

Cable Club

User Manual

Original Script By mGriffin

Commissioned By Khaikaa

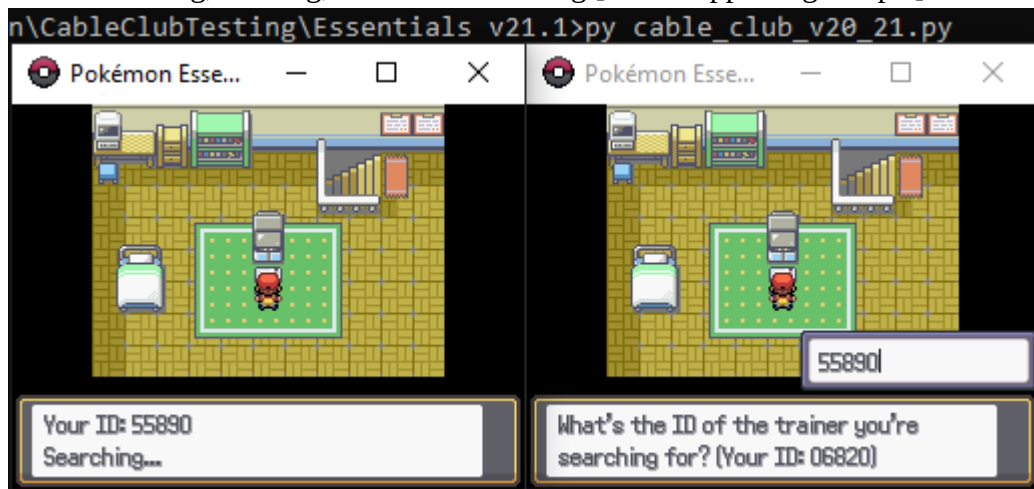
Maintained By Vendily

Table of Contents

Features.....	3
Installing the Script.....	4
Step 0: Installing the Hotfixes.....	4
Step 1: Dropping in the Files.....	4
V16\17\18 Instructions: Insert the Script Sections.....	4
V19+ Instructions: Insert the Plugin.....	4
Step 2: Setting up the Python File.....	4
Step 3: Adjusting the Ruby Settings.....	5
Step 4: Giving Access to the Cable Club.....	6
Step 5: Testing Locally (Requires Python 3.6+).....	6
Common Issues.....	6
FileNotFoundError: [Errno 2] No such file or directory: './PBS/abilities.txt'.....	6
"I'm sorry, the Cable Club server is down at the moment.".....	6
"I'm sorry, your party contains Pokémon not allowed in the Cable Club.".....	7
Cable Club's Record Mixer & Events \ EventHandlers.....	8
Defining a Record Mixer Entry.....	8
Sending Data with the RecordWriter.....	9
Sending an Array.....	9
Making Helper Methods.....	9
Receiving Data with the RecordParser.....	9
Receiving an Array.....	10
Working with Data Types.....	10
The Server's RecordParser and RecordWriter.....	11
Example RecordMixer Entry.....	11
Events & EventHandlers.....	11
Cable Club on Google Cloud.....	12
Getting the Virtual Machine Set Up.....	12
Step 0: Fixing the HOST on the Server.....	12
Step 1: Create the VM.....	12
Step 3: Reserving the External Static IP Address.....	14
Step 4: Setting up the Firewall.....	14
Step 5: SSH-ing into the VM Instance.....	15
Setting the Startup Script.....	16
Step 0: Fixing the PBS_DIR and LOG_DIR on the server.....	16
Step 1: Adding the Startup Script.....	16
Step 2: Restarting the Server.....	17
Can't Connect After Restarting.....	17
Known Issues.....	18
All Versions.....	18
V16.....	18
V17.....	18
V18.....	19
Third Party Scripts.....	19

Features

- Online Battling, Trading, and Record Mixing [With Supporting Scripts]



- User configurable Online Trainer Types



- Compatibility with:
 - Essentials Deluxe
 - ZUD Mechanics
 - Terastal Phenomenon
 - Focus Meter System
 - PLA Battle Styles

Installing the Script

Cable Club's installation is almost the same regardless of which version of Essentials you are on.

! The following instructions will refer to the server python file as `cable_club.py`.
Later versions of the script have a suffix with the respective version number in
● the file name. Whenever `cable_club.py` appears, replace it with your respective version's filename.

Step 0: Installing the Hotfixes

It's not technically required, but better to have the fixes since battle related fixes are a part of the hotfixes, and the last thing you need in your online battles is a crash. That'll end the session prematurely.

Step 1: Dropping in the Files

V16\17\18 Instructions: Insert the Script Sections

There are two files, `cable_club.rb`, and `record_mixer.rb`, which must be installed with the `cable_club.rb` Script Section above the `record_mixer.rb` Script Section.

! Do **NOT** insert the Python file in the Script Editor.
● The Python file does not need to be inserted into the Script Editor. This will cause errors as it is not Ruby code.

V19+ Instructions: Insert the Plugin

The respective folder for your version can be copied and pasted into the root folder of your project. This will cause the Plugins folder to merge with the one in your project installing the script.

Step 2: Setting up the Python File

All versions have some constants at the top that should at least be confirmed to match with your project, such as installed plugins, or may need later modification when it comes to uploading to the cloud [See Cable Club on Google Cloud]. Some constants are only for specific versions, and are labeled as such.

Constant Name	Explanation
HOST	The IP address this server should attempt to set up. Almost always set to either <code>127.0.0.1</code> (localhost) or <code>0.0.0.0</code> (Google Cloud)
PORT	The Port the server is listening on.

Constant Name	Explanation
PBS_DIR	The path to the PBS folder relative to the executing location (not the same as the location the file is located).
LOG_DIR	The path to the folder where the server . log file should be created. This folder should have write access.
GAME_VERSION	[v19+] The earliest version of the game that should be allowed to connect to the server. Versions older than this would be rejected on connection.
POKEMON_MAX_NAME_SIZE	The maximum length of a Pokémon's name. May need to be set higher if a species name is longer than the max the player can input in game.
PLAYER_MAX_NAME_SIZE	The maximum length of the Player's name.
MAXIMUM_LEVEL	The maximum level a Pokémon can reach.
IV_STAT_LIMIT	The maximum IVs a Pokémon can have in one stat.
EV_LIMIT	The total maximum EVs a Pokémon can have.
EV_STAT_LIMIT	The maximum EVs a Pokémon can have in one stat.
SKETCH_MOVE_IDS	An array of Internal IDs of moves that have the Sketch function code. This affects the normal validation checks, allowing for any defined move to be on the Pokémon, not just the ones defined for the species.
EBDX_INSTALLED	[v19-] Set to true if EBDX (not EBS) is installed.
VERSION_18	[v18-] Set to true if this server is for a v18 project.
DELUXE_INSTALLED	[v20+] Set to true if Essentials Deluxe is installed.
ZUD_INSTALLED	[v20+] Set to true if ZUD Mechanics is installed.
PLA_INSTALLED	[v20+] Set to true if PLA Battle Styles is installed.
TERA_INSTALLED	[v20+] Set to true if Terastal Phenomenon is installed.
FOCUS_INSTALLED	[v20+] Set to true if Focus Meter System is installed.

Step 3: Adjusting the Ruby Settings

All versions of the script have the same settings.

Constant Name	Explanation
HOST	The IP address the script should attempt to connect to. Almost always set to either 127.0.0.1 / localhost or the External Static IP address from your host.
PORT	The Port the script should connect to. Always set to the same port as the Python file.

Constant Name	Explanation
ONLINE_TRAINER_TYPE_LIST	An array of Trainer Types the player can select to appear as online with <code>pbChangeOnlineTrainerType</code> . Each entry should be a Trainer Type ID or an Array with a Trainer Type ID that corresponds to each gender the player character can be, for gender-locked options. You can mix and match both types together.
ENABLE_RECORD_MIXER	Set to true to give the option for Mixing Records to be selected in the Cable Club's activity list.
DISABLE_SKETCH_ONLINE	When true, Sketch fails when used on online battles, like in Gen 2 games. When false, the moves are restored back the way they were before the battle started, like in the Gen 3+ games.

Step 4: Giving Access to the Cable Club

The specifics really depend on your project, but calling `pbCableClub` will allow the player to attempt to connect to the server, so long as they have at least 1 Pokémon in their party and the party passes validation.

Step 5: Testing Locally (Requires Python 3.6+)

Now that everything is set up, we can force a compile to ensure the plugin is installed, and copy the python file into the root folder of your project if it's not already there. Using the Command Prompt, navigate to your game's folder. There, you can run `py cable_club.py` to start the server, and open two windows of your game to test.

Common Issues

FileNotFoundError: [Errno 2] No such file or directory: './PBS/abilities.txt'

The exact path for the filename will vary, but if it is `abilities.txt`, then your `PBS_DIR` is not set up correctly. With the Local Testing setup above, the execution folder is the same folder the `cable_club.py` is located, the root folder, meaning that the PBS folder is in the sub folder `/PBS` relative to it. (The `.` is the current execution folder.)

If it's not `abilities.txt` that it failed to find, then you are missing a PBS file. This error is more likely to occur on the cloud.

"I'm sorry, the Cable Club server is down at the moment."

This error message appears in game if the game fails to connect to the server at all. Double check that the `HOST` and `PORT` settings in the server and script are set up correctly. The `PORTS` should always match. For local testing, both should have the `HOST` set to `127.0.0.1`, while for Google Cloud tests, the `HOST`

on the server should be set to 0.0.0.0, while the script has the HOST set to the External Static IP address for the server.

“I'm sorry, your party contains Pokémon not allowed in the Cable Club.”

This error message appears in game if the server rejects the party due to the server side validation failing. If you shut down the server, you can run the command with a lower logging level with `py cable_club.py --log=DEBUG` to get a full report of all issues with the party. How exactly you fix the error depends on the exact cause for the error, with the most common one being a mismatch between the amount of data sent by the script, and the amount processed during validation on the server, causing off by one errors.

The debug party is known to trip the validation errors.

Cable Club's Record Mixer & Events \ EventHandlers

Cable Club has a Record Mixer feature, from the Gen 3 and 4 games, which allows script creators to register data to be sent between players playing online together. To set up your script with the Record Mixer, you only need to define the Cable Club as an optional requirement.

Essentials Version	Minimum Version for Optional Requirement
v16, v17, v18	1.7
v19	2.2
v20, v21	3.1 (3.5 for v21)

Defining a Record Mixer Entry

A Skeleton Record Mixer Entry, which can be used as a base, looks like this:

```
if defined?(RecordMixer)
  RecordMixer.register(:test_record,{
    "name" => proc { _INTL("Test Record")},
    "prepareData" => proc {},
    "writeData" => proc {|writer|},
    "parseData" => proc {|record|},
    "finalizeData" => proc {}
  })
end
```

Below is a longer explanation of each:

Property	Explanation
:test_record	The id for this record internally. This should be unique so prefixing it with the name of your script is ideal.
"name"	A proc with no arguments that returns the name displayed to the player when processing this record.
"prepareData"	An optional proc with no arguments. If included, it runs before data is written and parsed, allowing for pre-transfer edits if necessary.
"writeData"	A proc with 1 argument, a RecordWriter object. You can send data via this writer using the methods on it. All data written must be parsed in the same order. [See Sending Data with the RecordWriter]
"parseData"	A proc with 1 argument, a RecordParser object. You can receive data via this writer using the methods on it. All data must be parsed in the same order it is written. [See Receiving Data with the RecordParser]
"finalizeData"	An optional proc with no arguments. If included, it runs after all data is written and parsed, allowing for post-transfer edits if necessary.

The `if defined?(RecordMixer) / end` is necessary if this is an optional requirement, so that your script continues to function without the Cable Club being installed. v16\17\18 versions are also not guaranteed to have installed the Record Mixer script section.

Sending Data with the RecordWriter

The RecordWriter has a number of methods available to it for sending different data types. More complicated forms of data sending can be formed by combining these available methods. Internally, all of these different methods convert to string.

Method	Explanation
<code>bool(b)</code>	Adds a boolean value for sending.
<code>int(i)</code>	Adds an integer value for sending.
<code>str(s)</code>	Adds a string value for sending.
<code>sym(s)</code>	Adds a symbol value for sending.
<code>nil_or(t, o)</code>	Takes 2 arguments, a symbol version of one of the methods above, and the value to be sent, which can be nil. <code>nil_or(:sym, pkmn.item_id)</code> sends the <code>item_id</code> , which can be nil if the Pokémon is not holding an item.

Sending an Array

To send an array, first send the length of the array with `writer.int(i)`. Afterwards, loop over the array and add each element as normal. Sending the length allows the RecordParser to know how many elements to expect.

```
writer.int(pkmn.numMoves)
pkmn.moves.each do |move|
  writer.sym(move.id)
  writer.int(move.ppup)
end
```

Making Helper Methods

Sometimes, it's too complicated to do everything all at once. Creating helper methods that individually send data break down the problem into something more simple. You just need to pass the writer, and any data the helper method would need to be able to process it.

Cable Club itself has a `write_party` method, which calls a `write_pkmn` method.

Receiving Data with the RecordParser

The RecordParser has similar methods to the RecordWriter, except instead of sending data it is receiving it. While internally, all the data sent online is converted to strings, the RecordParser will convert it back, meaning it will error if it gets data it does not expect.

Method	Explanation
<code>bool</code>	Converts the current data segment into a boolean and returns it
<code>int</code>	Converts the current data segment into an integer and returns it
<code>str</code>	Converts the current data segment into a string and returns it
<code>sym</code>	Converts the current data segment into a symbol and returns it
<code>nil_or(t)</code>	Takes 2 arguments, a symbol version of one of the methods above, and the value to be sent, which can be nil. <code>nil_or(:sym)</code> expects to receive either a symbol or nil.

Receiving an Array

To receive an array, first get the length of the array with `record.int`. With this, we can loop that many times until we have the full array.

```
record.int.times do |i|
  pkmn.add_first_move(record.sym)
end
```

Working with Data Types

Sometimes the data you need to send or receive is complex. Perhaps it can be nil but if it's not, you don't have an easy data type you can pass with `nil_or`. In these situations, you can play with sending extra data, or sending data types manually.

Below is an excerpt from `write_pkmn`, for sending fused Pokémon

```
writer.bool(!pkmn.fused)
if pkmn.fused
  write_pkmn(writer, pkmn.fused)
end
```

It's easier to know if there is or isn't a fused Pokémon by directly sending a boolean, and any data type can be forced into a boolean by putting `!!` in front of it, as it converts it into a boolean, and then reverses it so that it matches the truth type of the original. (nil and false are false, everything else is true.)

Below is the equivalent section in `parse_pkmn`

```
if record.bool() # fused
  pkmn.fused = parse_pkmn(record)
end
```

Since we have the boolean from earlier, we don't need to worry if the next Pokémon we process is a fused one or a normal party member, as the boolean tells us that yes, this is a fused Pokémon.

Other times you may need to send dummy data by itself, such as just a `nil`. `nil` internally is an empty string, so you can send `writer.str("")`. Alternatively, you can use `nil_or`, but pass `nil` instead of a variable, with `writer.nil_or(:bool, nil)`.

The Server's RecordParser and RecordWriter

Generally, you don't need to modify the server, unless you are sending more data with the party, which would require being processed as well.

Server RecordParser	Server RecordWriter	Script RecordParser	Script RecordWriter
<code>bool</code>	<code>-</code>	<code>bool</code>	<code>bool(b)</code>
<code>bool_or_none</code>	<code>-</code>	<code>nil_or(:bool)</code>	<code>nil_or(:bool, b)</code>
<code>int</code>	<code>int(i)</code>	<code>int</code>	<code>int(i)</code>
<code>int_or_none</code>	<code>-</code>	<code>nil_or(:int)</code>	<code>nil_or(:int, i)</code>
<code>str</code>	<code>str(s)</code>	<code>str</code>	<code>str(s)</code>

As Python does not have symbols, symbols should be read as strings on the server. `nil` is an empty string, so if you must check for a `nil`-able `str` or `sym` field, you can check it against an empty string if it must be checked. Empty strings are considered `false` for the sake of conditionals as well.

Example RecordMixer Entry

Below is an example `RecordMixer` entry that swaps Game Variable 100 between the two variables. It does not need to finalize any data, and does not use that property.

```
if defined?(RecordMixer)
  RecordMixer.register(:variable_swap, {
    "name" => proc { _INTL("Swap Variables") },
    "writeData" => proc {|writer| writer.int($game_variables[100]) },
    "parseData" => proc {|record| $game_variables[100] = record.int }
  })
end
```

For a more complex example, the `Secret Bases Remade` script sends all secret bases on the save file to the other player, and then increases the bases sent variable on the base, so that older copies of a base do not replace newer copies.

Events & EventHandlers

`Cable Club` has one Event (`v19-`) \ `EventHandler` (`v20+`) defined, for when the `online_trainer_type` is changed, providing one argument, the new trainer type.

Essentials Version	Minimum Version for Optional Requirement
<code>v19-</code>	<code>CableClub.onUpdateTrainerType+= proc { new_ttype }</code>
<code>v20+</code>	<code>EventHandlers.add(:cable_club_trainer_type_updated, proc { new_ttype })</code>

Cable Club on Google Cloud

Cable Club can be hosted on a VM on [Google Cloud](#), acting as a dedicated server. Google Cloud has a 3 month Free Trial, with \$300 worth of trial credit, as well as a Free Tier, provided that you do not exceed their limits ([See Compute Engine Here](#)).



Google Cloud requires a credit card to be attached to your account, even if you only use within the Free Tier limits. If you are unable to use a credit card, you **can not** use Google Cloud.



If you wish to use a different host, these instructions may not match up entirely with that of your hosting provider. I am unable to provide support for providers other than Google Cloud.



Be sure to confirm what the Free Tier consists of, as it may change over time. If the instructions conflict with what is allowed as the Free Tier, follow the Free Tier.



The estimate for the VM instance we create here is about \$5.25 USD without the Free Tier limits. But it should only charge for network traffic beyond the provided 1GB or to Australia or China. The amount varies depending on how much traffic you get for your game.
Usage Estimates from users of Cable Club estimate about \$5 USD per month at most, though the cost is generally lower than that for Google Cloud.

Getting the Virtual Machine Set Up

Step 0: Fixing the HOST on the Server

The HOST constant for the server python file must be set to 0.0.0.0.

Step 1: Create the VM

From the Sidebar Menu in the top left, scroll down, hover over “Compute Engine” and select “VM Instances”. On this new page, select “Create Instance” from the top.

The next page displays your selections as well as a billing calculation. Despite the cost on the right side, if you do meet the Free Tier requirements, you won’t be charged.

The Region must be set to either us-west1 (Oregon), us-central1 (Iowa), or us-east1 (South Carolina).

The Series is E2, and the Machine type is e2-micro.

The availability must be set to Standard.

Region *
us-central1 (Iowa) ▼ ?
Region is permanent

Zone *
us-central1-a ▼ ?
Zone is permanent

Machine configuration

✓ General purpose

Compute optimized

Memory optimized

GPUs

Machine types for common workloads, optimized for cost and flexibility

Series
E2 ▼
CPU platform selection based on availability


Machine type

Choose a machine type with preset amounts of vCPUs and memory that suit most workloads. Or, you can create a custom machine for your workload's particular needs. [Learn more](#)

PRESET

CUSTOM

e2-micro (2 vCPU, 1 core, 1 GB memory) ▼



vCPU
0.25-2 vCPU (1 shared core)

Memory
1 GB

✓ ADVANCED CONFIGURATIONS


Availability policies

VM provisioning model
Standard ▼

A configured VM Instance for Free Tier.

The next few settings can be skipped over, until the Boot Disk. I can confirm that Ubuntu 20.04 LTS works with the Cable Club, as it has Python 3.8 installed.

Boot disk ?

Name	instance-1
Type	New balanced persistent disk
Size	10 GB
License type ?	Free
Image	 Ubuntu 20.04 LTS

CHANGE

A configured Boot Disk, with Ubuntu 20.04 LTS.

Once you've confirmed all your selections are correct, select Create at the bottom. Google Cloud will spend some time processing in the background your new VM instance.

Step 3: Reserving the External Static IP Address

From the Sidebar Menu in the top left, scroll down, hover over "VPC Network" and select "IP Addresses". On this new page, select "Reserve External Static IP Address" from the top.

Set the type to IPv4, and attach it to the VM instance we just created. Select Reserve at the bottom.

Copy this External Static IP Address, and set it as the HOST in the Script (**NOT** the Server).

Step 4: Setting up the Firewall

From the Sidebar Menu in the top left, scroll down, hover over "Network Security" and select "Firewall Policies". On this new page, select "Create Firewall Rule" from the top.

The direction of traffic should be set to Ingress, to allow people into the server.

Action on Match is should be set to Allow.

The target can be set to All instances (as we only have the one instance).

The Source Filter is set to IPv4, and the source range is 0.0.0.0/0, for all IP addresses.

Protocols and Ports should be set to Specified protocols and ports, with TCP checked, and set to the value of PORT in the Server.

Select Create at the bottom.

Action on match ?

☒ Allow

☐ Deny

Targets

All instances in the network

Source filter

IPv4 ranges

Source IPv4 ranges *

0.0.0.0/0

Second source filter

None

Destination filter

None

Protocols and ports ?

☐ Allow all

☒ Specified protocols and ports

☒ TCP

Ports

9999

A configured Firewall Rule.

Step 5: SSH-ing into the VM Instance

From the Sidebar Menu in the top left, scroll down, hover over “Compute Engine” and select “VM Instances”. Click the “SSH” Button for the instance.

From here, we need to upload the server file and the required PBS files.

Essentials Version	Required PBS Files
v16, v17, v18	abilities.txt, moves.txt, items.txt, pokemon.txt, pokemonforms.txt [v17+], tm.txt
v19	abilities.txt, moves.txt, items.txt, pokemon.txt, pokemonforms.txt
v20, v21	abilities.txt, moves.txt, items.txt, pokemon.txt, pokemon_forms.txt

If your server expects the PBS files to be in a sub folder, we need to mkdir one now, and mv the PBS files into that folder.

```
vendily@instance-1:~$ mkdir PBS
vendily@instance-1:~$ cd PBS
vendily@instance-1:~/PBS$ ls
vendily@instance-1:~/PBS$ cd ..
vendily@instance-1:~$ ls
PBS  abilities.txt  cable_club.py  items.txt  moves.txt  pokemon.txt
pokemon_forms.txt
vendily@instance-1:~$ mv *.txt ~/PBS
vendily@instance-1:~$ ls
PBS  cable_club.py
```

From here, we can test the server, by calling `python3 cable_club.py` and connecting to it in game.

Setting the Startup Script

If you close the SSH window, the server stops running. To fix this we must set up startup scripts, so that the server automatically runs when the VM instance starts up.

Step 0: Fixing the PBS_DIR and LOG_DIR on the server

Since the startup scripts run from the root, the PBS_DIR and LOG_DIR must be set relative to the execution folder.

You must put `./home/YOUR_USERNAME_HERE` at the start of the constant, where YOUR_USERNAME_HERE is the green part before the `@` in the SSH window, and adjusted further if there is a subfolder. Using the example above, it'd be `./home/vendily/PBS`.

Reupload the server to the VM, ensuring you rm the old copy first. Close the SSH window once the upload is completed.

Step 1: Adding the Startup Script

Click the name of your VM instance, which will bring you to an information page on the instance. Click Edit at the top.

Scroll to the Management Section. Here you can set the startup script, adjusting further if the server is in a sub folder.

```
#!/bin/bash
nohup python3 /home/YOUR_USERNAME_HERE/cable_club.py &
```

This will run the server in the background automatically. You can also set Automatic Restart to On here as well.

Select Save at the bottom.

Step 2: Restarting the Server

Check the box for your VM instance and select Stop, then once the instance completes the Stop process, select Start.

Once the instance finishes starting, you can connect in game to the server.

Can't Connect After Restarting

Connection issues after rebooting are most likely due to a misconfigured PBS_DIR or startup script path. To check if the server is running, SSH into the VM instance, and run the command `ps aux`. There will be many different processes running, but we are looking in the rightmost column for one that starts with `python3`, which would be the server. If you cannot find it, the server is not running.

To check for an error while starting up, run

```
sudo journalctl -u google-startup-scripts.service > file.txt
```

And download the `file.txt` produced. This file will show the log of the server starting up, though it may have an error. See Common Issues for more details as these will be similar errors to locally hosting the server.

If you forgot to change the HOST or PORT in the script, you can create a `serverinfo.ini` file with the information so you don't need to recompile plugins just to test.

```
HOST = 127.0.0.1  
PORT = 9999
```

Known Issues

All Versions

- Forms considered Invalid
 - If the Pokémon has a hidden form change, such as Rockruff or the Split Evo Regionals, they must be set in `pokemonforms.txt` or `pokemon_forms.txt`, otherwise, the server sees that this is a form that is not defined, and thus invalid. The server does not need to have the same PBS files as the game itself, so you can add these forms to the species without affecting the game.
- Moves Learned by Pre-Evolutions considered Invalid
 - The server considers every move possible to be learned by a species or any of its forms as valid. But if the species is not set to be able to learn moves from the prevo, such as egg moves, then the Pokémon will fail validation. The server does not need to have the same PBS files as the game itself, so you can add these moves to the evolutions without affecting the game.
- Custom Moves\Items\Abilities are Desyncing
 - Don't use `rand` calls directly. You should always be using `pbRandom` in your battle code, which ensures that the random calls go through Cable Club's overwritten `pbRandom` method, which does not use the base `rand` method. You only need to change `rand` calls that mechanically affect the game. Cosmetic ones, like the ones in EBS, or just to play a set of SEs are fine to leave as is, as those `rand` calls do not affect the RNG of online battles.

V16

- Form Specific Moves considered Invalid
 - Since V16 does not have `pokemonforms.txt`, all form numbers are allowed, but the data defined in the scripts would not be known by the server. The server can have its own copy of `pokemonforms.txt`, much like in v17 or v18, and the server would process it as expected for those versions. TMs are defined with `SPECIES_X`, though the `_X` part isn't necessary, as long as at least one form, or the base form, knows it.

V17

- Undefined method '`trainer_type`' in `pbBattleAnimationOverride`
 - This is actually a vanilla bug, not Cable Club. The `trainer_type` should be `trainertype` instead.

V18

- Undefined local variable or method 'b' in pbMessageOnRecall
 - This is actually a vanilla bug, not Cable Club. The b should be battler instead.

Third Party Scripts

- EBDX: Switching out battlers just acts really weird
 - I never really found the cause for it, and I don't want to support the unofficial releases of EBDX. Trades still work though.
- Essentials Deluxe: Variation in Potentially available Battle Mechanics
 - Since the Essentials Deluxe Battle Mechanic plugins have both their priority system and specific requirements for the activation of each, if players can activate the Cable Club from anywhere, certain mechanics such as Dynamax may only be available to one side. This shouldn't cause a desync, as battle mechanics do not verify their conditions before activating in the Attack Phase, but can be unfair if one player has access to a mechanic the other does not. Modifying `def self.do_battle` to set up the available mechanics is best, with Terastal Phenomenon already recharging the Tera Orb before the battle and restoring the old charge afterwards.