Voltseon’s Multiplayer Solution

A plugin for Pokémon Essentials by Voltseon

This document serves as the documentation for the Pokémon Essentials plugin: “Voltseon’s Multiplayer Solution” (VMS).

Document version: 1.0

For VMS versions 1.0 and below.

January 3, 2024

Preface

Welcome to this document written and maintained by Voltseon. If you are reading this, it means that you are interested in learning how to use and implement Voltseon's Multiplayer Solution (VMS), which is great.

In this document, I will cover anything that is available within the plugin itself and help you set it up. Do note that this document will not help with tasks this plugin is not meant to do. There will not be anything related to down/upgrading the plugin to a different version of *Pokemon Essentials* or even porting it to other toolkits such as *Pokémon Studio / Pokémon SDK*.

I did not make *Pokemon Essentials*, and the creators and developers should be credited properly. I do not profit from the distribution or use of VMS, and it should therefore also not be redistributed. If you have paid for VMS, please take immediate action, and contact the appropriate parties for a refund. This document and related plugin are provided free of charge and should not be sold under any circumstances.

With that all being said, please make sure to read this document carefully and if there are any questions you may have feel free to contact me from the support section.

Table of Contents

1. Getting Started

1.1. Prerequisites

To get started you will need to download and install *Pokemon Essentials* v21.1 which can be downloaded from *Relic Castle*: https://reliccastle.com/essentials/news/. If you are unsure how to get started with *Pokemon Essentials* please refer to the guides: https://essentialsengine.miraheze.org/wiki/Guides.

Make sure you own a copy of *RPG Maker XP* as it is pretty much required to start making games for *Pokemon Essentials*. You *could* make fangames without it, but you'll start hitting a bunch of roadblocks when trying to change maps and events.

To start using the multiplayer functionality you will need some kind of server or network relay so players can interact through the internet. Alternatively, you could ship the server to your players and allow them to run LAN servers (not recommended as you're giving players access to any server data). This option is further explained in the following chapters.

1.2. Setting up

Before setting up anything, make sure you have the latest version of VMS downloaded and extracted from the thread page: https://reliccastle.com/resources/1453/. Since the plugin has only been tested for *Pokemon Essentials* v21.1, it is recommended you have this installed and confirm that it works. Additionally, install any *Pokemon Essentials* hotfixes that are released and available, as they are made to fix any found bugs after the version release.

Finally, this document assumes you are using Windows 10 or above. This document does not provide any support for other operating systems such as MacOS or Linux.

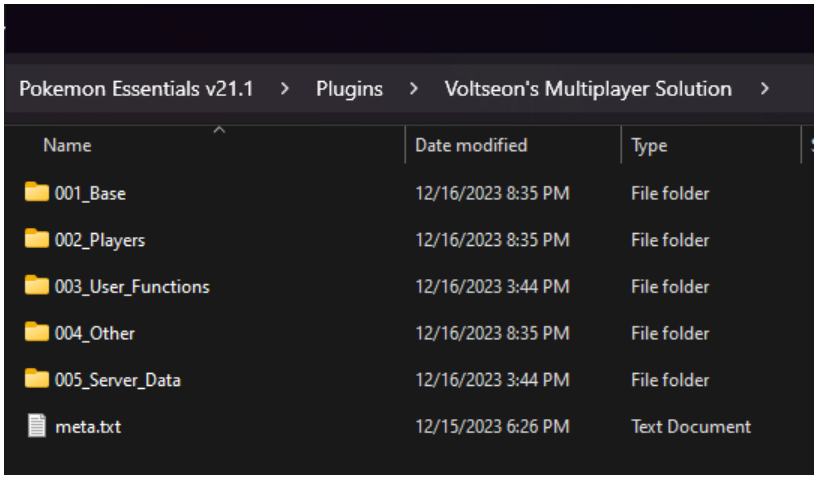
2. Installation and Setup

2.1. Downloading the plugin

To download VMS, go to the plugin thread found at https://reliccastle.com/resources/1453/. Once you have downloaded the plugin, extract the *VMS_09.zip* file somewhere you can easily access the contents. If you are having trouble extracting a zip file, follow the steps listed on this webpage from Microsoft: https://support.microsoft.com/en-us/windows/zip-and-unzip-files-8d28fa72-f2f9-712f-67df-f80cf89fd4e5. The plugin also requires the use of *rainefallUtils*, a utility script created by rainefall. It is used to create and remove events/ other players onto your game. It can be found at https://reliccastle.com/resources/1079/. Make sure to download and install the latest version that is compatible with VMS (that being the same *Pokemon Essentials* version).

2.2. Adding it to your project (client)

After extracting the zip file, you should see two folders: "*Voltseon's Multiplayer Solution*" and "*Server (DO NOT SHIP)"*. Put the former in your game's Plugins folder (create one if there is none). So, your files should now look like in the screenshot.



The latter folder contains your server files, put these somewhere safe as they will be required in the next step, which is setting up your server. If you aren't planning on giving your players access to your server files, make sure to not put this folder inside of your game files.

2.3. Setting up your server (server)

2.3.1. Local Server (LAN)

This is the most useful option for testing purposes as you will be able to make changes to your server quickly. It is recommended you test your server with this option as it will make testing easier.

Before we begin, it is recommended you have a secondary machine ready that also has your game installed. That way you can ensure that the server is running correctly.

To start you will need to figure out your local IP address, this is different from your public IP address as it is used to communicate within your own network. On Windows you can easily find your IP by opening a command prompt (search '*cmd*' in Windows Search and it should pop up). Type '*ipconfig*' and hit enter, you should see a bunch of data related to your IP. What you are interested in is an 'IPv4 address'. It usually looks something like '192.168.172.10' though it can change depending on your network setup. If you don't see an IPv4 address type '*ipconfig /all*' instead. If you are still running into issues, try looking at this article by Microsoft: https://support.microsoft.com/en-us/windows/find-your-ip-address-in-windows-f21a9bbc-c582-55cd-35e0-73431160a1b9. Next, you will need to open up a port for public access to use. Usually 25565 works (which is the default) as that one is usually already open and available, however if you are running into connection issues follow this article by Microsoft on how to open up a port on Windows: https://learn.microsoft.com/en-us/sql/reporting-services/report-server/configure-a-firewall-for-report-server-access?view=sql-server-ver16. If you're still encountering problems, make sure your port is open for port forwarding in your router settings.

After finding your IP address and an open port, put them into your '*config.ini*' file of your server as well as in the '001_VMS_Config.rb' (doing so is described in chapter 2.5). Once you have set these up, you will be required to install Ruby if you haven't already. Installing Ruby is fairly simple, the required steps can be found on the Ruby docs (https://www.ruby-lang.org/en/documentation/installation/).

Now you are ready to do your first test run, amazing! Navigate to your server files and open a terminal window (right click inside of your folder and select '*Open in Terminal'*, if you don't see this option simply type '*cmd*' in the top bar and hit enter) you should see a PowerShell or Command Prompt window pop up. Type '*ruby Server.rb*' and hit enter. If you have done all the steps correctly you should see a message pop up with '*[HH:MM:SS] Server started on HOSTNAME:PORT.'*. If so, you have set up your server correctly. As long as this window is open your server is running on your machine. If you want to stop the server or restart it use the shortcut '*Ctrl+C*' and you can use the up arrow to get your previous command back

and hit enter to start the server again. If you are making changes to your server files, make sure to restart your server as it could crash otherwise.

2.3.2. Dedicated Server (Google Cloud)

Google Cloud is a service that allows you to do a whole lot of cloud-based development. We are interested in their virtual machine (VM) service. *Google Cloud*, upon its first use, gives users some free credit to use. You will need some kind of payment method (Credit Card) and need to pay (a minimum of) roughly 5$/month for their services. However, if your service is not in use (for example you have it disabled) you won't be billed.

To start off, we'll have to go to the google cloud website. You can log in if you haven't already. Click on the 'Go to console' button to get started. Click on create a project in order to start, if you already have a project you can click on the dropdown in the top left corner to create a project. The name and organization of the project can be anything. Click on 'View your project'. Sometimes *Google Cloud* will send you to a random subpage like 'Vertex-AI', instead we'll be going back to the console at https://console.cloud.google.com/. Click on 'Create a VM', if you are prompted to enable 'Compute Engine API' do so. You will have to wait a moment for it to be enabled, then retry to create the VM.

The name of the VM can be anything. Select a region and zone based on what is closest to you or what you are expecting most of your players to be the closest to (the price between regions can differ so keep that in mind). The recommended series is N1 as it is the cheapest option. For the machine type select f1-micro under the 'shared-core' tab. The Provisioning model can be left on standard. All the other options can be left on their default values.

After creating a VM is done you can click on the SSH button to open a window containing your VM. You should also see an Internal IP and External IP there. The Internal IP is the hostname used for your server configuration, whilst the External IP will be the hostname used for your plugin configuration. For more information about configuration go to chapter 2.5.

When the SSH window is opened you'll quickly get an authentication prompt, log in to the same google account you are using for your cloud service if you are prompted to. Before we do anything, we will first have to update our virtual machine. This can be done by simply typing 'sudo apt update' and hitting enter. It should start updating the virtual machine and shouldn't take too long. Now we will try to upload our server files. Before we do that, we'll have to update the configuration of the server to use the host and port of the server (for more information on that go to chapter 2.5.2). The hostname of the server will be the Internal IP that is shown on the console, your port can still be 25565 as it is

currently not in use. If you want to use a different port you will have to manually open the ports through the *Google Cloud Firewall* dashboard.

Uploading our server files is very simple, as the SSH window has the option to Upload and Download files at the top of the window. Simply upload the server files onto the virtual machine. To confirm the files have been sent over to the virtual machine type 'ls'. This command will list all files in the current folder. If you only see a folder and not the server files type 'cd FOLDER_NAME' and this will navigate you to inside that folder. Type 'ls' again to confirm the files are in this folder (if you are in the wrong folder you can type 'cd ..' to go back to the previous folder).

Now that we have uploaded our server files you will have to install Ruby onto the virtual machine. This should be very easy, for a Debian or Ubuntu server it should be 'sudo apt-get install ruby-full'. It will start installing Ruby. If a message prompt appears type 'Y' to confirm. After Ruby is installed, you can simply type 'ruby Server.rb' to start the server. If you have done all the steps correctly you should see a message pop up with '*[HH:MM:SS] Server started on HOSTNAME:PORT.*'. If so, you have set up your server correctly. You can now go to the next chapter to start connecting to your server. It is important to remember that to connect to the server you will need the External IP instead of the Internal IP that can be seen on the *Google Cloud* dashboard.

2.3.3. Dedicated Server (Linode)

Linode is a Cloud Computing Service that allows you to host your own virtual machine. It is fairly easy to use, however, it isn't exactly free. Firstly, you will need to have some kind of payment method (*Google Pay*, *PayPal* or Credit Card). You will also need to pay a minimum of 5$/month for their services if you are running a simple server. However, there is a 7 day money back guarantee and they don't bill you instantly, so you can try it out before paying. On top of that, *Linode* frequently has some kind of promotion where new users get some free credit in order to try some of the services.

To start with *Linode*, go to their website and sign in, this can be done with *Google* or by creating an account. For an account you will need to give a username, verify a phone number, and verify a payment method (they won't bill you just yet). They also require a billing address which is the address where you live.

Once you have logged in you can create a Linode from the dashboard. The image is the kind of OS your server will be running on. The default is Debian 11 so that is what we'll be using. The region can be anything, however it is recommended to pick something close to you or wherever you're expecting the most players to be close to. You will want to select from the 'Shared CPU' tab the '*Nanode 1GB*' option

as it's the cheapest. The label can be whatever you want. It will ask you for a strong root password, it is important that you remember this password as it will be required later. Both the SSH-Key and Firewall are not required but can be added later if you want them. VLAN is not supported for all regions but could be useful, however we will not be using it. Finally, you can press 'Create Linode' and you will be done with the setup.

After waiting a little bit for the machine to boot up you can click 'Launch LISH Console' in the top right corner of your Linode dashboard. This will open a new window containing the command window of your Linode. First, we will have to log in onto your machine using the username 'root' and the password you set earlier (when typing the password, you won't see any letters appear, this is for safety reasons, but you are in fact typing). When you are successfully logged into the machine type 'sudo apt update'. This will update everything on the machine to the latest version which is necessary for it to work properly.

Now that we have the virtual machine ready, we will try to upload our server files. Before we do that, we'll have to update the configuration of the server to use the host and port of the server (for more information on that go to chapter 2.5.2). The hostname of the server will be the public IP that is shown on the Linode dashboard, your port can still be 25565 as it is currently not in use. If you want to use a different port use the 'sudo ufw allow PORT_NUMBER' command on the virtual machine (this is not required if you don't have a firewall configured).

To upload our server files, we'll be using *WinSCP*, software used to send SCP packets (files) to other machines. *WinSCP* can be downloaded from their website, once the download is complete open the installation file and use all the default options. After the installation is complete, open the software. Set the protocol to SFTP (if it isn't already) and for the hostname use the same IP address you used for the configuration (shown on the Linode dashboard). The port can stay as 22 and the username and password are the same as you used on the virtual machine (root & YOUR_ROOT_PASSWORD). You will then get a pop up, select 'Yes'. On the left side navigate to your server files and drag them over to the right side. You should get a pop up when it is successful. To confirm the files have been sent over to the virtual machine open the LISH window again and type 'ls'. This command will list all files in the current folder. If you only see a folder and not the server files type 'cd FOLDER_NAME' and this will navigate you to inside that folder. Type 'ls' again to confirm the files are in this folder (if you are in the wrong folder you can type 'cd ..' to go back to the previous folder).

Now that we have uploaded our server files you will have to install Ruby onto the virtual machine. This should be very easy, for a Debian or Ubuntu server it should be 'sudo apt-get install ruby-full'. It will start installing Ruby. If a message prompt appears type 'Y' to confirm. After Ruby is installed, you can simply type

'ruby Server.rb' to start the server. If you have done all the steps correctly you should see a message pop up with *'[HH:MM:SS] Server started on HOSTNAME:PORT.'*. If so, you have set up your server correctly.

You can now go to the next chapter to start connecting to your server. The hostname and port are visible in the LISH window. If you are running into issues with *Linode* you can contact them on their support page.

2.3.4. Alternative Options

If none of these previous options are viable for your use case, you could look into something else entirely. These alternatives won't be explained in as much detail and will require some research and tinkering of your own as they are not recommended.

Firstly, you could ship your server with your game. This is not recommended as users will have access to the server and can mess with data that could cause issues in-game. People will have to follow the steps in option #1 (**Local Server LAN**) to set up their own server. Players will need to install Ruby on their machine to do this, however, so make sure to communicate this to your players. Players will then connect to each other using their own IP-address.

Another option is using an existing server such as your own (if applicable). If you do own your own server (such as an ubuntu server) you will have to make sure you can run Ruby programs on it. Depending on what kind of server you are running this may or may not already be the case. All you will have to do is put the server files onto your server, open an available port for public access (25565 should usually be available). Sometimes you will need to do some extra steps in your firewall to allow outside machines to connect, but after that you will just have to update your '*config.ini*' file to use your public IP-address and the open port. You will then be able to connect to your server by connecting to that IP-address and port.

Some other off-the-shelf server hosting options are Azure, AWS, Hostinger and Contabo. There will be no setup steps provided but the steps to get your server running on these servers should be similar to the steps listed under the #2 option (**Dedicated Google Cloud**). Keep in mind that some of these will require you to pay fees in order to keep your server up and running.

Finally, you could always run the server on your own PC/machine. This will mimic the use of a server, however the drawback of this is that you will be required to keep your machine always running in order to ensure that players are always able to connect. It is also very unsafe as players will be connecting to your network, which can be dangerous.

2.4. Connecting to the server

To start connecting to the server you will first have to make sure that you have the plugin installed correctly and that your server is running. To check if you have the plugin installed just start your game in debug mode. In the terminal/ console you should see a message pop up with '*-> Loaded plugin: Voltseon's Multiplayer Solution (ver. 1.0)*'. To ensure that your server is running go into your server's terminal and run '*ruby Server.rb*' if you haven't already, you should see a message with '*[HH:MM:SS] Server started on HOSTNAME:PORT.'*. If so, you have set up the plugin and server correctly.

After this you should update your plugin's configuration to set the hostname and port to the correct info. Changing the plugin's configuration is further described in chapter 2.5. After this just go into your game and connect to a cluster with the script command '*VMS.join(CLUSTER_ID)*' where *CLUSTER_ID* is any 5-digit code. For testing purposes, you can replace it with 10000 but it could be any 5-digit code. If you have changed your configuration to use the pause menu instead you can also always use the VMS option in the pause menu for it. Make sure that the plugin has the same hostname and port as your server in both configurations as well as all machines and the server being connected to the same network/ internet connection.

If everything went correctly you should be connected to the server. You can confirm this by either checking the console in the server (if you have logging enabled) or your game's terminal (if you have logging enabled and are in debug mode). It is recommended you have a separate machine with your game and plugin (with the same configured host and port) on it as well in order to test if multiple players can join the same cluster without crashing and can see each other correctly.

2.5. Configuration

2.5.1. Plugin

Inside of the plugin's `001_Base` folder is a `001_VMS_Config.rb` file. This is a Ruby script that can be opened with any test editor like Notepad. This file contains a bunch of configuration constants that you can change. All the possible configurations and an explanation follows:

1. '*LOG_TO_CONSOLE*': Whether the game should log to the console. Setting this to 'true' means you will receive logs about game events. Set this to 'false' to not receive any logs. Logs are only visible to players in debug mode as it requires access to the game's console. By default, this is set to 'true'.

2. '*SHOW_SELF*': When set to 'true' the game will also show yourself from the server's perspective, meaning you essentially have a duplicate version of

yourself. This can be useful for testing but isn't recommended for a release version of your game. By default, this is set to 'false'.

3. '*HOST*': This is the hostname or IP address of your server that is used for people to connect to the server. By default, this is set to '"localhost"'.

4. '*PORT*': This is the port of the server. It is an identifier for your server's machine to let it know that all incoming and outbound data through that port is for and from the server. It is important that the port on your server's machine is open for public networks as otherwise you wouldn't be able to join from outside the server's network. By default, this is set to '25565'.

5. '*USE_TCP*': Whether the game uses TCP or UDP. These are protocols used to transfer data. TCP is reliable but slow, while UDP is generally faster but less reliable. By default, this is set to 'false' but for some games TCP would be beneficial. Make sure when you change this to also change the server configuration to use the same protocol.

6. '*TICK_RATE*': This is how many times you will send data to the server per second. It is recommended that this number is somewhere between 20-80. By default, it is set to 60. A higher number means that the server will receive more data per second. However, you can flood the stream and data will become less accurate. It should be set to something that is the same amount, or lower than the server's tick rate. If you want to disable it, set it to 0.

7. '*HANDLE_MORE_PACKETS*': Mostly relevant for UDP, when this is set to 'false' it will ensure that all received data is new. Meaning that some useful packets that help make everything run smoothly are thrown away. By default, this is set to 'true'.

8. '*ADDED_DELAY'*: This acts as a recency bias for the previous configuration, meaning that the higher this is set to, the older the data can be that you accept. By default, this is set to 0.09 seconds.

9. '*TIMEOUT_SECONDS*': How many seconds a player can be idle for before being kicked. Idle means not receiving any data from this player. By default, this is set to 30 seconds, but it is not recommended to set this too low or too high.

10. '*SEED_SYNC'*: Whether to try and sync the seed for all players. A seed or randomization seed is a hash or code that is used by the pseudo random number generator of the game to create randomized values. If all players have the same seed all random events *should* happen at the same time. However, some events

could offset this number such as a wild Pokémon battle. By default, this is set to 'false'.

11. '*THROUGH*': If this is set to 'true' you can walk through other players. Depending on your game you may want to enable or disable this. By default, this is set to 'false'.

12. '*INTERACTION_PROC'*: This is a proc or procedure that is called when interacting with another player. This procedure yields/ is given a player_id, player, and event, meaning you can use these three variables inside of your procedure. 'player_id' is that player's player_id, which is an integer (number). 'player' is the player object that is received from the server, for more info on player objects go to chapter 3.5. Finally, 'event' is a reference to the event you are currently interacting with. If you don't want to be able to interact with other players set this to 'proc { }'.

13. '*INTERACTION_WAIT*': This is how long (in seconds) you will wait to receive an interaction confirmation / denial. Setting this too low could make it so you will cancel an interaction before the confirmation is sent on the server due to latency. Disabling this (by setting it to 0) could potentially have the risk of being infinitely stuck in a loop waiting for the other player. By default, this is set to 30.

14. '*SYNC_ANIMATIONS*': This is an array/ list of animation ids that will be synced with other players. These ids are visible in the Database/Animation tab in RPG Maker XP. If you want all animations to be synced set this to '[]', and if you want none to be synced set it to '[0]'. By default, this is set to '[2, 3, 4]'.

15. '*CULL_DISTANCE*': This is the maximum distance (in tiles) between you and other players before they are made invisible. This should be set to a few tiles past how far the player can see. This is to prevent pop-ins. By default, this is set to 10 tiles.

16. '*SMOOTH_MOVEMENT*': When this is set to 'true' it will start linear interpolating (lerping) other players' positions. Meaning it will try to smooth out their positions to make it less snappy, but also less accurate.

17. '*SMOOTH_MOVEMENT_ACCURACY*': When *SMOOTH_MOVEMENT* is enabled, it will use this number to determine how accurate the smoothing should be. The higher this number, the more accurate the positions will be. Closer to 0 and there will be more smoothing. By default, this is set to 0.5 (halfway).

18. '*SNAP_DISTANCE*': When *SMOOTH_MOVEMENT* is enabled, it will use this number to determine when to snap a player into their position. It will check if the player is currently farther than this number away from their actual position. This number represents a distance in pixels (1 tile = 32 pixels). By default, this is set to 192 (6 tiles).

19. '*ACCESSIBLE_FROM_PAUSE_MENU*': Setting this to 'false' means you cannot access the VMS menu where you can join and disconnect from a cluster from the pause menu. By default, this is set to 'true'.

20. '*ACCESSIBLE_PROC*': This is a simple proc/ procedure that is called to check whether the player can access the menu. For this to be possible 'ACCESSIBLE_FROM_PAUSE_MENU' must be set to 'true'. You can put anything that needs to be checked in this procedure such as '$player.badges > 5' to make the menu only available after the player has received their 6th gym badge.

21. '*MENU_NAME*': This is the name of the VMS menu that will appear in the pause menu. By default, this is set to 'VMS'.

22. '*SHOW_CLUSTER_ID_IN_PAUSE_MENU*': When set to 'true' your cluster id will be shown in the top left corner of the screen when the pause menu is opened. This can be useful as players may need this cluster id to connect to another player. By default, this is set to 'true'.

23. '*SHOW_PING*': When this is set to 'true' your current ping (how long it takes to receive messages from the server in milliseconds) in the game window's title. Your ping will only show when you are connected to a cluster. By default, this is set to 'true'.

24. '*SHOW_PLAYERS_ON_REGION_MAP*': When this is set to 'true' you will be able to see other players live on the region/ town map as they walk around. Do note that having a lot of players connected could make the map start lagging. By default, this is set to 'true'.

25. '*ENCRYPTION_DEFAULTS*': These are default parameters used for initializing basic objects of that class type in a hash format that are used for decrypting these objects. If you are unfamiliar with programming it is best to leave these untouched.

2.5.2. Server

The server comes with a '*config.ini*' file. This is a basic text document that contains simple configurations for your server. You can open the file with any text editor like Notepad. All the possible configurations and an explanation follows:

1. '*host*': This is the hostname or IP address of your server that is used for people to connect to the server.

2. '*port*': This is the port of the server. It is an identifier for your machine to let it know that all incoming and outbound data through that port is for and from the server. It is important that the port is open for public networks as otherwise you wouldn't be able to join from outside the server's network.

3. '*check_game_and_version*': Whether the server should check for players' game and version. This is useful to ensure players from different games or versions of your game don't try to connect to the server and crash other players.

4. '*game_name*': The name of your game, the server will only accept players who are trying to connect to the server from a game with this name. The name of your game can be seen and changed in *RPG Maker XP* by clicking on '*Game*' on the top bar and clicking on '*Change Title…*'.

5. '*game_version*': The version of your game, the server will only accept players who are trying to connect to the server from a game with this version. You can change the version of your game by going into your game's scripts and changing the '*GAME_VERSION*' configuration under '*001_Settings.rb*'.

6. '*max_players*': This is how many players are allowed onto one cluster. This does not mean a total of allowed players connected to the server. If too many players are connected to each other in the same cluster, it could cause lag for all players. Setting it to 4, or anything below 8 is recommended.

7. '*log*': Whether the server should log to the console. Setting this to 'true' means you will receive logs about server events. Set this to 'false' to not receive any logs.

8. '*heartbeat_timeout*': How many seconds a player can be idle for before being kicked. Idle means not receiving any data from this player. By default, this is set to 5 seconds, but it is not recommended to set this too low or too high.

9. '*use_tcp*': Whether the server uses TCP or UDP. These are protocols used to transfer data. TCP is reliable but slow, while UDP is generally faster but less reliable. By default, this is set to 'false' but for some games TCP would be

beneficial. Make sure when you change this to also change the plugin configuration to use the same protocol.

10. '*threading*': Whether the server should use threading for each cluster. This will make all clusters run asynchronously but will cost more memory. By default, this is set to 'true'.

11. '*tick_rate*': This is how many times per second the server will send data. This should be between 20-80. The higher the number, the more frequent the updates are sent. Anything higher than 80 would be too much data/ overkill. While anything under 20 is usually not enough data (depending on your type of game). By default, this is set to 60 ticks per second.

When changing the configuration of the server make sure to restart your server. Making changes to the config file can either crash the server or leave it on an outdated version (depending on your sever hosting solution).

2.6. Removing the plugin

Removing the plugin is a simple task. As the plugin does not require any extra files or folders. You should just be able to remove the folder from your 'Plugins' folder. In case none of your other Plugins require '*rainefallUtils*' and your game does not make use of any of the utilities provided by the plugin feel free to remove that as well.

The same can be done for the server, as it doesn't depend on other folders or files you can simply delete the server file from wherever it is currently stored. If your game still reads and detects the plugin you could try recompiling plugins by deleting the '*PluginScripts.rxdata*' file found in your 'Data' folder and reopening your game in debug mode.

3. Script understanding

3.1. Base

This section will be discussing the '*001_Base*' folder in the plugin files. All scripts in this folder are Ruby scripts and can be opened using any text editor such as Notepad.

Firstly, '*001_VMS_Config.rb*' is a script containing all configuration data. This is where all base-level changes can be made that most people will want to change, such as the connecting server's host and port. For more in-depth details about the configuration look at chapter 2.5.1.

Secondly, '*002_VMS_Message_Config.rb*' is a collection of all messages that are displayed directly to the player (not console messages). Some of these have '{1}' and '{2}' in the text. These represent variables that are handled in the script, such as a player's name or a Pokémon's name. You can disable any message by setting it to "".

Lastly, '*003_VMS_Connection_Handler.rb*' is a complicated script as it handles all of the direct connections with the server and processes received and sent data.

1. We have the 'join' and 'leave' methods which, as their names imply, joins, and leaves a cluster.

2. The 'update' method is called each graphic update and handles showing your ping, as well as sending and receiving server data. It also checks if someone is trying to interact with you or if a player has timed out.

3. The process method turns received data into a player and syncs the seed if possible. It also handles event creation/ deletion for players.

4. The 'clear_events' method simply iterates through all players and deletes their events.

5. Then we have 'clean_up_events' iterates through all events on your map and deletes all player events that shouldn't be there.

6. The 'send_message' method simply sends a message to the server.

7. We have 'generate_player_data' which is one of the most important methods if you are trying to change the data that is sent through the server. This creates a hash of all data that you are trying to send. This hash is read out again in the '002_Players/001_VMS_Player_Handler.rb' 'handle_player' method and '005_Server_Data/001_VMS_Player.rb' 'update' method.

3.2. Players

This section will be discussing the '*002_Players*' folder in the plugin files. All scripts in this folder are Ruby scripts and can be opened using any text editor such as Notepad.

Firstly, '001_VMS_Player_Handler.rb' is a script that has everything to do with handling received player data and interacting with that data.

1. The 'interact_with_player' method will try to send an interaction to another player.

2. The 'check_interaction' method will try and receive an interaction. When an interaction is received, it will start to wait for the other player to send an action or to cancel. The player can also cancel themselves if they want to stop interacting.

3. The 'send_interaction' method will wait until the other player has started to also interact with you. It will then give you the option to Swap, Trade, Battle or Cancel the interaction. Swap will give the other player the option to pick, while the others are self-explanatory. It will then wait for the other player to confirm.

4. The method 'create_event' simply creates a new event based on the given player.

5. The 'check_timeout' method checks if the player has timed out, meaning that they haven't sent you a packet in a while.

6. There is a method for 'player_still_connected' which will check if the given player still exists in the server, if so return that player, otherwise return 'nil'.

7. The 'await_player_state' method is an abstracted version of waiting for the given player's state. This method is used a lot in back-and-forth communication between two players that require input from both players.

8. The 'update_party' method simply changes the stored player's party to a dehashed version in case it was still hashed.

9. There is a 'sync_animations' method which will go through all sent animations by the given player and play them if they aren't already playing.

10. Finally, there is a ‘handle_player’ method which is called everytime player data is received. It tries to update the party data, sync given animations, and does any event related edits if possible and necessary.

Secondly, ‘002_VMS_Trade_Handler.rb’ is a script that has the required method to perform a trade, it firstly checks if both players are connected. It checks if both players have any tradable Pokémon. If so, let the players pick the Pokémon they want to trade from their party. You cannot pick from your boxes as only party data is being synced on the server. Then there is a cancellation check and a wait for the other player to select a Pokémon. After this, both players can do a final check on both their own Pokémon and the Pokémon they will be receiving. You can either confirm or cancel this trade. If both players confirm, a simple trade is performed, and you are forced to save (in order to prevent Pokémon duplication).

Finally, there is a ‘003_VMS_Battle_Handler.rb’ script which has everything to do with battling. All that really happens here is a bunch of script overrides of base *Pokemon Essentials*’ battles to make a battle between players possible by turning the AI that is normally used into an AI that makes decisions based on data received through the server. This is all being done by sending choice data in the player state variable.

3.3. User Functions

This section will be discussing the ‘*003_User_Functions*’ folder in the plugin files. The script in this folder is a Ruby script and can be opened using any text editor such as Notepad.

The only script that can be found in here is the ‘001_VMS_User_Functions.rb’ script which contains a list of methods that would commonly be used in development.

1. ‘*VMS.get_variable*’: This method returns an online (shared) variable from the cluster with the given id. If the variable doesn’t exist, it will return *nil*.

2. ‘*VMS.set_variable*’: This method will set an online (shared) variable of the server to the given value. If the variable doesn’t exist yet, it will create it.

3. ‘*VMS.ping*’: This method simply returns the ping of the player in seconds.

4. ‘*VMS.sync_seed*’: This method uses all connected player’s ids and names as well as the cluster id to set the randomization seed.

5. ‘*VMS.see_party*’: This method will open a party screen of the given player. Any changes made in this screen are not saved, however be careful when

using this with a box link or giving/ taking items from the Pokémon, as this will duplicate/ delete Pokémon and items. It is recommended that this method be only used in testing environments.

6. '*VMS.teleport_to*': This method will teleport you to the specified player by their id.

7. '*VMS.is_connected?*': This method simply gives a true or false depending on if you are connected.

8. '*VMS.get_self*': This method will return the player object of yourself as it is on the server.

9. '*VMS.get_player*': This method will return the player object of the given player.

10. '*VMS.get_players*': This method will return an array/ list of all players that are connected to your cluster including yourself.

11. '*VMS.get_player_count*': This method simply returns the number of players connected to your cluster.

12. '*VMS.get_cluster_id*': This method simply returns the id of the currently connected cluster.

13. '*VMS.interaction_possible?*': This method checks if the player can interact. This is used for other players to see if they can interact with you.

14. '*VMS.get_interaction_time*': This method returns the number of frames a loop should be to wait for the '*INTERACTION_WAIT*' time given in the config. If this is disabled it will go for the maximum integer limit of ruby number of frames (4611686018427387903).

15. '*VMS.hash_pokemon*': This method simply turns a Pokemon object into a string.

16. '*VMS.dehash_pokemon*': This method turns a hashed Pokémon back into a Pokemon object and recalculates its state appropriately.

17. '*VMS.clean_up_basic_array*': This recursive method goes through all elements of a given array and removes all data that isn't a serializable/ basic type (string, symbol, integer, boolean, etc.).

18. '*VMS.scene_update*': This is a simplified version of the game's update method, mimicking it for certain loops.

19. '*VMS.event_deletion_possible?*': This method checks if it is possible to delete the event of a player by checking if the event exists and if the player can delete the event.

3.4. Other

This section will be discussing the '*004_Other*' folder in the plugin files. All scripts in this folder are Ruby scripts and can be opened using any text editor such as Notepad.

Firstly, the '001_VMS_Overrides.rb' script contains everything of base *Pokemon Essentials* that the plugin has to override in order to function properly. The most important override being the '*vms*' attribute in *Game_Temp* all the way at the top of the script. This will contain all data the player has to store whilst connected. This is stored in game temp as we don't want to store this data on the save file.

Lastly, the '002_VMS_Utilities.rb' script contains a few simple methods that provide some abstract methods that are used in the script.

1. '*VMS.basic_type?*': Checks if the given variable is serializable/ basic.

2. '*VMS.encrypt*': This recursive method turns any data into a hash containing its data type and the stored data.

3. '*VMS.decrypt*': This method does the opposite of encrypt, restoring the data.

4. '*VMS.string_to_integer*': Simply turns a string into an integer using a prime.

5. '*VMS.array_compare*': Simply checks if the contents of the two given arrays are the same.

3.5. Server Data

This section will be discussing the '*005_Server_Data*' folder in the plugin files. The script in this folder is a Ruby script and can be opened using any text editor such as Notepad.

The sole script in this folder is '001_VMS_Player.rb' which contains the player class. This class is used to build up a player object containing all data related to connected players. Because this class is so valuable it is used in both the plugin and on the server. For this reason, any data that is changed in this script has to be

copied over onto the same script 'Player.rb' on the server. Some notable data that is stored and methods used are as follows:

1. '*id*': This is the id of the player used to identify them. Your own player id can also be found by using '$player.id' and a shorter version is seen on the trainer card.

2. '*address*': This is the IP address of the given player, for obvious reasons this is only stored on the server and is not sent to other players. It is also only used with UDP and not for TCP.

3. '*port*': This is the connected port of the given player. It is also only stored on the server and only used for UDP.

4. '*can_be_nil*': This is only useful for the server and is not used by the player. It is a simple array of all the attributes that can be nil. Anything that isn't in this array will not be overwritten when a nil is received.

5. '*VMS::Player.update*': This method will iterate through all received data and update the current object based on this data. It will also locally set the hearbeat.

6. '*VMS::Player.to_hash*': This method transforms the current player object into a hash. Some data is skipped, and floats are rounded to 3 decimal places.

3.6. Server

The server contains 4 Ruby scripts and one '*ini*' file. Starting with the 'config.ini', this is a configuration file containing anything that may frequently be changed. More info about changing the configuration can be found under chapter 2.5.2.

The 'Cluster.rb' script contains the basic class for a cluster. A cluster is basically a branch of the server that connects some players to each other. This is useful if you don't want all players in your game to be connected to each other but allow players to host their own lobbies/ clusters (refer to the image below as to how clusters and the server works).

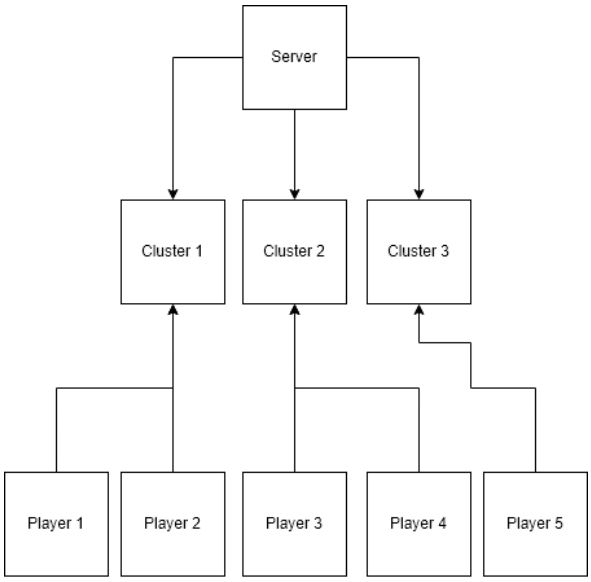


Figure: Server/Cluster/Player structure in a diagram.

The ‘Config.rb’ script is a simple utility script that allows the other scripts to easily access the ‘config.ini’ file.

The ‘Player.rb’ script contains the player class. This class is used to build up a player object containing all data related to connected players. Because this class is so valuable it is used in both the plugin and on the server. For this reason, any data that is changed in this script has to be copied over onto the same script ‘005_Server_Data/001_VMS_Player.rb’ in the plugin. For more details on the player class go to chapter 3.5.

Finally, the ‘Server.rb’ is the meat of the server scripts. This script contains all logic for the server. All the server does is listen for incoming data and create, update, and delete clusters accordingly.

4. Modifying and Adding

4.1. Changing your server communication

VMS allows for two protocols that are very commonly used in videogame networking: TCP and UDP. TCP provides reliable, ordered, and error-checked delivery of data, making it suitable for games requiring high reliability. UDP, on the other hand, is simpler and provides no guarantees, making it more suitable for time-sensitive games prioritizing speed and efficiency over reliability. Generally, UDP is recommended, however TCP can be useful in some cases.

By default, the plugin uses UDP, but changing it is simple as can be seen in their respective sections in chapter 2.5 where you will just have to change a value in the server's and plugin's configuration. Make sure to restart the server after changing the protocol.

Changing to something else entirely (like HTTP) is possible, however it is not recommended. TCP and UDP are industry standards for videogame communication as they are generally faster. Changing the protocol to anything outside of those two is not supported but can be done.

4.2. Reducing latency

Optimization is important in order to make the player's experience as smooth as possible. Since VMS is not made by a huge professional team but by one student it is expected not to be completely perfect. Also, since the server and plugin are both written in Ruby, a programming language that is not known for its speed, it can be expected to not run as fast as it could be. Disclaimer, all options listed below require scripting knowledge.

A first step that could be taken to reduce latency, or in other terms, to receive data from the server faster, is to send smaller packets. This means reducing the number of variables that are being sent to the server. All data that is being sent to the server is found in '/005_Server_Data/001_Player.rb' and the same file in the server's 'Player.rb' script.

Another option is to further encrypt all the data being sent. Currently all encryptions being done is through the Marshal and zlib modules. However, converting data to some type of binary packet is what most companies currently use, so looking into that could prove useful.

Finally, some more optimizations could be made in how many packets are being sent and received per second. Currently the server updates all players at a rate that is the maximum iterations per second that Ruby or your server's machine

allows. Meanwhile your game will send a packet every Graphic update. However, to optimize further you could also only send a packet anytime a change happens.

5. Game Examples

This is a section reserved for highlighting games that make use of VMS. Unfortunately, as of right now, there are no games to highlight in this section…

6. Version History

v1.0

The first full release version of the script, containing a bunch of bug fixes and general improvements.

Released on: January 3, 2024

Changelog:

- Added support for game and version authentication by the server.
- All data is further compressed with 'zlib' to ensure that packets aren't too large.
- Added '*Online Variables*' which are essentially variables that are shared between all players in a cluster.
    - To set an online variable use: '*VMS.set_variable(id, value)*'.
    - To get an online variable use: '*VMS.get_variable(id)*'.
- Made it so other events can walk and interact straight through other players.
    - This is made possible by making all other player events through whenever an event starts running/ the map interpreter starts running.
- Made it so you'll attempt to disconnect from a cluster when closing the game, rather than waiting for you to timeout.
- Added a Message Configuration that houses all player shown messages.
- Made it so you can use items in battles.
- Server:
    - Provided a better method for ensuring all data is sent to and received by the server.
    - Changed the server terminal to have some colors in it.
    - Fixed a crash that would happen if the server disconnected you from a cluster.
    - Added a threading option to make clusters run asynchronously.
    - Added a tick rate option to limit the amount of data sent per second.
- Added a tick rate option to the players side to limit the amount of data the player sends.
- Added some configurable settings for smoothing the movement of players. This makes movement slightly less accurate but will make it seem more natural.
- Made cull distance (how far away player has to be before they are invisible) a configurable option.
- Added the option to only receive the newest data, with a configurable delay so you can get both fast and reliable data.
- Players will now hold an '*is_new*' attribute that simply returns whether the current version of the player is from a new packet. This is only relevant when *HANDLE_MORE_PACKETS* is set to 'false'.
- Both players will need to have able Pokémon to start a battle.
- Added a method to teleport to other players.
- Made players calculate their bush depth.

v0.9

Initial release version containing basic server client communication and providing some player features.

Released on: December 16, 2023

Changelog:

- There is a Ruby server that keeps track of clusters and can send data to players.
- Clusters keep track of connected players and will send data to all players.
- Players can connect to a cluster using TCP or UDP.
- Players only send data that is new to the cluster to keep packets small.
- Pokémon data is compressed so the cluster can quickly receive your party.
- Connect with a server using a menu option.
- See your current cluster ID in the pause menu.
- Player syncing:
    - See other players' characters.
    - The surf base is visible and moves with them.
    - See what direction they are facing and their current animation frame.
    - Collision can be toggled through the config.
    - See when a player is jumping.
    - Players are live visible on a map.
    - Player events are created and destroyed when they join/ disconnect.
    - Player events are created and destroyed when they leave/ enter a map.
    - Animations are synced (which ones are synced is configurable)
- Player interactions:
    - You can interact with another player and wait until they receive your interaction.
    - You cannot interact with a player that is busy (an event is running, in a battle, interacting with another player, etc.)
    - During an interaction you have the option to switch the lead, meaning the other player gets to decide what happens.
    - You can ask the other player to trade, after which you will have to select a Pokémon from your party. Once both players have agreed a trade will commence.
    - You can ask the other player to do a Single Battle. This battle is not configurable, and you have the option to fight, switch or forfeit. It is a live battle.
- Your ping to the server is calculated and shown in the title of your window.
- Players are timed out after they don't send any data for a while (e.g. internet disconnected)

7. Support

7.1. Contact

If you have any questions, you can try contacting me (Voltseon) through any of the outlets listed below.

RelicCastle: Voltseon's Multiplayer Solution Thread
RelicCastle: Direct messages
Email: voltseon@gmail.com
X (formerly Twitter): @RealVoltseon

7.2. Credits

Plugin, Server and Documentation written by:	Voltseon
Plugin testing by:	Voltseon, ENLS & KennyCatches
Research assistance by:	ENLS
Documentation proofread by:	ENLS & KennyCatches
rainefallUtils by:	rainefall