCE_callFunction command: creating arguments. So, let's use the function "Check" and the brick named "Test" again. Except this time, you will use the Named Brick target instead of the Self target. You will "on(Input) -> {Test} -> VCE_callFunction -> [Check] [ ]. This alone will call the function "Check" on all bricks named "Test". Doing this no longer ties up the second box, so it is used for something else. Let's use a basic variable replacer, <var:client:name>. If you put "<var:client:name>" into the second box, an argument will be created. The number of arguments starts at 0, and increments by 1. Since "<var:client:name>" is the only thing in the box, it becomes Argument0, or "arg0". Arguments can be displayed with a variable replacer like so: <var:brick:arg0>. Now, "<var:client:name>" is in that second text box which is a VCE_callFunction event with the named brick "Test" as the target. Because "Test" is the target, <var:brick:arg0> applies to all bricks with the name of "Test". Therefore, because "<var:client:name>" is the first argument (arg0), <var:brick:arg0> is the same as <var:client:name>. More than just values may be used as arguments. Custom variables, constants, and characters all work too. This can be handy if you need to transfer the variables from one brick to another. You can have more than one argument in a VCE_callFunction event. Each argument can be seperated by an absolute value sign " | ". So, let's say we have the variables "Gold", "Silver" and "Bronze" on an unnamed brick. You need those variables to be transfered to a brick named "Elements". All you need is to create and function, and the call that function through the Named Brick target. The second box will contain the three variables that we are turning into arguments. They are as followed: "<var:brick:gold> | <var:brick:silver> | <var:brick:bronze>". The variable "Gold" from the unnamed brick will become "arg0" on the brick named Elements. "Silver" will be "arg1", and "Bronze" will be "arg2". You may have as many arguments as you can fit into the text box. Now that you've learned how to use functions, it's time to move on to last section of the VCE Handbook.

VarLinks

Welcome to the VarLinks section in the VCE Handbook. Variable Links come in two parts. The actual link, and what it does, along with the input of onVariableUpdate. First off, variable links are only to be used the chat message events. When used, they will simply create a link, and when you click that link, a defined client variable will be set to a

defined number.

Allow me to break it down. Like variable replacers, a variable link starts with a "<" sign, and ends with a ">" sign. With variable links, there are four parts. The first part is the phrase "varlink", which classifies this as a variable link. Seperated by a colon, the second part is the link name. For example, if I put "Click Here", the link would show as "Click Here". Simple concept. Now, the third section is seperated by an underscore, rather than a colon. After the underscore, you classify the name of the client variable you wish to use. Finally, the fourth section is seperated by another colon, and you type the value that you wish the defined client variable to be.

The big picture looks like this ". So, in a chat message event would show up as "Click Here", and when you clicked that link, the client variable "clicked" (<var:client:clicked>) would be set to the value of 1. So, how would you make something happen at the same time as someone clicking a link? Simple. Use onVariableUpdate. The input of onVariableUpdate is triggered when a variable is changed, whether it is by a variable link, or by using VCE_modVariable. In the example variable link, I could use the event "onVariableUpdate -> Client -> VCE_ifVariable -> [Clicked] == [1] [ ]" to see if someone clicked the link. Variable links have multiple uses. Now that you know how to use variable links, go put it to good use!