

Complete FireRed Upgrade



Never use Pokémon Game Editor (PGE)!

Contents

Understanding C Syntax	5
Defines & Comments	5
enum	5
Types	6
External Declarations	6
Arrays	7
Structures.....	8
Switch Statements	9
Setup	11
Recommended Insertion Steps.....	11
Necessary Modifications.....	12
Pokémon Defines:	12
Item Defines:.....	15
Relevant Additions	15
Optional Byte Changes.....	16
Capitalization/Decapitalization	16
Attack Descriptions	16
Configuration Options.....	16
Configurable Options	17
Var Options	17
Flag Options	20
Start Menu Features	23
Pedometer Flags	23
Battle Frontier Options	24
Character Customization Vars	26
Healing Place Hack	27
TM / HM / Move Tutor Options	27
Times of Day.....	28
Other Number Definitions	28
Badge Obedience	30
OW Palette Ids	30
Pre-Battle Mugshot Options	31
Safari Zone Options.....	31
Randomizer Options	31

Memory Locations	32
Misc Features	32
Misc Battle Effect Options	39
Ability Options.....	40
Damage Calculation Options.....	40
Capturing Pokémon Options.....	40
Exp. Gain Options.....	41
Other Battle Options.....	41
DexNav Options	43
Engine Setup	44
Mega Evolution / Primal Reversion / Ultra Burst.....	44
Z-Moves.....	52
Trainer Sliding Messages.....	53
Multi Battles.....	55
Wild Double Battles	57
Trainer Backsprites.....	58
Battle Terrain	60
Battle Music	61
Poke Balls	62
Trainers With EVs.....	63
Battle Frontier.....	65
Upgraded TM/HM Expansion	69
Reusable TMs	69
Pickup.....	70
Select from PC Hack	70
Time of Day Based Wild Encounters	71
Swarms.....	74
Roaming Pokemon	74
Day & Night System	75
Pre-Battle Mugshots	78
New Field Moves.....	79
Other Features Included	81
Save Expansion.....	81
Updated & New Item Effects	81
Trainer Face Fix	81

DexNav	81
Dynamic Overworld Palettes	81
Ability Pop-Ups.....	81
Hidden Abilities	82
Expanded Text Names.....	82
Pokédex Screen Stats	82
Turbo Boost.....	82
Various Customizable Updates	82
Triple Layer Tiles ¹⁰	83
Expanded Coins.....	84
Multichoice Windows	84
Tile Interaction Scripts	85
Script Specials	86
Creating New Battle Mechanics.....	126
Moves.....	126
Abilities	129
Poke Balls	130
Items	133
Code Files	134
Table Compendium.....	142
Engine Scripts.....	149
Clean.py.....	149
String.py	149
Credits	151

Colour Coding Legend

Encoding	Represents
Bold	Indicates file paths or things to take note of.
<i>Italicized</i>	Used for things such as proper names or scripting command names.
Green	Used to indicate a definition that should be changed in src/config.h . In example scripts it is used for comments and strings.
Dull Orange	Used to indicate another constant defined somewhere (not in the config file). In scripts it is used for the #define and #org directives as well.
<i>Italicized Purple</i>	Used to indicate a table of some sort that can be modified (usually found in

	src/Tables , but not always).
Deep Orange	Used when referring to function names.
Gold	Used for certain definitions like <code>#ifdef</code> .
Blue Underlined	These words are hyperlinks. Clicking on them will take you somewhere (may or may not be in the document).
Red	Used for titles. In scripts they represent references to other scripts.
Faded Blue	Used for showing the inputs for script specials.

Understanding C Syntax

To use this engine, you are not expected to know how to code in [C](#). However, it is wise to read through the following explanations to learn how to modify the various data structures found in the engine.

Defines & Comments

Read [this](#) article for defines. Read [this](#) article for comments. Read [this](#) article for conditional compilation.

Additionally, you may simply type `#define SOMETHING` without any associated value. This is used several times throughout the engine to allow the user to customize various options.

enum

Using an enumerated type is like using the `#define` directive, however it can simplify the process of declaring several constants at once. Unlike `#define`, *enums* do not require you to declare the number of each constant. Each declaration is the value immediately following the previous value. Each declaration within an *enum* should end with a `,`, and each *enum* should begin and end with curly braces `{}` and be followed by a `;`.

Example 1:

```
enum
{
    SUNDAY,
    MONDAY,
    TUESDAY,
};
```

In the example above, `SUNDAY` evaluates to 0 (it is the default starting value), `MONDAY` evaluates to 1, and `TUESDAY` evaluates to 2.

Example 2:

```
enum
{
    MONDAY = 1,
    TUESDAY,
```

```

    THURSDAY = 4,
    FRIDAY,
};

```

In the example above, **MONDAY** evaluates to 1, **TUESDAY** evaluates to 2, **THURSDAY** evaluates to 4, and **FRIDAY** evaluates to 5. Notice that values can be skipped by using the = sign.

Types

In C, each variable or data element has a type. The following are the most widely used types in the engine and should be made familiar:

- **u8**: The data is 1 byte, 0 to 255.
- **s8**: The data is 1 byte, -128 to 127.
- **u16**: The data is 2 bytes, 0 to 65535. (Size of Pokémon species, items, and moves)
- **s16**: The data is 2 bytes, -32768 to 32767
- **u32**: The data is 4 bytes, 0 to 4294967295.
- **s32**: The data is 4 bytes, -2147483648 to 2147483647
- **u8***: The data is a pointer (pointers are 4 bytes) to some bytes. (Used for strings & pics)
- **u16***: The data is a pointer (pointers are 4 bytes) to some list of 2-byte entries.
- **u32***: The data is a pointer (pointers are 4 bytes) to some list of 4-byte entries.
- **const**: Placed before any of the above types. This means that the data is unmodifiable after compilation (basically any permanent data in the rom that can't be changed during gameplay).
- **static**: Placed before any of the above types. This means that the variable/data structure/function will be used and is only used in the file it is declared in.

External Declarations

Due to this engine being multi-file, when using data found in other files, it is necessary to declare that elements from other files exist. If declaring in a **.c** (as opposed to **.h**) file, these declarations should be done beneath all of the **#includes** found at the top of the file. If these data structures are not declared before their occurrence in the file, the compiler will throw an *undeclared (first use in this function)* error. There are several cases when you will need to write your own declarations:

- When attempting to use an image compiled by the engine with the file name *MyPicFileName*
 - Using an image's tiles: *extern const u8 [IMAGE_NAME]Tiles[];*
 - Eg. **extern const u8 MyPicFileNameTiles[];**
 - Using an image's palette: *extern const u16 [IMAGE_NAME]Pal[];*
 - Eg. **extern const u16 MyPicFileNamePal[];**
 - Using an image's compressed palette: *extern const u8 [IMAGE_NAME]Pal[];*
 - Eg. **extern const u8 MyPicFileNameCompressedPal[];**
 - Using an image's tilemap: *extern const u8 [IMAGE_NAME]Map[];*
 - Eg. **extern const u8 MyPicFileNameMap[];**
- When attempting to use a string compiled in one of the **.string** files.

- `extern const u8 [STRING_NAME][];`
 - Eg. `extern const u8 gText_MyString[];`
- When attempting to use a script written in one of the `.s` files.
 - `extern const u8 [SCRIPT_NAME][];`
 - Eg. `extern const u8 EventScript_MyScript[];`
 - Don't forget to also type `.global [SCRIPT_NAME]` in the `.s` file!
 - Eg. `.global EventScript_MyScript`

Arrays

Commonly referred to as “tables” or “lists”, arrays contain the bulk of the data editable in the engine. All elements in an array are the same type. Similar to *enum*, each declaration within an array should end with a “,”, each array should be preceded by the type of each element, begin with `[]` (= *enum* doesn't have this), start and end with curly braces `{}`, and be followed by a “;”.

Example 1:

```
u16 sMyArray[] =
{
    5,
    2665,
    11,
};
```

The above array is a standard array of 2-byte entries. In a hex editor it would look like:

05 00 69 0A 0B 00

Example 2:

```
u8 sMyArray[] =
{
    MONDAY,
    TUESDAY,
    SUNDAY,
};
```

In the above example, it is assumed that the constants SUNDAY, MONDAY, and TUESDAY declared in the *enum* first example above are declared prior. This array is an array of 1-byte entries. In a hex editor it would look like:

01 02 00

(Remember, Monday is 1, Tuesday is 2, and Sunday is 0)

Example 3:

```
const u8* sMyArray[] =
{
    [MONDAY] = gText_MyString2,
    [THURSDAY] = gText_MyString3,
    [SUNDAY] = gText_MyString1,
```

```
};
```

In the above example, the same assumption as in *Example 2* holds, as well as the assumption that the strings are declared prior. This array is an array of pointers to strings found in the rom. The day-of-week constants are used here are used to declare array index. So although they seem out of order, the array still compiles as (notice the two empty spots (0) b/c Thursday is 4):

```
[Pointer to gText_MyString1] [Pointer to gText_MyString2] 0 0 [Pointer to gText_MyString3]
```

Structures

Structs are like arrays in the sense that they store data, but unlike arrays, data stored within them does not need to be all the same type. Additionally, each element (referred to as a *member*) can be accessed with its unique name.

Example 1 - Declaring a Struct

```
struct MyStructTemplate
{
    u8 member1;
    u16 member2;
    u16 member3;
};
```

Before *structs* can be used, they must be declared. The above *struct* has been declared to have three members: *member1* is a byte, *member2* is 2-bytes, and *member3* is 2-bytes. Each member declaration ends with a “;”. *Structs* begin and end with {}, and are followed by a “;”.

Example 2 - Using a Struct

```
struct MyStructTemplate sMyStruct =
{
    .member1 = 5,
    .member2 = SPECIES_CHARMANDER,
    .member3 = ITEM_ORAN_BERRY,
};
```

The above struct uses the declaration from *Example 1* (as well as some other constants defined elsewhere). Each member in *MyStruct* is initialized by calling the member name preceded by a “.”, and ending the line with a “,”.

Example 3 - Struct Arrays

```
struct MyStructTemplate sMyStructArray[] =
{
    {
        .member1 = 5,
        .member2 = SPECIES_CHARMANDER,
        .member3 = ITEM_ORAN_BERRY,
    },
};
```



```

{
    .member1 = 5,
    .member2 = SPECIES_SQUIRTLE,
    .member3 = ITEM_ORAN_BERRY,
},
};

```

Structs can be elements of arrays too! No further explanation needed.

Switch Statements

Understanding *switch statements* are not necessary for using this engine, but they can help you understand how to modify predefined *switch statements* on your own.

The general outline of a *switch statement* looks like this:

```

switch (someVariable) {
    case 0:
        Do something;
        break;
    case 1:
        Do something;
        break;
    default:
        Do something;
        break;
}

```

where each “case” determines what happens when “someVariable” contains that value. For example, if someVariable equals 1, then the switch statement will jump to *Case 1*. If the value is not one of the listed cases, then whatever is under *default* will be executed. If there is no *default*, then nothing will happen if the case value is not found. Cases can also be represented by constants. To add a case, simply add it underneath one of the pre-existing cases but above default (**TUESDAY** is intended to be added):

```

switch (Clock->dayOfWeek) { //Clock->dayOfWeek is a variable that holds the day of week
    case SUNDAY:
        Do something on Sunday;
        break;
    case MONDAY:
        Do something on Monday;
        break;
    case TUESDAY:
        Do something on Tuesday;
        break;
    default:
        Do something other days of week;
        break;
}

```

Cases can also be stacked, causing both values to result in the same functionality:

```
switch (Clock->dayOfWeek) {  
    case SUNDAY:  
    case MONDAY:  
        Do something on Sunday & Monday;  
        break;  
}
```

Setup

Recommended Insertion Steps

The CFRU should ideally only ever be applied once to a rom. It's not a good idea to insert it in a rom, add things such as maps and scripts to the generated rom, and then reuse the generated rom as a base for the CFRU again. This tutorial will help circumvent this issue.

1. Add the [Dynamic Pokémon Expansion](#) (and other [suggested hacks](#)) to your rom. This will be referred to as *Rom 1*.
2. Add the CFRU to a [vanilla FireRed rom](#) at the same offset you plan on adding it to your hack. This will be referred to as *Rom 2*.

3. Open the generated file **offsets.ini** and search for *gMoveNames*.

```
gMoveMenuInfoIcons: 08931856
gMoveNames: 08992530
gMovesCanUnfreezeAttacker: 089304B2
```

Record the offset that you find there. This will be referred to as *offset 1*.

4. Assuming your moves list ends with the Z-Moves, search for *Z_Move_1* in **offsets.ini**.

```
ZZ_YELLOWRING: 0891E164
Z_Move_1: 0899479E
Z_Move_10: 08994813
```

Record the offset that you find there. This will be referred to as *offset 2*.

5. Open *Rom 2* in a [hex editor](#) and select all data from *offset 1* up to but not including *offset 2*.

00994780	FF FF FF FF BD D9 E0 D9 D6 E6 D5 E8 D9 FF FF FF	
00994790	FF C2 E3 E0 D8 00 C2 D5 E2 D8 E7 FF FF FF D4 AE	
009947A0	C7 E3 EA D9 00 A2 FF FF FF FF FF D4 AE C7 E3 EA	
Offset: 992530	Block: 992530-99479D	Length: 226E

6. Copy this data and paste it at *offset 1* in *Rom 1*. At *0x148* place a pointer to *offset 1*.

```
00992530 AE FF FF FF FF FF FF FF FF FF FF FF CA E3 E9
00992540 E2 D8 FF FF FF FF FF FF FF FF C5 D5 E6 D5 E8 D9
00992550 00 BD DC E3 E4 FF FF BE E3 E9 D6 E0 D9 E7 E0 D5

00000140 38 40 3D 08 E0 5E 24 08 30 25 99 08 F8 56 45 08
```

7. *Rom 2* has no more use so feel free to delete it.
8. Treat *Rom 1* as your base rom for the CFRU. It is recommended to do all scripting, mapping, and [trainer editing](#) in *Rom 1*. When you're ready to test, run the command `python scripts//make.py` and test on the generated *test.gba* (it'll take only a second to recompile if no changes have been made to the CFRU). This *test.gba* also makes a great location to test scripts to make sure they work properly before finalizing them.

Necessary Modifications

The following modifications must be made before setting up the engine!

Pokémon Defines:

1. Open the files `include/constants/species.h` and `include/constants/pokedex.h`.
2. Modify the Pokémon indices found in this file to match the ones in your hack.
3. If you have not added in any new Pokémon to Fire Red, you can leave the unused species indices as their default values.
4. If you have added new Pokémon, ideally, you should not delete any Pokémon names and just assign all species you aren't using a unique number starting after your last Pokémon slot. However, if this is too tedious for you, you'll need to do the following two things.
5. First, modify the following tables by removing entries you don't wish to use:

Table	File
<i>sSpecialZMoveTable</i>	<code>src/set_z_effect.c</code>
<i>sSmartWildAITable</i>	<code>src/Battle_AI/AI_master.c</code>
<i>gWildSpeciesBasedBattleBGM</i>	<code>src/Tables/Music_Tables.c</code>
<i>gMiniorCores</i>	<code>src/form_change.c</code>
<i>sBannedBackupSpecies</i>	<code>src/form_change.c</code>
<i>sTypeToArceusForm</i>	<code>src/form_change.c</code>
<i>sTypeToSilvallyForm</i>	<code>src/form_change.c</code>
All Tables	<code>src/Tables/Pokemon_Tables.c</code>

6. Secondly, the following Pokémon are necessary and must be changed if you are not using them (preferably all to unique numbers - I can't guarantee they'll all work as the same number). You can also try deleting their defines in `include/constants/species.h`. This should (but not always) work:

Species	Reason
SPECIES_NONE 0x0	It's 0. Don't change or remove.
SPECIES_BULBASUR 0x1	Referred to by vanilla FR roaming code.
SPECIES_CHARMANDER 0x4	Referred to by vanilla FR roaming code.
SPECIES_NIDORAN_M 0x20	Special breeding.
SPECIES_FARFETCHD 0x53	Signature item Stick .
SPECIES_CUBONE 0x68	Signature item Thick Club .
SPECIES_MAROWAK 0x69	Signature item Thick Club .
SPECIES_MAROWAK_A 0x40F	Signature item Thick Club .
SPECIES_CHANSEY 0x71	Signature item Lucky Punch . Luck Incense breeding.
SPECIES_MR_MIME 0x7A	Odd Incense breeding.
SPECIES_DITTO 0x84	Signature items Quick Powder & Metal Powder . Daycare.

SPECIES_SNORLAX 0x8F	Full Incense breeding.
SPECIES_PICHU 0xAC	Volt Tackle breeding.
SPECIES_MARILL 0xB7	Sea Incense breeding.
SPECIES_SUDOWOODO 0xB9	Rock Incense breeding.
SPECIES_WOBBUFFET 0xCA	Lax Incense breeding.
SPECIES_UNOWN 0xC9	Has many forms. Change those too.
SPECIES_MANTINE 0xE2	Wave Incense breeding.
SPECIES_RAIKOU 0xF3	Referred to by vanilla FR roaming code.
SPECIES_ENTEI 0xF4	Referred to by vanilla FR roaming code.
SPECIES_SUICUNE 0xF5	Referred to by vanilla FR roaming code.
SPECIES_AZURILL 0x15E	Sea Incense breeding.
SPECIES_WYNAUT 0x168	Lax Incense breeding.
SPECIES_ROSELIA 0x16B	Rose Incense breeding.
SPECIES_CLAMPERL 0x175	Signature items Deep Sea Tooth & Deep Sea Scale .
SPECIES_CASTFORM 0x181	Signature ability: Forecast .
SPECIES_VOLBEAT 0x182	Special breeding.
SPECIES_LATIAS 0x197	Signature item Soul Dew .
SPECIES_LATIOS 0x198	Signature item Soul Dew .
SPECIES_CHIMECHO 0x19B	Pure Incense breeding.
SPECIES_EGG 0x19C	It's an egg. Don't change or remove.
SPECIES_BUDEW 0x1CB	Rose Incense breeding.
SPECIES_BURMY 0x1D1	Changes form after battle.
SPECIES_BURMY_SANDY 0x2C3	Changes form after battle.
SPECIES_BURMY_TRASH 0x2C4	Changes form after battle.
SPECIES_CHERRIM 0x1DA	Signature ability: Flower Gift .
SPECIES_CHERRIM_SUN 0x2EF	Signature ability: Flower Gift .
SPECIES_CHINGLING 0x1E6	Pure Incense breeding.
SPECIES_BONSLY 0x1EB	Rock Incense breeding.
SPECIES_MIME_JR 0x1EC	Odd Incense breeding.
SPECIES_HAPPINY 0x1ED	Luck Incense breeding.
SPECIES_MUNCHLAX 0x1F3	Full Incense breeding.
SPECIES_MANTYKE 0x1FF	Wave Incense breeding.
SPECIES_ROTOM 0x214	Special breeding.
SPECIES_DIALGA 0x218	Signature item Adamant Orb .
SPECIES_PALKIA 0x219	Signature item Lustrous Orb .
SPECIES_GIRATINA 0x21C	Signature item Griseous Orb .
SPECIES_GIRATINA_ORIGIN 0x2CE	Signature item Griseous Orb .
SPECIES_PHIONE 0x21E	Special breeding from Manaphy .
SPECIES_DARKRAI 0x220	Only species that can use Dark Void .
SPECIES_SHAYMIN 0x221	Changes form when frozen.
SPECIES_SHAYMIN_SKY 0x2CF	Changes form when frozen.
SPECIES_DARMANITAN 0x260	Signature ability: Zen Mode .

SPECIES_DARMANITANZEN 0x2E1	Signature ability: Zen Mode .
SPECIES_KELDEO 0x2BC	Changes form with Secret Sword .
SPECIES_KELDEO_RESOLUTE 0x2F5	Changes form if doesn't know Secret Sword .
SPECIES_MELOETTA 0x2BD	Changes form with Relic Song .
SPECIES_MELOETTA_PIROUETTE 0x2EA	Changes form with Relic Song .
SPECIES_GRENINJA 0x2FE	Signature ability: Battle Bond .
SPECIES_ASHGRENINJA 0x347	Signature ability: Battle Bond .
SPECIES_FURFROU 0x310	Special breeding.
SPECIES_AEGISLASH 0x315	Signature ability: Stance Change .
SPECIES_AEGISLASH_BLADE 0x341	Signature ability: Stance Change .
SPECIES_ZYGARDE 0x33A	Signature ability: Power Construct .
SPECIES_ZYGARDE_10 0x345	Signature ability: Power Construct .
SPECIES_ZYGARDE_COMPLETE 0x346	Signature ability: Power Construct .
SPECIES_HOOPA 0x33C	Special message why trying to use Hyperspace Fury .
SPECIES_HOOPA_UNBOUND 0x33D	Only Pokémon that can use Hyperspace Fury .
SPECIES_GROUDON_PRIMAL 0x38D	Is referenced as a <i>Red Primal</i> .
SPECIES_KYOGRE_PRIMAL 0x38E	Is referenced as a <i>Blue Primal</i> .
SPECIES_LYCANROC 0x3C2	Special battle animation.
SPECIES_LYCANROC_N 0x416	Special battle animation.
SPECIES_LYCANROC_DUSK 0x43A	Special battle animation.
SPECIES_WISHIWASHI 0x3C3	Signature ability: Schooling .
SPECIES_WISHIWASHI_S 0x417	Signature ability: Schooling .
SPECIES_MINIOR_SHIELD 0x3DF	Signature ability: Shields Down .
SPECIES_MINIOR_RED 0x429	Signature ability: Shields Down .
SPECIES_MINIOR_BLUE 0x42A	Signature ability: Shields Down .
SPECIES_MINIOR_ORANGE 0x42B	Signature ability: Shields Down .
SPECIES_MINIOR_YELLOW 0x42C	Signature ability: Shields Down .
SPECIES_MINIOR_INDIGO 0x42D	Signature ability: Shields Down .
SPECIES_MINIOR_GREEN 0x42E	Signature ability: Shields Down .
SPECIES_MINIOR_VIOLET 0x42F	Signature ability: Shields Down .
SPECIES_MIMIKYU 0x3E3	Signature ability: Disguise .
SPECIES_MIMIKYU_BUSTED 0x430	Signature ability: Disguise .

Also be sure to change the species indices in **asm_defines.s** as well (just copy and paste the data from before but use a text editor to do a search and replace of all **#define** with **.equ** and all " 0x" with ", 0x" (without quotation marks and notice the whitespace).

If any Pokémon is missing from the above table please submit an issue!

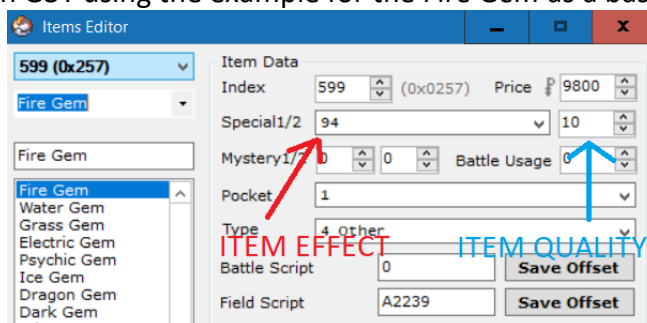
Item Defines:

Part 1

1. Open the file **include/constants/items.h**.
2. Modify the item indices found in this file to match the ones in your hack.
3. **DO NOT DELETE ANY ITEM NAMES**. If there is an item you are not using, then set its index to some random really high number (for example, **0xFFFFE**) (avoid **0xFEFE** and **0xFFFF**). Make sure to also remove these items from both **gFlingTable** and **gItemsByType** found in **src/Tables/Item_Tables**.

Part 2

1. Open the file **include/constants/hold_effects.h**.
2. Scroll down to where it says **"//NEW ITEM EFFECTS"**. This is the list of new hold item effects. When relevant, the item quality is included.
3. Set up your items in G3T using the example for the *Fire Gem* as a base:



As you can see, the item effect is set to 94 and the item quality is set to 10 (the move type for *Fire*).

Relevant Additions

The following are a list of relevant and recommended add-ons created by other users of the [PokeCommunity](#) that for some reason or another, were chosen to be left out from this engine. If you choose to apply any of these, they should be applied prior to applying the engine.

- [Dynamic Pokémon Expansion](#) (**HIGHLY RECOMMENDED**)
- [Overworld Form Change](#) (Only Arceus, Silvally, Genesect, & Giratina are included in the CFRU)
- [DS-Style Pokémon Selection Screen](#)
- [Black & White Pokémon Summary Screen](#) (**HIGHLY RECOMMENDED**)
- [Sideways Stairs](#) (**HIGHLY RECOMMENDED**)
- [MrDollSteak's Decap. and Attack Rombase](#) (For the decapitalization)
- [Nature Coloured Stats](#) (see *BW Summary Screen* comments for alternate implementation)
- [IV Rankings](#) (see *BW Summary Screen* comments for alternate implementation)
- [Unown Report](#) (**NOT COMPATIBLE**)
- [HGSS Kanto Reloaded Music Patch](#)
- [HGSS Johto Music Patch](#)
- [BW/BW2 Music Patch](#)
- [All-Instrument Patch](#) (Make sure to change insert offset of engine to avoid overwriting)
- [Squeetz' Music Rom Base](#)

Optional Byte Changes

Open the file **bytereplacement** in a text editor and search for **##Optional Byte Changes##**. Place a **#** before anything here (until **##Remove Things##**) to remove it from the engine.

Capitalization/Decapitalization

This engine does not decapitalize vanilla strings. However, there are certain strings that have been decapitalized/capitalized and can be changed. The following files have such strings which can be modified in a text editor:

- **strings/ability_name_table.string**
- **strings/attack_name_table.string**
- **strings/frontier_trainer_names.string**
- **strings/overworld_strings.string** (specifically search for *Bag Strings*)
- **strings/party_menu.string** (specifically *gMenuText_Move*)
- **strings/poketools.string**
- **strings/type_names.string**
- **strings/z_move_names.string**

Attack Descriptions

Some of the newer attack descriptions were written to span five lines. This is not supported by the original FireRed summary screen, so it is highly recommended to update to the [Black & White Pokémon Summary Screen](#) (linked to above). If you do not wish to do that, find any attack descriptions which span five lines and shorten them manually.

Configuration Options

See below.

Configurable Options

There are many configurable options in the file **src/config.h**. These options are meant to provide the user with as much versatility as possible. Below is a more detailed description of each option than show in the configuration file:

Var Options

<i>Flag Definition</i>	<i>Description</i>
VAR_TERRAIN	<p>Setting this var to one of the following values before initiating a battle will load the battlefield with the corresponding terrain:</p> <ol style="list-style-type: none">1: Electric Terrain2: Grassy Terrain3: Misty Terrain4: Psychic Terrain
VAR_TOTEM	<p>This represents are series of vars using for initiating battles with Totem Pokémon. There are four vars in total, each representing a specific Pokémon slot on the field. Adding the following values to the var will indicate which slot that var is for:</p> <p>0: <i>Player Pokémon in Singles, Left Player Pokémon in Doubles</i></p> <p>1: <i>Enemy Pokémon in Singles, Right Enemy Pokémon in Doubles</i></p> <p>2: <i>Right Player Pokémon in Doubles</i></p> <p>3: <i>Left Enemy Pokémon in Doubles</i></p> <p>The vars must be set to the addition of two values. Choose one from each of the following sets:</p> <p>Stats:</p> <ol style="list-style-type: none">1: <i>Attack</i>2: <i>Defense</i>3: <i>Speed</i>4: <i>Special Attack</i>5: <i>Special Defense</i>6: <i>Accuracy</i>7: <i>Evasion</i> <p>Amount:</p> <p>0x10: <i>Increase Stat by 1</i></p> <p>0x20: <i>Increase Stat by 2</i></p> <p>0x30: <i>Increase Stat by 3</i></p> <p>0x40: <i>Increase Stat by 4</i></p>

	<p>0x50: Increase Stat by 5 0x60: Increase Stat by 6 0x90: Decrease Stat by 1 0xA0: Decrease Stat by 2 0xB0: Decrease Stat by 3 0xC0: Decrease Stat by 4 0xD0: Decrease Stat by 5 0xE0: Decrease Stat by 6</p> <p>So, for instance, in a single battle, having the enemy Pokémon start the battle with its <i>Attack</i> raised by 2, you would set the var <code>VAR_TOTEM + 1</code> to the value of 0x21 (0x1 + 0x20).</p> <p>Additionally, this feature can also be used in trainer battles. If used here, any Pokémon sent out into the set position will have their stats raised. So, using the example from above, all the opposing trainer's Pokémon in <i>Singles</i> would start with their <i>Attack</i> raised by 2.</p>
VAR_BACKSPRITE_SWITCH	<p>Setting this var to a value other than 0 will change the default back sprite loaded for the player in battle.</p> <p>See <code>src/Tables/back_pic_tables.c</code> for a list of available backsprites.</p>
VAR_BATTLE_BG	<p>If uncommented, setting this var to a value other than 0 will cause the regular battle background loaded to be replaced by a custom one.</p> <p>See <code>/include/battle.h</code> for a list of options.</p> <p>Search for <code>BATTLE_TERRAIN_GRASS</code> in the file to see them.</p>
VAR_SWARM_INDEX	<p>A var that is automatically set by the engine. It contains the index in the swarming table of the species that is currently swarming. Swarms are set to change on a daily basis. If <code>TIME_ENABLED</code> is commented out, then the code will need to be modified to find an alternative method to enable swarms. It can be found in <code>src/wild_encounter.c</code>.</p> <p>The swarming table can be edited by searching for gSwarmTable in <code>src/Tables/wild_encounter_tables.c</code>.</p>
VAR_SWARM_DAILY_EVENT	<p>A pair of two vars (this one and the one immediately following it) that are automatically set by the engine. They are used to determine if a swarm has already been chosen for the given date. If <code>TIME_ENABLED</code> is commented out, then these vars will be set once and never again.</p>
VAR_DEFAULT_WALKING_SCRIPT	<p>If uncommented, then a var that contains the index of the script in walking script table that is set to run on every step. Set this var to 0 if you do not want any custom scripts to be</p>

	run each step. To add a walking script, search for <i>sDefaultWalkingScripts</i> in src/overworld.c and either add (<i>const u8*</i>) pointers to scripts preloaded in the ROM, or define your own script (like <i>EventScript_WalkingScript1</i>) in one of the assembly files and declare it at the top like: <i>extern const u8 EventScript_WalkingScript1[];</i>
VAR_DEXNAV	A var that holds the species to search for in the Overworld via the <i>DexNav</i> feature. Press <i>Select</i> in the DexNav GUI to save.
VAR_STATUS_INDUCER	If uncommented, then a var that if set, causes wild and Trainer Pokémon to be generated with the set status condition. It is split into an upper and lower byte. If the upper byte is set to 0, then this status condition will be given to Pokémon until this var is cleared. Any value in the upper byte other than 0 acts as a timer and is subtracted by 1 for each battle completed. For example, setting this var to <i>0x0640</i> will paralyze all enemy Pokémon in battles for 6 battles. See here for a list of status bytes.
VAR_SECOND_OPPONENT	A var that can be set by the engine (with <i>trainerbattle 0xA</i> or <i>trainerbattle 0xB</i>) to represent the trainer id of the second trainer in battles against two opponents. If set manually in conjunction with <i>FLAG_TWO_OPPONENT</i> , a battle against two opponents will be started the next time a trainer battle is initiated.
VAR_PARTNER	A var that can be set by the engine (with <i>trainerbattle 0xA</i> or <i>trainerbattle 0xC</i>) to represent the trainer id of the partner trainer in multi battles. If set manually in conjunction with <i>FLAG_TAG_BATTLE</i> , a battle with a partner against a single trainer will be started the next time a trainer battle is initiated. If setting manually, take care to set <i>VAR_PARTNER_BACKSPRITE</i> as well.
VAR_PARTNER_BACKSPRITE	A var that can be set by the engine (with <i>trainerbattle 0xA</i> or <i>trainerbattle 0xC</i>) to represent the backsprite id of the partner trainer in multi battles. If setting manually, take care to also set <i>VAR_PARTNER</i> and <i>FLAG_TAG_BATTLE</i> .

Flag Options

NOTE: Many of the following flags are cleared at the end of battle. To remove this, open the file `src/end_battle.c` and remove the flag from [gEndBattleFlagClearTable](#). Many of the following flags can also be disabled by commenting out the lines where they are defined.

Flag Definition	Description
FLAG_INVERSE	Setting this flag will enable Inverse Battles . This flag is automatically cleared at the end of each battle. Comment out to remove <i>Inverse Battles</i> .
FLAG_SKY_BATTLE	Setting this flag will indicate to the engine that a Sky Battle is in progress. This flag is automatically cleared at the end of each battle. Comment out to remove <i>Sky Battles</i> .
FLAG_NO_CATCHING	Setting this flag will cause enemy Pokémon to always dodge balls thrown at them. This flag is automatically cleared at the end of each battle. Comment out to not use this flag.
FLAG_NO_RUNNING	Setting this flag prevents the player from running away during wild battles. This flag is automatically cleared at the end of each battle. Comment out to not use this flag.
FLAG_NO_CATCHING_AND_RUNNING	This flag acts as a combination of the above two flags. This flag is automatically cleared at the end of each battle. Comment out to not use this flag.
FLAG_CATCH_TRAINERS_POKÉMON	Setting this flag allows the player to capture Pokémon belonging to the opposing trainer. Capturing a Pokémon in this way will automatically end the battle. This flag is automatically cleared at the end of each battle. Comment out to remove this feature.
FLAG_EXP_SHARE	Setting this flag activates the Gen 6+ Exp. Share . Comment out to use the old Exp. Share.
FLAG_DOUBLE_BATTLE	Setting this flag will cause battles against trainers to be Double Battles , if possible. Comment out to not use this flag.
FLAG_TAG_BATTLE	This flag is set by the engine when the scripting command <i>trainerbattle 0xA</i> or <i>trainerbattle 0xC</i> is used in a script to activate a tag battle. If setting this flag manually, take care to also set VAR_PARTNER and VAR_PARTNER_BACKSPRITE . This flag is automatically cleared at the end of each battle.

FLAG_TWO_OPPONENT	<p>This flag is set by the engine when the scripting command <i>trainerbattle 0xA</i> or <i>trainerbattle 0xB</i> is used in a script to activate a battle against two opponents. If setting this flag manually, take care to also set VAR_SECOND_OPPONENT.</p> <p>This flag is automatically cleared at the end of each battle.</p>
FLAG_ACTIVATE_TUTORIAL	<p>If TUTORIAL_BATTLES is defined, Setting this flag activates Professor Oak's tutorial during the next trainer battle.</p> <p>This flag is automatically cleared at the end of each battle. Comment out to not use this flag.</p>
FLAG_WILD_CUSTOM_MOVES	<p>Setting the flag before a wild battle starts will create the wild Pokémon with the moves given in the input vars. This works with both regular wild battles and scripted wild battles (if scripted, set the input vars before using the <i>wildbattle</i> scripting command). Setting any value to 0xFFFF will cause the default move to be loaded in that slot. Setting any value to 0x0 will load a blank move in that slot. Note that there are additional inputs for wild double battles. Comment out to not use this flag. The input is as follows:</p> <p>Var 0x8000: Move 1 - Pokémon 1 Var 0x8001: Move 2 - Pokémon 1 Var 0x8002: Move 3 - Pokémon 1 Var 0x8003: Move 4 - Pokémon 1 Var 0x8004: Move 1 - Pokémon 2 (Wild Double) Var 0x8005: Move 2 - Pokémon 2 (Wild Double) Var 0x8006: Move 3 - Pokémon 2 (Wild Double) Var 0x8007: Move 4 - Pokémon 2 (Wild Double)</p>
FLAG_SMART_WILD	<p>Setting this flag allows wild Pokémon to use the basic AI checks used in trainer battles.</p> <p>This flag is automatically cleared at the end of each battle. Comment out to not use this flag.</p>
FLAG_SCALE_WILD_POKEMON_LEVELS	<p>Setting this flag will cause all random wild Pokémon encounters (this does not include scripted encounters) to have Pokémon with levels that match the lowest level in your party.</p> <p>Comment out to not use this flag.</p>
FLAG_SCALE_TRAINER_LEVELS	<p>Setting this flag causes all Trainer Pokémon to have levels that match the highest level in your party.</p> <p>Comment out to not use this flag.</p>

FLAG_HIDDEN_ABILITY	Setting this flag before a Wild battle causes Wild Pokémon to be generated with their hidden abilities . It also lets the <i>givepokemon</i> scripting command give Pokémon with their hidden abilities. This flag is automatically cleared at the end of each battle.
FLAG_DOUBLE_WILD_BATTLE	Setting this flag causes all wild battles to be against two wild Pokémon in a <i>Double battle</i> format (if the player has at least two viable Pokémon on their team). This flag is automatically cleared at the end of each battle. Comment out to not use <i>Wild Doubles</i> .
FLAG_NO_RANDOM_WILD_ENCOUNTERS	Setting this flag will stop Pokémon from appearing while walking through grass or caves, or while surfing on water. Pokémon can still appear if the player chooses to fish, smash rocks, or use <i>Sweet Scent</i> . Comment out to not use this flag.
FLAG_REMOVE_EVO_ITEM	A flag set by the engine to help with certain item-based evolutions.
FLAG_SHINY_CREATION	If set, the Pokémon generated by the game will be shiny. This includes wild Pokémon, gift Pokémon, or all the Pokémon in the next trainer battle. This flag is automatically cleared at the end of each battle. Comment out to not use this flag.
FLAG_AUTO_RUN	Setting this flag enables auto-run. For convenience, this flag is toggled in the overworld by pressing the <i>L-Button</i> (as long as the button setting is not set to <i>L=A</i>). By default, this flag is defined in the config to be the same as the flag that allowed running in vanilla FR.
FLAG_RUNNING_ENABLED	If this line is uncommented, this flag can be used to control when the player can run. If this flag is not set, the player will be forced to walk.
FLAG_DISABLE_BAG	Setting this flag prevents the bag from being utilized in-battle.
FLAG_MOVE_RELEARNER_IGNORE_LEVEL	Setting this flag allows the Move Relearner to show a full list of moves Pokémon can learn through level-up (not restricted by the Pokémon's level). Comment out to not use this flag.
FLAG_EGG_MOVE_RELEARNER	Setting this flag allows the <i>Move Reminder</i> to teach Egg Moves instead. Comment out to not use this flag.

Start Menu Features

Any of the following flags can be commented out to remove them from the engine.

Definition	Description
FLAG_SYS_BAG_HIDE	This flag allows the hacker to toggle the <i>Bag</i> in the start menu, for events where the player isn't allowed to use items or lost their bag for various reasons. If this is commented out, <i>BAG</i> will always be present on the start menu. (Set to Hide)
FLAG_SYS_PLAYER_HIDE	This allows the hacker to toggle on/off the <i>Trainer Card</i> from the start menu. Commenting this out will cause <i>PLAYER</i> to always be present. (Set to Hide)
FLAG_SYS_SAVE_HIDE	This allows the hacker to toggle the <i>Save Game</i> feature from the start menu. Commenting this out causes <i>SAVE</i> to be permanent on the start menu. (Set to Hide)
FLAG_SYS_DEXNAV	This allows the hacker to toggle <i>TOOLS</i> from the start menu. If this flag is defined and not set, <i>POKéDEX</i> will show up on the start menu (if the Pokédex flag is set of course). When the flag is set, <i>TOOLS</i> will replace <i>POKéDEX</i> , which yields a separate menu including both <i>POKéDEX</i> and <i>DEXNAV</i> . If this is commented out, The DexNav feature will be inaccessible. Do not give the player the DexNav before giving them the Pokédex!
FLAG_POKETOOLS_MENU	This flag causes <i>TOOLS</i> to open a separate start menu as opposed to the multichoice list generated by default. This flag is purely for aesthetic purposes, although an advanced hacker could use this to create two separate start menus. If this is commented out, the default <i>TOOLS</i> multichoice will load (see above).

Pedometer Flags

Setting any of these flags will initiate a pedometer of the corresponding size. The pedometer value can be read using *special 0x8A*.

Definition	Description
FLAG_LONG_PEDOMETER	4 byte pedometer (max value 0xFFFFFFFF or 4 294 967 295)
FLAG_MED_PEDOMETER	2 byte pedometer (max value 0xFFFF or 65 535)
FLAG_SMALL_PEDOMETER_1	1 byte pedometer (max value 0xFF or 255)
FLAG_SMALL_PEDOMETER_2	1 byte pedometer (max value 0xFF or 255)

Battle Frontier Options





These flags and vars should be used in any battle facility (not just Battle Tower).

Definition	Description
FLAG_BATTLE_FACILITY	Setting this flag indicates to the engine that the Player is in a Battle Facility . This means that: <ul style="list-style-type: none">Trainer Pokémon will be generated within the restrictions of the tier set in the var VAR_BATTLE_FACILITY_TIER. The amount of trainer Pokémon generated will match the number set in the var VAR_BATTLE_FACILITY_POKE_NUM.Trainer Pokémon will have Pokémon generated with the level contained in the var VAR_BATTLE_FACILITY_POKE_LEVEL.The battle format will be loaded from the var defined in VAR_BATTLE_FACILITY_BATTLE_TYPE.The music in-battle will be played based on the value contained within the var VAR_BATTLE_FACILITY_SONG_OVERRIDE.The bag will be inaccessible in battle.
VAR_BATTLE_FACILITY_POKE_NUM	Setting this var to a value between 1 & 6 dictates the quantity of Pokémon the player and trainers can use in the Battle Facility. Setting it to 0 will default in 1. Setting it to a number greater than 6 will default in 6.
VAR_BATTLE_FACILITY_POKE_LEVEL	Setting this var to a value between 1 and what is defined in MAX_LEVEL will set all Pokémon in the Battle Facility to that level. Setting it to 0 will default in 1. Setting to a number greater than MAX_LEVEL will default in MAX_LEVEL .
VAR_BATTLE_FACILITY_BATTLE_TYPE	Setting this var to one of the below values set the battle format in the Battle Tower: 0: Single Battle 1: Double Battle 2: Multi Battle 3: Link Multi Battle 4: Random Single Battle 5: Random Double Battle 6: Random Multi Battle

	Random battles assign the player a random team. Any other value defaults in Single Battle.
VAR_BATTLE_FACILITY_TIER	<p>Setting this var to one of the below values indicates to the engine which ruleset should be following for battles in the Battle Tower:</p> <p>0: Regular Battle Tower Rules 1: No Restrictions 2: Smogon OU 3: Smogon Uber 4: Smogon Little Cup 5: Skeli's Middle Cup / GS Cup 6: Smogon Monotype 7: Smogon Camomons 8: Uber Camomons 9: Little Cup Camomons 10: Middle Cup Camomons / GS Cup Camomons 11: Smogon Scalemons 12: Smogon 350 Cup 13: Smogon Averagemons 14: Smogon Benjamin Butterfree 15: Battle Mine Format 1 (OU, Camomons, Benjamin Butterfree) 16: Battle Mine Format 2 (Scalemons, 350 Cup, Averagemons) 17: Battle Mine Format 3 (Little Cup, Little Cup Camomons)</p>
VAR_BATTLE_FACILITY_TRAINER1_NAME	This var is automatically set by the engine to hold the index of the random name for the first Battle Tower trainer. It is set to 0xFFFF after every battle. Do not set it manually.
VAR_BATTLE_FACILITY_TRAINER2_NAME	This var is automatically set by the engine to hold the index of the random name for the second Battle Tower trainer in Multi Battles. It is set to 0xFFFF after every battle. Do not set it manually.
VAR_BATTLE_FACILITY_SONG_OVERRIDE	Setting this var to a song Id will cause that song to be played in <i>Battle Tower</i> battles and <i>Link Battles</i> .
VAR_FACILITY_TRAINER_ID	This var is set by <i>special 0x52</i> to indicate which trainer class and details is being spawned as the first opponent. The frontier trainer details should be added to the <i>gTowerTrainers</i> table which can be found in src/Tables/battle_frontier_trainers.c .
VAR_FACILITY_TRAINER_ID_2	This var is set by <i>special 0x52</i> to indicate which trainer class and details is being spawned as the second opponent in multi battles. The frontier

	trainer details should be added to the <i>gTowerTrainers</i> table which can be found in src/Tables/ battle_frontier_trainers.c
VAR_FACILITY_TRAINER_ID_PARTNER	This var is set by <i>special 0x52</i> to indicate which trainer class and details is being spawned as the player's partner in multi battles if the partner is chosen to be randomized. The trainer details should be added to the <i>gFrontierMultiBattleTrainers</i> table found in src/Tables/Frontier_Trainers.c .

Character Customization Vars

Definition	Description
VAR_PLAYER_WALKRUN	Set this var to change the player's walking/running overworld sprite frames on map reload. The upper byte is used as the table Id, For example, setting to 0x0200 will load the walking/running frames from table 2, sprite 0.
VAR_PLAYER_BIKING	Switch player biking frames (same rules as above).
VAR_PLAYER_SURFING	Switch player surfing frames (same rules as above).
VAR_PLAYER_VS_SEEKER	Switch player VS Seeker frames (same rules as above).
VAR_PLAYER_FISHING	Switch player Fishing frames (same rules as above).
VAR_PLAYER_VS_SEEKER_ON_BIKE	Switch player Biking/Vs Seeker frames (same rules as above).
VAR_TRAINERCARD_MALE	Set this var to the trainer sprite id of the male player front sprite that appears on the trainer card.
VAR_TRAINERCARD_FEMALE	Set this var to the trainer sprite id of the female player front sprite that appears on the trainer card.
VAR_RUNTIME_CHANGEABLE	<p>If a person event has a given overworld table id 0xFF, it can be changed at runtime by changing these variables to a sprite number. There are 15 variables used in total.</p> <p>For example, setting VAR_RUNTIME_CHANGEABLE+2 to 16, will cause all NPCs with ids 0xFF02 to appear with the little boy overworld sprite (in vanilla FR).</p> <p>Person event no: </p> <p>Picture no: </p> <p>Unknown:  </p> <pre>#define VAR_RUNTIME_CHANGEABLE 0x4080 #org 0x800000 setvar VAR_RUNTIME_CHANGEABLE+2 16 "Var 0x4082"</pre>

Healing Place Hack

The following vars relate to JPAN's healing place hack. If `SET_HEALING_PLACE_HACK` is not defined, ignore these vars.

VAR_HEALINGMAP	Set this var to the map and bank for the player to respawn to after whiting out. For example, if it is set to 0x0104, the player will respawn in their room (vanilla FR) (map bank 4, map 1).
VAR_HEALING_XPOS	Set this var to the x-position the player will respawn at on the map in <code>VAR_HEALINGMAP</code> .
VAR_HEALING_YPOS	Set this var to the y-position the player will respawn at on the map in <code>VAR_HEALINGMAP</code> .

TM / HM / Move Tutor Options

Definition	Description
EXPANDED_TMSHMS	Allows TMs and HMs past the original 58 to be used up to 128 total TMs + HMs.
EXPANDED_MOVE_TUTORS	Allows Move Tutors past the original 16 up to 128. This also removes the restriction that only allows the powerful elemental moves to be learned by the Kantonian starters.
NUM_TMS	Set this to the total number of TMs in the game.
NUM_HMS	Set this to the total number of HMs in the game.
NUM_MOVE_TUTORS	The number of move tutors. 64 is a good number.
LAST_TOTAL_TUTOR_NUM	<p>Should be equal to $(\text{NUM_MOVE_TUTORS} - 1) + 8$. Must be set to an actual integer or the compilation will not work. This allows special additional move tutors to be added without having to change the size of the compatibility table. These are:</p> <ol style="list-style-type: none">1. Draco Meteor2. Secret Sword3. Relic Song4. Volt Tackle5. Dragon Ascent6. Thousand Arrows7. Thousand Waves8. Core Enforcer <p>For more information regarding the move tutor defines, see <code>include/constants/tutors.h</code>.</p>
TMS_BEFORE_HMS	Uncomment this line if you want the HMs to appear after the TMs in your bag.
DELETABLE_HMS	Uncomment this line if you'd like HMs to be deleted

	without the use of the <i>Move Deleter</i> like normal moves.
REUSABLE_TMS	Allows TMs to be reused infinitely without being removed from the bag. Also prevents TMs from being sold, held, or bought more than once. If using this feature, don't forget to assign all TMs a <i>Mystery 1</i> (in G3T) value of 1!

Times of Day

Definition	Description
TIME_MORNING_START	If TIME_ENABLED is defined, set this to the hour (in 24 hr system) that morning starts. This is also the day start time used for many events that only have a daytime/nighttime variant.
TIME_DAY_START	If TIME_ENABLED is defined, set this to the hour (in 24 hr system) that day starts.
TIME_EVENING_START	If TIME_ENABLED is defined, set this to the hour (in 24 hr system) that evening starts.
TIME_NIGHT_START	If TIME_ENABLED is defined, set this to the hour (in 24 hr system) that night starts.

Other Number Definitions

Definition	Description
KANTO_DEX_COUNT	Number of Pokémon in the regional Pokedex.
NATIONAL_DEX_COUNT	Number of Pokémon in the national Pokedex.
MAX_LEVEL	The highest possible level for a Pokémon. If you change this value, make sure to also modify the equivalent value found in asm_defines.s .
NUM_TRAINER_CLASSES	The number of trainer classes. Vanilla FR has 107.
EVOS_PER_MON	If you've changed the number of evolutions per Pokémon, update this number. Vanilla FR has 5.
EV_CAP	The most EVs a Pokémon can accrue for a given stat.
POWER_ITEM_EV_YIELD	The amount of additional EVs gifted in a certain stat by Power Items such as the Power Bracer .
DUSK_BALL_MULTIPLIER	The catch rate (*10) for Dusk Balls . So 30 is 3.0x.
STANDARD_IV	The number of IVs for each stat that standard Trainer's Pokémon are generated with.
SWARM_CHANCE	The chance in percent that a swarm Pokémon will appear on a route if there is currently a swarm in progress on that route.
WILD_DOUBLE_RANDOM_CHANCE	The chance that a wild double will be initiated if the player is walking in grass with a background byte

	with its 4 th bit set. For instance, grass with a background byte of 0x5 will have a chance of starting a wild double battle, and grass with a background byte of 0x25 will have a chance of starting a wild double battle and be covered by the player (water is similarly 0x6 and 0x26).
CREATE_WITH_X_PERFECT_IVS	Set this to the number of 31 IVs Pokémon defined in the table gSetPerfectXlvList should be generated with. For example, if Mewtwo is in the table, and this is defined to 3, any Mewtwo generated (wild or Trainer) will always have at least 3 IVs set to 31. This does not include roaming Pokémon. Table found in src/Tables/pokemon_tables.c .
CREATE_ROAMER_WITH_X_PERFECT_IVS	Set this to the number of 31 IVs roaming Pokémon should be generated with.
EGG_HATCH_LEVEL	Set this to level Pokémon <i>Eggs</i> should hatch at. Before Gen 4 it was 5, from Gen 4 onwards it has been 1.
AI_TRY_TO_KILL_RATE	In battles against a trainer with AI flags of 1 (only check if the move shouldn't be used), the AI will try to use a move to knock out the opponents XX percent of the time, where XX is the number defined here. Setting this to 0 means basic AI will always use random moves in-battle, assuming that move has no reason not to be used.
MB_OMNIDIRECTIONAL_JUMP	Setting a tile's behaviour byte to this number makes the tile act as a 4 way ledge. Meaning, the player will jump over it when approaching it from all 4 directions. For a related option, see CAN_ONLY_USE_OMNIDIRECTIONAL_JUMP_ON_HEIGHT_2 .
MB_ROCK_CLIMB_WALL	Setting a tile's behaviour byte to this number makes it climbable with the move Rock Climb . See New Field Moves for more instructions.
MB_LAVA	Setting a tile's behaviour byte to this number allows the player to surf on any of these tiles if they have a Fire-type Pokémon in their party. Comment out to remove this feature.
MAP_PLAYER_HOME	The map bank and map number of the player's home. This is used to help load the correct message when the Player <i>white's out</i> . The upper byte represents the map bank, while the lower byte represents the map number. So:

	$((4 \ll 8) \mid 0)$ Means <i>map bank 4</i> and <i>map number 0</i> , aka, the Player's home in vanilla FR.
MAX_COINS_DIGITS	Number of digits of game corner coins the player can hold (eg. default is 4 digits = up to 9999 coins).

Badge Obedience

Definition	Description
BASE_OBEDIENCE_LEVEL	The highest level that a traded Pokémon will obey the player at if they have no badges.
BADGE_1_OBEDIENCE_LEVEL	The highest level that a traded Pokémon will obey the player at if they have 1 badge.
BADGE_2_OBEDIENCE_LEVEL	The highest level that a traded Pokémon will obey the player at if they have 2 badges.
BADGE_3_OBEDIENCE_LEVEL	The highest level that a traded Pokémon will obey the player at if they have 3 badges.
BADGE_4_OBEDIENCE_LEVEL	The highest level that a traded Pokémon will obey the player at if they have 4 badges.
BADGE_5_OBEDIENCE_LEVEL	The highest level that a traded Pokémon will obey the player at if they have 5 badges.
BADGE_6_OBEDIENCE_LEVEL	The highest level that a traded Pokémon will obey the player at if they have 6 badges.
BADGE_7_OBEDIENCE_LEVEL	The highest level that a traded Pokémon will obey the player at if they have 7 badges.

OW Palette Ids

These represent the OW palette Ids of the ninja overworld disguises. If you have changed the default OW palettes, you may want to change these values as well.

Definition	Description
TREE_DISGUISE_PAL_ID	The OW palette Id the ninja tree disguise uses (movement type <i>0x39</i> in AdvanceMap).
ROCK_DISGUISE_PAL_ID	The OW palette Id the ninja mountain disguise uses (movement type <i>0x3A</i> in AdvanceMap).
WEIRD_DISGUISE_PAL_ID	The OW palette Id the ninja weird disguise uses (I have no idea what the movement type is).

Pre-Battle Mugshot Options

Different vars for the [Pre-Battle Mugshots](#).

Definition	Description
FR_PRE_BATTLE_MUGSHOT_STYLE	Uses the default preset mugshots for the FireRed Elite 4 and Champion
VAR_PRE_BATTLE_MUGSHOT_STYLE	Change mugshot tilemap style.
VAR_PRE_BATTLE_MUGSHOT_SPRITE	Can be used to change the player's mugshot image to a VS. Symbol.
VAR_MUGSHOT_PLAYER_PAL	Change player's tilemap palette for Two Bars.
FLAG_LOAD_MUGSHOT_SPRITE_FROM_TABLE	If the opponent's or player's sprite has a replacement mugshot sprite in the table <i>sPreBattleMugshotSprites</i> , then the sprite will be loaded from there. Comment out to not use this flag. Table found in src/Tables/mugshot_tables.c

Safari Zone Options

The following options allow you to set specific safari ball and safari step counter initial and maximum values.

Definition	Description
EXPAND_SAFARI_BALLS	If defined, the player can hold up to 0xFFFF (65535) Safari Balls!
SAFARI_ZONE_MAX_STEPS	Maximum number of Safari steps the player can have remaining (see <i>special 0x89</i> to add/remove steps).
SAFARI_ZONE_BALL_START	The number of Safari balls the player will start the safari zone with.
MAX_SAFARI_BALLS	The maximum number of Safari balls the player can have (See <i>special 0x87</i> to add/remove safari balls up to this amount).

Randomizer Options

Definition	Description
FLAG_POKEMON_RANDOMIZER	Setting this flag causes any Pokémon created to have its species randomized. The Pokémon created can fall anywhere between SPECIES_BULBASUAR and NUM_SPECIES . Pokémon banned from being created should be added to the <i>gRandomizerSpeciesBanList</i> table found in src/Tables/pokemon_tables.c . Comment out to not use this flag.

FLAG_POKEMON_LEARNSET_RANDOMIZER	Setting this flag randomizes the level-up learnset of each Pokémon. Comment out to not use this flag.
FLAG_ABILITY_RANDOMIZER	Setting this flag randomizes the possible abilities for each Pokémon. The abilities found in the gRandomizerAbilityBanList table found in <code>src/Tables/pokemon_tables.c</code> will never be chosen.

Memory Locations

<i>Definition</i>	<i>Description</i>
EXISTING_FOSSIL_IMAGE_TABLE_ADDRESS	If FOSSIL_IMAGE HACK is defined, and you already have a fossil image table inserted somewhere in your hack that you wish to use, uncomment this line and replace the given pointer with your pointer. See Special 0x18B for table details.
EXISTING_OW_TABLE_ADDRESS	If you have used JPAN's hacked engine to add new overworld tables in, and you would like to use the table already inserted in your hack to keep track of these tables, uncomment this line and replace the given pointer with your pointer. Note that error handling for invalid table ids will be disabled if you use this.

Misc Features

<i>Definition</i>	<i>Description</i>
TIME_ENABLED	Various features are updated to account for the time of day. Certain palettes are also dynamically faded depending on the time of day. Removing this feature will always result in <i>Daytime</i> .
DNS_IN_BATTLE	If TIME_ENABLED is defined, then certain background palettes will be faded dynamically in battle depending on the time of day. These values can be set in <code>include/new/dns_data.h</code> .
OVERWRITE_RIVAL	Loads the buffered rival's name for trainer classes 0x51, 0x59, and 0x5A.
TRAINER_CLASS_POKE_BALLS	Creates trainer Pokémon with custom Poke Balls based on trainer class determined by what is set in <code>src/Tables/class_based_poke_ball_table.c</code> .

TRAINERS_WITH_EVS	<p>Trainer Pokémon with a custom moveset, custom item, an AI value greater than 1, and an IV value (labeled EVs in most trainer editors) greater than 0 will have custom stats loaded from the spread number indicated by the IV value in src/Tables/trainers_with_evs_table.h. For example, setting the IV value to 1 will loaded the spread labeled “1” in <i>gTrainersWithEvsSpreads</i>.</p> <p>Modifying the required conditions to activate this feature can be done by searching for the line <i>#ifdef TRAINERS_WITH_EVS</i> in src/build_Pokémon.c.</p>
CONTINUE_LOST_BATTLES	<p>If TUTORIAL_BATTLES is defined, then if Var 0x8000 is set to 0xFEFE before a battle begins, <i>trainerbattle 0x9</i> can be used to continue a lost battle. The result of the battle will then be stored in Var LAST_RESULT (LastResult).</p> <p>If TUTORIAL_BATTLES is not defined, then <i>trainerbattle 0x9</i> will always allow a lost battle to be continued.</p>
DISPLAY_REAL_MOVE_TYPE_ON_MENU	<p>When choosing a move or viewing a Pokémon’s moves on the summary screen, the move type the move will become when used will be displayed (such as Hidden Power, Weather Ball in Weather, etc.).</p>
DISPLAY_REAL_ACCURACY_ON_MENU	<p>When pressing the <i>L-button</i> while choosing a move, the "true" move accuracy will be displayed. For example, the move <i>Psychic</i> used by a Pokémon with <i>Compound Eyes</i> will have its accuracy appear as 130.</p>
DISPLAY_REAL_POWER_ON_MENU	<p>When pressing the <i>L-button</i> while choosing a move, the "true" move power will be displayed. For example, moves like <i>Fury Cutter</i> and <i>Return</i> will show their correct power.</p>
DISPLAY_EFFECTIVENESS_ON_MENU	<p>Move types will be highlighted in the move menu based on their type</p>

	<p>effectiveness against opponents. The highlighting scheme is as follows:</p> <ul style="list-style-type: none"> • Green: Super Effective • Yellow: Not Very Effective • Red: No Effect • Bold: STAB (applies over above except for “No Effect”)
OVERWRITE_BG_FOR_LEADER_CHAMPION	Special Battle Background palettes will be loaded in for battles against Gym Leaders and the Champion, using the regular indoor background graphics as a base.
BRIDGE_FIX	The water battle background will only be loaded in battle if the player's surfing sprite is shown. This means that if the player is walking on water, the battle background loaded will be incorrect.
MEGA_EVOLUTION_FEATURE	Mega Evolutions can be used.
TUTORIAL_BATTLES	Professor Oak's tutorial will be activated for <i>trainerbattle 0x9</i> .
TANOBY_RUINS_ENABLED	Causes Unown to be spawned in maps using the Tanoby Ruins map names according to the current chamber. Error prevention has been added to also allow random Unown to be generated outside of the Tanoby Ruins maps.
ALTERING_CAVE_ENABLED	If the current map is the Altering Cave and Var 0x4024 is set, Wild Pokémon will spawn based on the contents of the var.
SWEET_SCENT_ONLY_IN_CLEAR_WEATHER	In certain generations, Sweet Scent only spawns wild Pokémon in the Overworld if the weather is clear.
OBEDIENCE_BY_BADGE_AMOUNT	Pokémon obedience is determined by the number of badges the Player has rather than by which badges the player has. The other badge defines in this case act as “number of badges acquired” instead of “acquired badge X”.
SAVE_BLOCK_EXPANSION	Expands the amount of memory that is saved when the player saves the game. This feature breaks compatibility with the FR <i>Mystery Gift</i> and <i>Trainer Tower</i> features. Uncommenting this line also requires removal of all related hooks.

	Search for <i>Save Expansion Hooks</i> in hooks . WARNING: Removing this also removes added vars, PC box space, roaming Pokémon, and many other features. It is highly recommended to keep it.
SELECT_FROM_PC	If uncommented, allows the player to select and manipulate data of Pokémon from the PC storage boxes. See PC Selection for more details.
SET_HEALING_PLACE_HACK	If uncommented, the whiteout hack from JPAN's FR engine is used, allowing <i>VAR_HEALINGMAP</i> , <i>VAR_HEALING_XPOS</i> , and <i>VAR_HEALING_YPOS</i> to be utilized to overwrite the default respawn point.
FOSSIL_IMAGE_HACK	Grants the ability to load custom images from a table using Special 0x18B . If <i>EXISTING_FOSSIL_IMAGE_TABLE_ADDRESSES</i> is commented out, then the table of images can be found by searching for <i>gFossilImageTable</i> in src/script_specials.c . Otherwise the table is loaded from <i>EXISTING_FOSSIL_IMAGE_TABLE_ADDRESSES</i> .
EVO_HOLD_ITEM_REMOVAL	Evolving a Pokémon by having it hold an item upon level up or trading removes the item after evolution (like normal). Commenting this out means Pokémon will retain their items after evolution.
EXPAND_MOVESETS	Adds level up moves for each Pokémon which can be found in src/Tables/learnsets.c . This file also includes learnsets for expanded Pokémon which are commented out by default. Comment this line if you would rather use the learnsets created in the <i>Dynamic Pokémon Expansion</i> . Commenting this line out without properly having expanded the level up moves in some way will cause Pokémon to learn garbage moves.

FATHER_PASSES_TMS	During breeding, any TMs the father knows will be passed down to the baby if it can learn that TM. This feature was removed from main series Pokémon games in Gen 6.
INHERIT_MASTER_CHERISH_BALL	If defined, an offspring can be hatched into a parent's <i>Master</i> or <i>Cherish Ball</i> (unlike in the actual games).
GIVEPOKEMON_CUSTOM_HACK	<p>The second last entry (last entry in <i>PKSV</i>) of the <i>givepokemon</i> scripting command allows you to give the player a custom designed Pokémon. Set this value to any number other than 0 and set the following vars:</p> <p>Var 0x8000: Move 1 Var 0x8001: Move 2 Var 0x8002: Move 3 Var 0x8003: Move 4 Var 0x8004: Nature Var 0x8005: 0 if not shiny. 1 if shiny. Var 0x8006: HP IV Var 0x8007: Attack IV Var 0x8008: Defense IV Var 0x8009: Speed IV Var 0x800A: Special Attack IV Var 0x800B: Special Defense IV</p> <p>If the value set for one of the moves is not a valid move (ie. 0xFFFF) the default move will be loaded in that slot.</p>
GIVEPOKEMON_BALL_HACK	The last byte of the <i>givepokemon</i> scripting command allows you to pass in a Poké Ball type to assign to the Pokémon. Ball types can be found in include/new/catching.h . Does not work in <i>PKSV</i> .
FRLG_ROAMING	When a roaming Pokémon is created, it will either be a <i>Entei</i> , <i>Raikou</i> , or <i>Suicune</i> , depending on the player's starter choice. No other roamer can be created other than these 3.
CAN_RUN_IN_BUILDINGS	Allows the player to run indoors (you're a jerk if you don't let them).

NO_POISON_IN_OW	Pokémon will not take damage from Poison in the overworld as the player walks.
POISON_1_HP_SURVIVAL	Instead of allowing Pokémon to faint from Poison in the overworld, Pokémon will survive the poison with 1 HP like in Gen 4. If NO_POISON_IN_OW is defined then this line is useless.
BW_REPEL_SYSTEM	When the player's repel wears off, they will be prompted with a textbox to use another one if they have in their bag.
AUTO_NAMING_SCREEN_SWAP	After the player types the first character as an uppercase character in any naming screen, the text will automatically flip to lowercase letters.
MULTIPLE_PREMIER_BALLS_AT_ONCE	When the Player buys more than 10 Poké Balls at once, they will receive Premier balls equal to the number of Poké Balls they bought divided by 10 (rounded down), as opposed to just a single Premier Ball regardless of how many Poké Balls were purchased.
NON_TRAINER_SPOTTING	This feature allows you to assign scripts to regular NPCs and set their trainer byte, to cause them to walk up to the player as if they were a trainer. Additionally, by setting the NPC's <i>Unknown</i> halfword in AdvanceMap (right under <i>Person Id</i>) to a flag, the game will check (and set) that flag before the NPC walks up to the player to prevent the script from running in an infinite loop (placing a flag here is not needed if you'll make the NPC disappear after their first encounter).
BIKE_ON_ANY_NON_INSIDE_MAP	Normally, the ability to bike on the map is determined by a byte in the map header. This feature makes it so the bike can be used on all outdoor maps. The bike can still be used on inside maps with the map header bike bit set.
GEN_4_PLAYER_RUNNING_FIX	Increases the lag between frames as the player OW runs, to simulate a more accurate Gen 4 running effect.

EXPAND_MOVE_REMINDER_DESCRIPTION	Move descriptions will take up 5 lines in the <i>Move Reminder</i> screen as opposed to 4. It is recommended to use this feature as many attack descriptions were expanded to take up 5 lines.																								
ITEM_PICTURE_ACQUIRE	An item image is shown in the lower right corner of the textbox when the player finds or obtains an item used one of the standard scripting functions (<i>callstd</i>). Key Items, TMs, & HMs appear enlarged in the centre of the screen instead.																								
EXPANDED_TEXT_BUFFERS	<div>Expands the number of text buffers available for string scripting. The following text buffers are now available.</div> <table><tr><th>XSE Buffer #</th><th>String Arg</th><th>Size</th></tr><tr><td>0x5</td><td>FD 07 (\v\h07)</td><td>32 bytes</td></tr><tr><td>0x6</td><td>FD 08 (\v\h08)</td><td>32 bytes</td></tr><tr><td>0x7</td><td>FD 09 (\v\h09)</td><td>32 bytes</td></tr><tr><td>0x8</td><td>FD 0A (\v\h0A)</td><td>32 bytes</td></tr><tr><td>0x9</td><td>FD 0B (\v\h0B)</td><td>32 bytes</td></tr><tr><td>0xA</td><td>FD 0C (\v\h0C)</td><td>100 bytes</td></tr><tr><td>0xB</td><td>FD 0D (\v\h0D)</td><td>100 bytes</td></tr></table>	XSE Buffer #	String Arg	Size	0x5	FD 07 (\v\h07)	32 bytes	0x6	FD 08 (\v\h08)	32 bytes	0x7	FD 09 (\v\h09)	32 bytes	0x8	FD 0A (\v\h0A)	32 bytes	0x9	FD 0B (\v\h0B)	32 bytes	0xA	FD 0C (\v\h0C)	100 bytes	0xB	FD 0D (\v\h0D)	100 bytes
XSE Buffer #	String Arg	Size																							
0x5	FD 07 (\v\h07)	32 bytes																							
0x6	FD 08 (\v\h08)	32 bytes																							
0x7	FD 09 (\v\h09)	32 bytes																							
0x8	FD 0A (\v\h0A)	32 bytes																							
0x9	FD 0B (\v\h0B)	32 bytes																							
0xA	FD 0C (\v\h0C)	100 bytes																							
0xB	FD 0D (\v\h0D)	100 bytes																							
FOOTSTEP_NOISES	When the player or any NPC walks through grass or sand, sounds will be played. The sounds that play can be altered by modifying the values in the functions PlayGrassFootstepNoise and PlaySandFootstepNoise both found in src/overworld.c . ¹																								
CAN_ONLY_USE_OMNIDIRECTIONAL_JUMP_ON_HEIGHT_2	If a tile's behaviour byte in AdvanceMap is set to MB_OMNIDIRECTIONAL_JUMP , then the player will only jump over the block when their height is 2 (0xC in																								

¹ Credits to [Squeetz](#) for the original routine.

	AdvanceMap, also labeled <i>Height 2 (0x3)</i>). This means that the player will not jump over the tile if they're surfing, for example.
HOOPA_CHANGE_IN_PC	Hoopla-Unbound will revert to its confined form when placed in or withdrawn from a PC Box.
SHAYMIN_CHANGE_IN_PC	Shaymin-Sky will revert to its land form when placed in or withdrawn from a PC Box. This feature was removed in Gen 7.
HIGH_PITCH_MEGA_PRIMAL_CRY	Whenever the cry of a Mega Evolved Pokémon is played, it will played at a higher pitch. This is useful if you're using the base form cries for the Mega Evolutions as well.
SCROLLING_MULTICHOICE	Enables scrolling multichoice menus by using special 0x158.
REPLACE_SOME_VANILLA_SPECIALS	Replaces the following FireRed specials: <i>Special 0x7C – BufferNickname</i> <i>Special 0x7D – CheckTradedPokemon</i> <i>Special 0x9E – NicknamePokemon</i> <i>Special 0x156 – StartGhostBattle</i> <i>Special 0x18B – ShowFossilImage</i> with newer versions. This will likely break compatibility with vanilla FR and require modifications to any scripts that use them.
REPLACE_ASH_WEATHER_WITH_WHITE_SANDSTORM	If uncommented (aka defined), replaces the falling ash weather effect with a white version of the sandstorm weather effect

Misc Battle Effect Options

Definition	Description
OLD_BURN_DAMAGE	Burn damage does 1/8 of max health instead of 1/16.
OLD_PARALYSIS_SPD_DROP	Paralysis lower Speed down to 1/4 instead of ½.
OLD_CONFUSION_CHANCE	Confusion stops attacks 50% of the time instead of 33%.
INFINITE_WEATHER	Weather abilities make weather last for infinite turns.
INFINITE_TERRAIN	Terrain abilities make terrain last for infinite turns.
NO_SHEER_COLD_NERF	Remove all Gen 7 Sheer Cold nerfs.
OLD_MOVE_SPLIT	The Physical/Special split is based on move types. Status moves are still set with the split byte, however.

Ability Options

Definition	Description
OLD_GALE_WINGS	Gale Wings activates regardless of the user's HP.
OLD_PRANKSTER	Prankster won't fail against Dark-Types.

Damage Calculation Options

Definition	Description
OLD_CRIT_DAMAGE	Critical hits to do 2x damage; 3x with Sniper .
CRIT_CHANCE_GEN_6	Uses the Gen 6 crit chance.
CRIT_CHANCE_GEN_2_TO_5	Uses the Gen 2-5 crit chance. Cannot be used in conjunction with CRIT_CHANCE_GEN_6 .
BADGE_BOOSTS	Having badges gives the player's Pokémon stat boosts .
OLD_ATE_BOOST	" Ate " abilities give a 1.3x boost instead of 1.2x.
OLD_GEM_BOOST	Gems give a 1.5x boost instead of 1.2x.
OLD_EXPLOSION_BOOST	Exploding moves halve the target's Defense.
OLD_HIDDEN_POWER_BP	Hidden Power has its Base Power calculated from the attacker's IVs.
PORTAL_POWER	Enables Hoopa-Unbound's signature ability in Pokémon Unbound, <i>Portal Power</i> . This reduces the power of non-contact moves by 25%.
OLD_SOUL_DEW_EFFECT	Soul Dew doubles Latios & Latias' Sp. Atk & Sp. Def.
OLD_PARENTAL_BOND_DAMAGE	The second hit of Parental Bond does 50% of the original damage instead of 25%.
GEN_6_POWER_NERFS	Reduces the base power of any moves that had their base Power reduced in X&Y. Commenting out increases the power of moves that would have had a higher power had they been released prior (such as Fleur Cannon and Origin Pulse to 140).
GEN_7_POWER_NERFS	Reduces the base power of Sucker Punch to 70.
BUFFED_LEECH_LIFE	Increases the power of Leech Life to 80.
DARK_VOID_ACC_NERF	Reduces the accuracy of Dark Void to 50.

Capturing Pokémon Options

Definition	Description
NO_HARDER_WILD_DOUBLES	In Gen 5, Pokémon encountered in wild double battles were harder to catch (based on how many species are owned). This feature implements that catch rate decrement.
CRITICAL_CAPTURE	Allows for Critical Capture to occur. The odds at which this occurs can be found in the function: <i>static bool8 CriticalCapture(u32 odds)</i> found in <code>src/catching.c</code> . They are based on how many species are owned.

Exp. Gain Options

Definition	Description
TRAINER_EXP_BOOST	Gives an Exp. boost for defeating a trainer's Pokémon. (Pre Gen 7)
OLD_EXP_SPLIT	Exp. is split amongst all participating Pokémon. (Pre Gen 6)
FLAT_EXP_FORMULA	Use a Flat Exp. calculation formula. (Gens 2- 4, 6)
GEN_7_BASE_EXP_YIELD	Base Exp. is retrieved from the table gBaseExpBySpecies found in the file src/Tables/Experience_Tables.c , instead of being loaded from the Pokémon's base stats. This is done to account for larger Exp. values that started in Gen 5. The table is pre-set to match Gen 7 Exp. values .
CAPTURE_EXPERIENCE	When a Pokémon is caught, experience will be rewarded as if the caught Pokémon fainted.

Other Battle Options

Definition	Description
NO_GHOST_BATTLES	Disables the Ghost battle feature from Pokémon Tower in Lavender town.
GEN4_PLUS_SELECTION_SCREEN	This does not give the Gen 4+ selection screen , it only adds in-battle features that supports it.
OBEDIENCE_CHECK_FOR_PLAYER_ORIGINAL_POKÉMON	Opens up the possibility that the Player's Pokémon can disobey them, as opposed to just traded Pokémon.
WILD_ALWAYS_SMART	All wild Pokémon use AI features meant for trainers.
HAIL_IN_BATTLE	Enables the Hail weather effect in battle when the OW weather is set to WEATHER_STEADY_SNOW (0x7).
FOG_IN_BATTLE	Enables the Fog weather effect in battle. Do not enable this feature without first enabling one of the fog features below!
FOG_IN_BATTLE_1	Enables the Fog weather effect when the OW weather is set to WEATHER_FOG_1 (0x6).
FOG_IN_BATTLE_2	Enables the Fog weather effect when the OW weather is set to WEATHER_FOG_2 (0x9).
FOG_IN_BATTLE_3	Enables the Fog weather effect when the OW weather is set to WEATHER_FOG_3 (0xA).
HIDE_HEALTHBOXES_DURING_ANIMS	Hides the healthboxes (battle bars, etc.) during move animations, and some special animations (like Mega Evolution). This has been done since Gen 4.
DONT_HIDE_HEALTHBOXES_ATTACKER_STAT_US_MOVES	If HIDE_HEALTHBOXES_DURING_ANIMS is defined, when the attacker is using a move that only targets itself, the healthboxes will

	not be hidden. This was done in Gen 4.
ENCOUNTER_MUSIC_BY_CLASS	The music played when a trainer spots the player in the overworld is determined by the trainer class, rather than the music Id set in the trainer data. The song options are listed in src/Tables/Music_Tables.c and can be modified by changing the values in <i>gClassBasedTrainerEncounterBGM</i> . Any class not defined in the array will be automatically set to BGM_EYE_BOY .
OKAY_WITH_AI_SUICIDE	The AI is allowed to use self-destructing moves.
HEALTHBAR_TYPE_ICONS	Pokémon types will always been shown next to a Pokémon's healthbar in battle. If not defined, this will only be shown in Camomons battles. The coordinates of these icons can be adjusted by modifying <i>sTypeIconPositions</i> found in src/move_menu.c .

DexNav Options

The following options relate to the [DexNav](#) feature and can be found in `include/new/dexnav_config.h`.

Definition	Description
DEXNAV_TIMEOUT	The number of seconds before the <i>DexNav</i> search times out for the player. The maximum is 1092 seconds.
SNEAKING_PROXIMITY	The amount of tiles (both horizontally and vertically) the player is allowed to be before having to sneak to avoid the Pokémon from running away. <i>Sneaking</i> is defined by not running or biking.
SEARCHLEVEL0_MOVECHANCE SEARCHLEVEL5_MOVECHANCE SEARCHLEVEL10_MOVECHANCE SEARCHLEVEL25_MOVECHANCE SEARCHLEVEL50_MOVECHANCE SEARCHLEVEL100_MOVECHANCE	The chance in percent that a Pokémon with any of these search levels will be encountered with an Egg Move in the first slot.
SEARCHLEVEL0_ABILITYCHANCE SEARCHLEVEL5_ABILITYCHANCE SEARCHLEVEL10_ABILITYCHANCE SEARCHLEVEL25_ABILITYCHANCE SEARCHLEVEL50_ABILITYCHANCE SEARCHLEVEL100_ABILITYCHANCE	The chance in percent that a Pokémon with any of these search levels will be encountered with its Hidden Ability (if it can have one).
SEARCHLEVEL0_ITEM SEARCHLEVEL5_ITEM SEARCHLEVEL10_ITEM SEARCHLEVEL25_ITEM SEARCHLEVEL50_ITEM SEARCHLEVEL100_ITEM	The chance in percent that a Pokémon with any of these search levels will be encountered with a held item (if it can have one).
SEARCHLEVEL0_ONESTAR SEARCHLEVEL5_ONESTAR SEARCHLEVEL10_ONESTAR SEARCHLEVEL25_ONESTAR SEARCHLEVEL50_ONESTAR SEARCHLEVEL100_ONESTAR SEARCHLEVEL0_TWOSTAR SEARCHLEVEL5_TWOSTAR SEARCHLEVEL10_TWOSTAR SEARCHLEVEL25_TWOSTAR SEARCHLEVEL50_TWOSTAR SEARCHLEVEL100_TWOSTAR SEARCHLEVEL0_THREESTAR SEARCHLEVEL5_THREESTAR SEARCHLEVEL10_THREESTAR SEARCHLEVEL25_THREESTAR SEARCHLEVEL50_THREESTAR SEARCHLEVEL100_THREESTAR	The chance in percent that a Pokémon with any of these search levels will be encountered with X number of perfect IVs, where X is equal to the number of stars.

Engine Setup

Mega Evolution / Primal Reversion / Ultra Burst

Before setting up Mega Evolution, two things must be done. First, make sure `EVOS_PER_MON` in the config file is set to the correct number (it should be the same as the number + 1 at 0x43116 in your rom). Second, if you're not using the *Dynamic Pokémon Expansion*, your Pokémon Editor of choice will need to be modified:

G3T:

In your Gen3Tools folder, open up **Customisation/Pokémon Editor.ini**, and add the line **FE=Mega Evolution** to the bottom of the file.

```
0D=Allow PKMN creation
0E=Create extra PKMN
0F=Beauty
FE=Mega Evolution
```

D&D:

Has Mega Evolution pre-installed, however it cannot set up Wish-based Mega Evolution correctly.

G3HS:

1. Open up the file **PokeRoms.ini**, find your rom code.
2. Modify "evolutionmethods" such that the 254th evolution method is set to *Mega Evolution*:

[illegible]

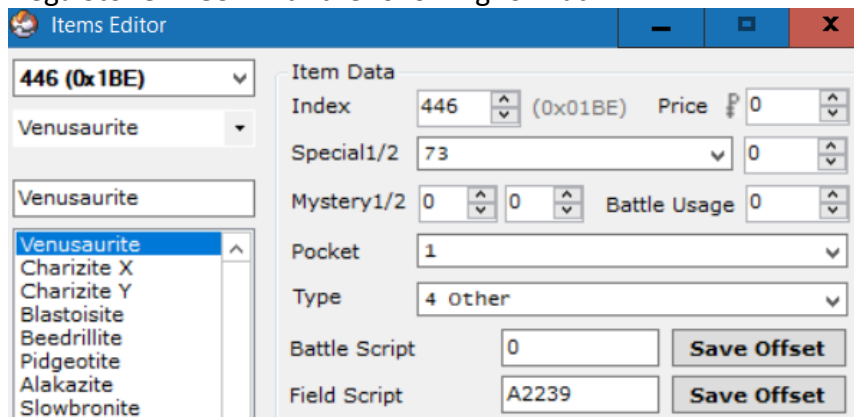
3. Modify “evomethodsproperties” such that the 254th method is set to *Item*.

[Mega Evolution](#)

Mega Evolution set up is similar to the how the [previous](#) Mega Evolution system by Touched was set up. If a Pokémon is able to Mega Evolve, Mega Evolution can be triggered by pressing start on the move menu once the mega trigger appears.

NOT Using Dynamic Pokémon Expansion:

1. Create a Mega Stone in G3T with the following format:



The screenshot shows the 'Items Editor' window. On the left, a list of items includes Venusaurite, Charizite X, Charizite Y, Blastoisite, Beedrillite, Pidgeotite, Alakazite, and Slowbronite. The 'Item Data' section on the right is configured as follows:

Field	Value
Index	446 (0x01BE)
Price	0
Special1/2	73
Mystery1/2	0
Battle Usage	0
Pocket	1
Type	4 other
Battle Script	0
Field Script	A2239

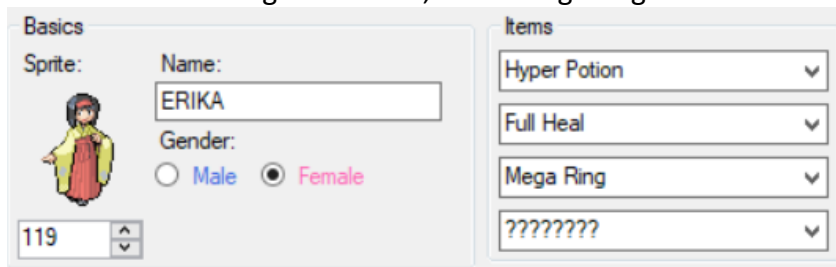
Buttons for 'Save Offset' are present next to the Battle Script and Field Script fields.

2. Create a Mega Ring key item. Its format is the same as any other key item.
3. Update the definition for the Mega Ring item in **include/constants/items.h**.
4. Open the file **src/mega.c** and search for *KeystoneTable*. Add your mega ring item to this table:

```
item_t KeystoneTable[] =  
{  
    ITEM_MEGA_RING,  
};
```

NOTE: The item does not need to be called *Mega Ring*. Any item added to this table can act as a mega ring.

5. If **DEBUG_MEGA** is defined in the config file, then Mega Evolution can be used from the start of the game without the requirement of having the Mega Ring in the bag. Otherwise, the player must have a Mega Ring item in the bag to use Mega Evolution (Mega Evolution can always be used in Link Battles or the Battle Frontier). If you want a trainer to be able to use Mega Evolution, add a mega ring item to their battle items:



The screenshot shows a trainer configuration window. The 'Basics' section on the left includes a sprite, name 'ERIKA', gender 'Female', and level '119'. The 'Items' section on the right shows a list of items: Hyper Potion, Full Heal, Mega Ring, and a placeholder '????????'. 'Mega Ring' is selected in the list.

Don't forget to give the trainer's Pokémon the relevant Mega Stone to hold!

6. Find the Pokémon in G3T that you wish to bestow the ability of Mega Evolution upon. Set up the evolution data with following template:

Venusaur

Sprites

Type
Grass
Poison

Abilities
Overgrow

Held Items
????????
????????

Evolution
M Venusaur | Mega Evolution

Condition Mega Evolution

Nothing required 190

Evolve to Go! M Venusaur

Unknown Bytes 0 0

Venusaurite

Brand New Species

In the above image:

- “Nothing required” is set to the mega stone item index created earlier.
- “Evolve to” is set to a new species representing the mega form (you’ll need to expand Pokémon or replace an existing one). Make sure this mega form has the same gender rate and Exp rate as the non-mega form or things will get messed up.

Wish Mega Evolution (for Rayquaza) should be set up using the following template:

Rayquaza

Sprites

Type
Dragon
Flying

Abilities
Air Lock

Held Items
????????
????????

Evolution
M Rayquaza | Mega Evolution

Condition Mega Evolution

Nothing required 47

Evolve to Go! M Rayquaza

Unknown Bytes 2 0

In the above image:

- “Nothing required” should be set to the move id for *Dragon Ascent* (0x22F / 559). G3T has issues with setting numbers past 0xFF, so you’ll need to hex edit or use a different Pokémon editor to set the proper value for *Dragon Ascent*.
- “Evolve to” is set to a new species representing Mega Rayquaza (you’ll need to expand Pokémon or replace an existing one). Make sure Mega Rayquaza has the same gender rate and Exp rate as Rayquaza or things will get messed up.
- The first of “Unknown Bytes” should be set to 2.

7. Set the Mega form's evolution data like the following template:

Notice that the item is left blank. Also notice that *Mega Rayquaza* still has its first unknown byte set to 2. Do not forget this!

8. Now just give the player a mega ring item and a Pokémon with the appropriate mega stone and they will be able to Mega Evolve!

Using Dynamic Pokémon Expansion:

Add evolution methods in **src/Evolution_Table.c** with the following format:

Regular Mega Evolution:

```
[SPECIES_VENUSAUR] = {{EVO_MEGA, ITEM_VENUSAURITE, SPECIES_VENUSAUR_MEGA, MEGA_VARIANT_STANDARD}},
[SPECIES_VENUSAUR_MEGA] = {{EVO_MEGA, ITEM_NONE, SPECIES_VENUSAUR, MEGA_VARIANT_STANDARD}},
```

Wish Mega Evolution:

```
[SPECIES_RAYQUAZA] = {{EVO_MEGA, MOVE_DRAGONASCENT, SPECIES_RAYQUAZA_MEGA, MEGA_VARIANT_WISH}},
[SPECIES_RAYQUAZA_MEGA] = {{EVO_MEGA, MOVE_NONE, SPECIES_RAYQUAZA, MEGA_VARIANT_WISH}},
```

Notes about Mega Evolution:

- Only a single Mega Evolution may be used by each side during any given battle.
- In multi battles, both trainers can Mega Evolve a single Pokémon, effectively allowing for two Mega Evolutions per side during any given battle.
- Mega Evolution is not prevented if any Pokémon on the side Ultra Bursted or underwent Primal Reversion.
- If Rayquaza holds a Z-Crystal, it'll be unable to Mega Evolve.

Primal Reversion

Primal Reversion does not rely on a mega ring to activate and will thus always activate if Kyogre or Groudon hold the appropriate item. Much less setup is required for Primal Reversion.

NOT Using Dynamic Pokémon Expansion:

1. Find the Red Orb and Blue Orb in G3T's item editor and modify them to match the following:

The image shows two side-by-side screenshots of the G3T item editor. The left screenshot is for the Red Orb (Index 276, 0x114). It shows the 'Item Data' tab with 'Special1/2' set to 93, 'Mystery1/2' set to 0, 'Pocket' set to 1, and 'Type' set to 4 Other. The right screenshot is for the Blue Orb (Index 277, 0x115). It shows the 'Item Data' tab with 'Special1/2' set to 93, 'Mystery1/2' set to 0, 'Pocket' set to 1, and 'Type' set to 4 other. Both items have 'Battle Script' set to 0 and 'Field Script' set to A2239.

The item effect should be set as 93 and the item quality should be set as 0 for the Red orb and 1 for the Blue Orb. These numbers determine whether the Primal Reversion is *Red* (0) or *Blue* (1).

2. Modify Kyogre and Groudon's evolution data in G3T to match the following:

The image shows two side-by-side screenshots of the G3T evolution editor. The left screenshot is for Kyogre, showing its evolution to PKyogre. The right screenshot is for Groudon, showing its evolution to PGroudon. Both evolutions are set to 'Mega Evolution' with 'Nothing required' and 'Evolve to' set to the respective primal form. The 'Unknown Bytes' are set to 1 and 0.

In the above images:

- “Nothing required” is set to the *Blue Orb* item index for Kyogre and to the *Red Orb* item index for Groudon. Note again that G3T does not represent these item indices correctly so you may need to use another editor.
- “Evolve to” is set to a new species representing the primal form (you’ll need to expand Pokémon or replace an existing one). Make sure this primal form has the same gender rate and Exp rate as the non-primal form or things will get messed up.
- The first of the “Unknown Bytes” is set to 1 to represent Primal Reversion.

3. Set up the primal forms' evolution data in G3T to match the following:

The image shows two side-by-side screenshots of the G3T evolution editor. The left window is for PKyogre, and the right window is for PGroudon. Both windows have a 'Sprites' button in the top right. The 'Type' dropdown is set to 'Water' for PKyogre and 'Ground' for PGroudon. The 'Abilities' dropdown is set to 'Eordie' for PKyogre and 'Elat' for PGroudon. The 'Evolution' dropdown is set to 'Kyogre | Mega Evolution' for PKyogre and 'Groudon | Mega Evolution' for PGroudon. The 'Condition' dropdown is set to 'Mega Evolution' for both. The 'Nothing required' field is set to 0 for both. The 'Evolve to' dropdown is set to 'Kyogre' for PKyogre and 'Groudon' for PGroudon. The 'Held Items' field is set to '????????' for both. The 'Unknown Bytes' field is set to 1 for both.

Notice that the items are left blank. Also notice that both Primal Pokémon still have their first unknown bytes set to 1. Do not forget this!

Note about Primal Reversion:

- The alpha and omega symbols on the health bar are generated based on which species is in its Primal form. By default, the alpha symbol is set to appear if the species is Primal Kyogre, and the omega symbol is set to appear if the species is Primal Groudon. To change this requirement, modify the following functions in **src/mega.c**:

```
bool8 IsBluePrimalSpecies(u16 species)
{
    return species == SPECIES_KYOGRE_PRIMAL;
}

bool8 IsRedPrimalSpecies(u16 species)
{
    return species == SPECIES_GROUDON_PRIMAL;
}
```

These can easily be modified by adding species to compare to. For example, to make the omega symbol to appear for Primal Dialga as well, make the following modification:

```
bool8 IsRedPrimalSpecies(u16 species)
{
    return species == SPECIES_GROUDON_PRIMAL || species == SPECIES_DIALGA_PRIMAL;
}
```

Don't forget to also define **PKMN_PRIMAL_DIALGA** in **include/constants/species.h**!

Using Dynamic Pokémon Expansion:

Add evolution methods in **src/Evolution_Table.c** with the following format:

```
[SPECIES_GROUDON] = {{EVO_MEGA, ITEM_RED_ORB, SPECIES_GROUDON_PRIMAL, MEGA_VARIANT_PRIMAL}},
[SPECIES_GROUDON_PRIMAL] = {{EVO_MEGA, ITEM_NONE, SPECIES_GROUDON, MEGA_VARIANT_PRIMAL}},
```

Ultra Burst

Primal Reversion does not rely on a mega ring to activate and will thus always activate if Necrozma holds the appropriate item. Much less setup is required for Ultra Burst.

NOT Using Dynamic Pokémon Expansion:

1. Create an *Ultraneurozium Z* item in G3T with the following format:

The screenshot shows the G3T Item Editor interface. On the left, a list of items is shown with 'Necrozium Z' selected. The main panel displays the 'Item Data' for this item. The 'Index' is set to 446 (0x01BE), 'Price' is 0, 'Special1/2' is 73, 'Mystery1/2' is 0, 'Battle Usage' is 0, 'Pocket' is 1, and 'Type' is 4 (other). The 'Battle Script' is 0 and the 'Field Script' is A2239. There are 'Save Offset' buttons for both scripts.

2. Modify **both** Necrozma forms' evolution data in G3T to match the following:

The screenshot shows two side-by-side windows from the G3T Evolution Editor. The left window is for 'Necrozma-M' and the right is for 'Necrozma-W'. Both windows show the 'Evolution' tab. The 'Type' is set to 'Psychic' for Necrozma-M and 'Ghost' for Necrozma-W. The 'Evolution' is set to 'Necrozma-U | Mega Evolution'. The 'Condition' is 'Mega Evolution'. The 'Nothing required' field is set to 189. The 'Evolve to' field is set to 'Go! Necrozma-U'. The 'Unknown Bytes' field is set to 3. There are 'Sprites' buttons for both forms.

In the above images:

- “Nothing required” is set to the *Ultraneurozium Z* item index. Note again that G3T does not represent these item indices correctly so you may need to use another editor.
- “Evolve to” is set to a new species representing *Ultra Necrozma* (you’ll need to expand Pokémon or replace an existing one). Make sure all *Necrozma* forms have the same gender rate and Exp rate or things will get messed up.
- The first of the “Unknown Bytes” is set to 3 to represent Ultra Burst.

- Set up the *Ultra Necrozma*'s evolution data in G3T to match the following:

The screenshot shows the G3T interface for editing the evolution data of Necrozma-U. The 'Type' section shows 'Psychic' and 'Dragon'. The 'Abilities' section shows 'Neuroforce'. The 'Held Items' section shows two blank slots. The 'Evolution' section shows a list with 'Necrozma-M' and 'Necrozma-W', both set to 'Mega Evolution'. The 'Condition' is set to 'Mega Evolution'. The 'Nothing required' field is set to 0. The 'Evolve to' field is set to 'Necrozma-M'. The 'Unknown Bytes' field is set to 3.

In the above image:

- The items are left blank.
- The first unknown byte is set to 3. Do not forget this!

Note about Ultra Burst:

- Contrary to what is shown in the above image, *Ultra Necrozma* does **not** need reversion data from both *Necrozma* fusion forms (it needs for at least one of them). *Ultra Necrozma* will always revert to the form it *Ultra Bursted* from at the end of the battle, regardless of which species is written in its evolution data. If the *Ultra Necrozma* was encountered in the wild, it will revert to the first species in its evolution list by default.
- Ultra Burst* is triggered the same way as *Mega Evolution* on the move menu.
- As *Ultra Burst* is not considered *Mega Evolution*, *Ultra Necrozma* can still use Z-Moves if it knows the appropriate base move (*Photon Geyser* by default).

Using Dynamic Pokémon Expansion:

Add evolution methods in **src/Evolution_Table.c** with the following format:

```
[SPECIES_NECROZMA_MANE] = {{EVO_MEGA, ITEM_ULTRA_NECROZIUM_Z, SPECIES_NECROZMA_ULTRA, MEGA_VARIANT_ULTRA_BURST}},
[SPECIES_NECROZMA_WINGS] = {{EVO_MEGA, ITEM_ULTRA_NECROZIUM_Z, SPECIES_NECROZMA_ULTRA, MEGA_VARIANT_ULTRA_BURST}},
[SPECIES_NECROZMA_ULTRA] = {{EVO_MEGA, ITEM_NONE, SPECIES_NECROZMA_MANE, MEGA_VARIANT_ULTRA_BURST},
                             {EVO_MEGA, ITEM_NONE, SPECIES_NECROZMA_WINGS, MEGA_VARIANT_ULTRA_BURST}},
```

Z-Moves

[Z-Moves](#) work akin to how they work in real Pokémon games.

If a Pokémon holds a *Z-Crystal* corresponding to a specific type, any move of that type can be turned into a Z-Move by pressing the *Start*-button on the move menu, and then the *A*-button to confirm the selection. If a move cannot be turned into a Z-Move, the *Start*-button will do nothing.

If a certain Pokémon holds its signature *Z-Crystal*, then its signature move can be turned into its signature *Z-Move* (also with the *Start*-button). A list of these signature Z-Moves can be found under [sSpecialZMoveTable](#) in `src/set_z_effect.c`.

The only setup required for Z-Moves involves the creation of Z-Crystals. Each Z-Crystal should be created in G3T with the following format:

The screenshot shows the G3T item editor interface. On the left, a list of memory banks is visible, with 'Flyinium Z' selected. The main area displays the 'Item Data' for '672 (0x2A0)'. The 'Index' is set to 563 (0x0233) and the 'Price' is 200. The 'Special1/2' is set to 130 and the 'Quality' is set to 2. The 'Mystery1/2' is set to 1 and 0, and the 'Battle Usage' is 0. The 'Pocket' is set to 1 and the 'Type' is '4 Other'. The 'Battle Script' is 0 and the 'Field Script' is A2239. There are 'Save Offset' buttons for both the Battle and Field Scripts.

The item effect should be set as 130 and the item quality should be set to the move type the Z-Crystal works for (in the above image it is set to 2 [[TYPE_FLYING](#)]). The *Mystery 1* byte can also be set to 1 to remove the item quantity for Z-Crystals (acts as if the player only has the single, unique Z-Crystal).

Special Z-Crystals should be set up similarly, the key difference being that item quality should be set to 255.

The screenshot shows the G3T item editor interface for 'Eevium Z'. The 'Item Data' for '673 (0x2A1)' is displayed. The 'Index' is 563 (0x0233) and the 'Price' is 200. The 'Special1/2' is 130 and the 'Quality' is set to 255. The 'Mystery1/2' is 1 and 0, and the 'Battle Usage' is 0. The 'Pocket' is 1 and the 'Type' is '4 Other'. The 'Battle Script' is 0 and the 'Field Script' is A2239. 'Save Offset' buttons are present for both scripts.

Once the Z-Crystals are created, have a Pokémon hold one, give it the appropriate move, and then watch the magic happen!

Trainer Sliding Messages²

In generations after Gen 3, Trainers could interrupt the battle with a message. This engine supports three kinds of those messages:

1. After the opponent's first Pokémon faints.
2. After the opponent's last Pokémon is sent in.
3. When the opponent's last Pokémon is low on health.

Trainers can have any combination (or none at all) of the above messages. To set these up, do the following:

1. Navigate to **src/trainer_sliding.c** and search for *sTrainerSlides*. This is the table used to define the sliding messages.
2. Add an entry with the following format:

```
{0x59, sText_BenFirstMonDown, sText_BenLastSwitchIn, sText_BenLastLowHP},
```

Where 0x59 is the trainer Id (Youngster Ben in this case), *sText_BenFirstMonDown* is the message displayed when the opponent's first Pokémon has fainted, *sText_BenLastSwitchIn* is the message displayed when the opponent switches in their last Pokémon, and *sText_BenLastLowHP* is the message displayed when the opponent's last Pokémon is on low health. If you would not like the trainer to say anything at one of these stages, simply replace the entry with *NULL*:

```
{0x59, sText_BenFirstMonDown, NULL, sText_BenLastLowHP},
```

In this case, the trainer will say something after the first Pokémon is defeated and when their last Pokémon is low on HP, but not after they send in their last Pokémon.

3. In the file **include/new/trainer_sliding_data.h**, add declarations for the strings you've defined. For example, for the first message *struct* declared above, the bottom of the file should look like this:

```
extern u8 BattleScript_TrainerSlideMsgEnd2[];  
  
extern u8 sText_BenLastSwitchIn[];  
extern u8 sText_BenLastLowHP[];  
extern u8 sText_BenFirstMonDown[];
```

² Credits to the [Emerald Battle Engine Upgrade V2.0](#) for the original source code.

4. Open the file **strings/trainer_sliding_strings.string** and add entries for the strings you've defined. For example, for the first message *struct* declared above:

```
#org @sText_BenFirstMonDown  
You hurt my friend!\p
```

```
#org @sText_BenLastSwitchIn  
It's all or nothing, now!\p
```

```
#org @sText_BenLastLowHP  
Oh, no[.]\p
```

The format for the strings follows similarly to *XSE*, the key difference being that each line does **not** start with "`=`". An equal's sign at the beginning of the line will be treated as such and be seen in the game. Also note that each line ends with "`\p`" in order to wait for the player's key press.

Follows these steps and continuously add new entries to the table to add flavour to battles!



Multi Battles

There are 4 different types of [Multi Battles](#) supported by this engine:

1. Player Vs. Two Trainer Opponents.
 - Can be set up in a script or by being spotted by two different trainers.
2. Player & Partner Vs. Two Trainer Opponents.
3. Player & Partner Vs. One Trainer Opponent.
4. Player & Partner Vs. Two Wild Pokémon.

There are two different ways of setting these up:

XSE Friendly Method

This method allows you to set up multi battles through scripted events.

- To set up a battle against two opponents:
 - Set the flag `FLAG_TWO_OPPONENT`.
 - Set the var `VAR_SECOND_OPPONENT` to the trainer id of the second trainer.
 - Use the *loadpointer* scripting command in conjunction with special `0xXX` to load the second trainer's defeat text.
 - Then use the scripting command *trainerbattle 0x3* (or *0x9*) to start the battle.
- To set up a battle with a partner:
 - Set the flag `FLAG_TAG_BATTLE`.
 - Set the var `VAR_PARTNER` to the trainer id of the partner.
 - Set the var `VAR_PARTNER_BACKSPRITE` to the [backsprite id](#) of the partner.
 - Then use the scripting command *trainerbattle 0x3* (or *0x9*) to start the battle.

This will initiate a battle with a partner against a single opponent.

- To battle with a partner and two opponents, set all flags and vars listed in the previous two steps, and then use the scripting command *trainerbattle 0x3* (or *0x9*) to start the battle. A sample script to do this looks as follows:

```
#define FLAG_TWO_OPPONENT 0x909
#define FLAG_TAG_BATTLE 0x908
#define VAR_SECOND_OPPONENT 0x5010
#define VAR_PARTNER 0x5011
#define VAR_PARTNER_BACKSPRITE 0x5012
#define SPECIAL_LOAD_SECOND_DEFEAT_TEXT 0xAC

#org @start
setflag FLAG_TWO_OPPONENT 'Setup battle against two opponents
setflag FLAG_TAG_BATTLE 'Setup battle with partner
setvar VAR_SECOND_OPPONENT 0x59 'Second opponent is Youngster Ben
setvar VAR_PARTNER 0x5B 'Team up with Youngster Josh
setvar VAR_PARTNER_BACKSPRITE 0x2 'Brendan's Backsprite
loadpointer 0x0 @SecondTrainerDefeatText
special SPECIAL_LOAD_SECOND_DEFEAT_TEXT
trainerbattle 0x3 0x5A 0x0 @FirstTrainerDefeatText 'First opponent is Youngster Calvin
end
```

Non-XSE Friendly Method

This method for setting up multi battles is not possible to code in XSE, but it is significantly easier to code and allows more versatility with random trainer scripts. It is recommended to use the Thumb assembler in conjunction the XSE defines provided in this engine to compile these custom scripts (which can then be called after inserting the hex data).

- To set up a battle against two opponents, use the following scripting command:

trainerbattle 0xB FOE_1_ID FOE_2_ID FOE_1_NPC_ID FOE_2_NPC_ID 0x0

INTRO_TEXT_A INTRO_TEXT_B DEFEAT_TEXT_A DEFEAT_TEXT_B CANNOT_BATTLE_TEXT

Where:

- **FOE_1_ID**: The trainer Id of the first opponent.
- **FOE_2_ID**: The trainer Id of the second opponent.
- **FOE_1_NPC_ID**: The local Id (person Id in AdvanceMap) of the first opponent.
- **FOE_2_NPC_ID**: The local Id (person Id in AdvanceMap) of the second opponent.
- **INTRO_TEXT_A**: The intro battle text said by the first opponent.
- **INTRO_TEXT_B**: The intro battle text said by the second opponent.
- **DEFEAT_TEXT_A**: The defeat text said by the first opponent.
- **DEFEAT_TEXT_B**: The defeat text said by the second opponent.
- **CANNOT_BATTLE_TEXT**: The text said by either opponent when the player doesn't have enough viable Pokémon to fight with.

This **trainerbattle 0xB** command is special such that you can assign it to random NPCs to effectively make better random double battles than with a *Twins* class, for example. When using this on two random NPCs, make sure they stand next to each other! Otherwise it'll look off when they walk towards the player together.

- To set up a battle with a partner, use the following scripting command:

trainerbattle 0xC FOE_ID PARTNER_ID PARTNER_BACKSPRITE_ID 0x0 DEFEAT_TEXT

Where:

- **FOE_ID**: The trainer Id of the opponent.
- **PARTNER_ID**: The trainer Id of the player's partner.
- **PARTNER_BACKSPRITE_ID**: The backsprite Id of the player's partner.
- **DEFEAT_TEXT**: The text said when the opponent loses the battle.

- To battle with a partner and two opponents, use the following scripting command:

trainerbattle 0xA FOE_1_ID FOE_2_ID PARTNER_ID PARTNER_BACKSPRITE_ID 0x0

DEFEAT_TEXT_A DEFEAT_TEXT_B

Where each of the title definitions is the same as listed *trainerbattle 0xB* and *trainerbattle 0xC*. Note that both *trainerbattle 0xB* and *trainerbattle 0xC* cannot be used on random NPCs. They must be used from within an event script.

And with that, you can set up amazing multi battles!

Wild Double Battles

Encountering two wild Pokémon at once was introduced in Gen 4 with it occurring when the player was teamed up with another player. Then, in Gen 5, it became possible to encounter two Pokémon at once in special grass. This engine supports both of those features.

Wild Double Battles With Partner

If there is ever a situation where you'd like all wild battles in a given area to be with a partner, add the following as an *On entering map/on menu close [5]* level script in AdvanceMap:

```
#define FLAG_DOUBLE_WILD_BATTLE 0x9F9
```

```
#org @start
```

```
setflag FLAG_DOUBLE_WILD_BATTLE
```

```
setflag FLAG_TAG_BATTLE 'Setup battle with partner (same as above)
```

```
setvar VAR_PARTNER_BACKSPRITE 0x2 'Brendan's backsprite (same as above)
```

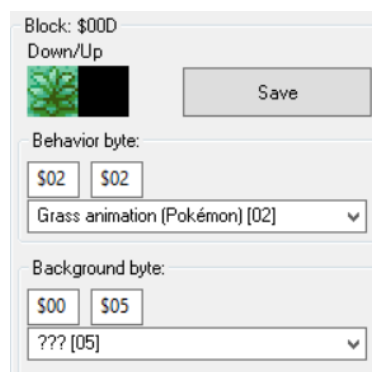
```
end
```

This will cause all battles against trainers on the map to be fought with a partner, and all wild battles fought on the map to be against two Pokémon. Conditions can also be added to the script (such as a *checkflag* to only execute the script if some flag is set). Don't forget to clear these flags once the player no longer needs a partner!

Wild Double Grass

Special grass tiles that can initiate wild battles can be created as well. Grass with a background byte of 0x5 will have a chance of starting a wild double battle, and grass with a background byte of 0x25 will have a chance of starting a wild double battle and be covered by the player. Wild double water tiles are similarly done using 0x6 and 0x26. The chance a wild double battle will be started when walking in this grass is determined by the value set (in percent) in *WILD_DOUBLE_RANDOM_CHANCE*.

Sample double wild grass in AdvanceMap:



Scripted Wild Double Battles

Here is a sample script:

```
wildbattle 0xFFFF 0x0 0x0 'Double indicator  
wildbattle PKMN_CLEFAIRY 20 ITEM_NONE  
wildbattle PKMN_PIKACHU 20 ITEM_NONE  
special 0x138 'Starts the battle  
waitstate
```

I'm not entirely sure if this script will compile properly in XSE, but if you use it in PKSV by replacing *wildbattle* with *battle* and *waitstate* with *waitspecial* the battle will begin properly.

Trainer Backsprites

Adding a backsprite into the game has never been easier.

1. Go to **graphics/Backsprites** and add a minimum of **4** backsprites to a single file for your new trainer. Make sure they're each a size of 64 x 64 and indexed to 16 colours! Sample *Brendan* and *May* backsprites come bundled with the engine.



gTrainerBackPic_
Brendan.png

2. Open **include/constants/trainers.h** and define a new constant for your new backsprite. I will call the tutorial one **TRAINER_BACK_PIC_BRENDAN_NEW**:

```
//Backsprites
#define TRAINER_BACK_PIC_RED           0
#define TRAINER_BACK_PIC_LEAF         1
#define TRAINER_BACK_PIC_BRENDAN      2
#define TRAINER_BACK_PIC_MAY           3
#define TRAINER_BACK_PIC_POKE_DUDE    4
#define TRAINER_BACK_PIC_OLD MAN      5
#define TRAINER_BACK_PIC_BRENDAN_NEW  6
```

3. Open **src/Tables/Backsprite_Tables.c** to allow the game to read your new backsprites.
4. Search for *gTrainerBackPicPaletteTable* and right above it add two lines with format:
extern const u8 [FILE_NAME_NO_EXTENSION]Pal[];
#define gTrainerPalette_[NAME] gTrainerBackPic_[FILE_NAME_NO_EXTENSION]Pal

So our entry for the new Brendan would look like:

```
extern const u8 gTrainerBackPic_BrendanPal[];
#define gTrainerPalette_Brendan gTrainerBackPic_BrendanPal
```

5. Now add this new palette underneath the Old Man's entry in the table *gTrainerBackPicPaletteTable*:

```
[TRAINER_BACK_PIC_OLD MAN] = {gTrainerPalette_OldMan, TRAINER_BACK_PIC_OLD MAN},
[TRAINER_BACK_PIC_BRENDAN_NEW] = {gTrainerPalette_Brendan, TRAINER_BACK_PIC_BRENDAN_NEW},
```

6. Add a line for the animation of the backsprite. The pointer should be **0x8239F44** for 5 frames, and **0x8239F54** for four frames. So since the backsprite has four frames:

```
#define gTrainerBackAnims_Brendan (const union AnimCmd* const*) 0x8239F54
```

7. Search for *gTrainerBackAnimsPtrTable* and add your new entry after the old man.

```
[TRAINER_BACK_PIC_OLD_MAN] = gTrainerBackAnims_OldMan,  
[TRAINER_BACK_PIC_BRENDAN_NEW] = gTrainerBackAnims_Brendan,
```

8. Add an entry in the *gTrainerBackPicCoords* table after the old man:

```
[TRAINER_BACK_PIC_OLD_MAN] =      {.coords = 8, .y_offset = 4},  
[TRAINER_BACK_PIC_BRENDAN_NEW] =  {.coords = 8, .y_offset = 4},
```

9. Next, add a declaration and a table for the image frames like so:

```
extern const u8 gTrainerBackPic_BrendanTiles[];  
static const struct SpriteFrameImage sTrainerBackPicTable_Brendan[] =  
{  
    {gTrainerBackPic_BrendanTiles,          0x800, 0},  
    {gTrainerBackPic_BrendanTiles + 0x0800, 0x800, 0},  
    {gTrainerBackPic_BrendanTiles + 0x1000, 0x800, 0},  
    {gTrainerBackPic_BrendanTiles + 0x1800, 0x800, 0},  
};
```

Format for each name is [FILE_NAME_NO_EXTENSION]Tiles.

10. Finally, search for *gSpriteTemplateTable_TrainerBackSprites* and add an entry for the new backsprite after the old man:

```
[TRAINER_BACK_PIC_OLD_MAN] =  
{  
    .tileTag = 0xFFFF,  
    .paletteTag = 0,  
    .oam = gOamData_TrainerBacksprite,  
    .anim = NULL,  
    .images = gTrainerBackPicTable_OldMan,  
    .affineAnim = gAffineAnim_TrainerBacksprite,  
    .callback = gSpriteCB_TrainerBacksprite,  
},  
[TRAINER_BACK_PIC_BRENDAN_NEW] =  
{  
    .tileTag = 0xFFFF,  
    .paletteTag = 0,  
    .oam = gOamData_TrainerBacksprite,  
    .anim = NULL,  
    .images = sTrainerBackPicTable_Brendan,  
    .affineAnim = gAffineAnim_TrainerBacksprite,  
    .callback = gSpriteCB_TrainerBacksprite,  
},
```

The entry in *.images* should be the same as the table added in step 9.
And you're done!

Battle Terrain

If you have inserted any new battle backgrounds using the tutorial [here](#), then certain modifications will need to be made to make the engine compatible with those backgrounds.

1. Open up the file **include/battle.h**. Search for **BATTLE_TERRAIN_CHAMPION** in the file, and add a new definition there. So, for instance, if your new background was a snow field:

```
#define BATTLE_TERRAIN_LANCE          0x12
#define BATTLE_TERRAIN_CHAMPION      0x13
#define BATTLE_TERRAIN_SNOW_FIELD    0x14
```

2. Open up the file **src/Tables/Terrain_Table.c**. Search for **BATTLE_TERRAIN_CHAMPION** in the file, and add a new entry to the table. So continuing on with the snow example:

```
[BATTLE_TERRAIN_CHAMPION + 4] =
{
    .camouflageType = TYPE_NORMAL,
    .secretPowerEffect = MOVE_EFFECT_PARALYSIS,
    .secretPowerAnim = MOVE_BODYSLAM,
    .naturePowerMove = MOVE_TRIATTACK,
    .burmyForm = SPECIES_BURMY_TRASH,
},

[BATTLE_TERRAIN_SNOW_FIELD + 4] =
{
    .camouflageType = TYPE_ICE,
    .secretPowerEffect = MOVE_EFFECT_FREEZE,
    .secretPowerAnim = MOVE_AVALANCHE,
    .naturePowerMove = MOVE_FROSTBREATH,
    .burmyForm = SPECIES_NONE,
},
```

Notice the entry name is the same as the definition from earlier + 4.

The table has entries for:

- *camouflageType*: The type the move [Camouflage](#) transforms into.
- *secretPowerEffect*: The secondary effect of the move [Secret Power](#).
- *secretPowerAnim*: The animation of the move *Secret Power*.
- *naturePowerMove*: The attack the move [Nature Power](#) becomes.
- *burmyForm*: The form [Burmy](#) transforms into after this battle. If you would not like *Burmy* to change form after this battle, leave it as **SPECIES_NONE** (as shown above).

Battle Music

There are three different music tables that can be set up. Each table can be found in **src/Tables/music_tables.c**. Any custom song definitions used in these tables should be added to **include/constants/songs.h**. Trainer classes can be found and added into the file **include/constants/trainer_classes.h**.

Class Based Encounter Music

If **ENCOUNTER_MUSIC_BY_CLASS** is defined, then the table, *gClassBasedTrainerEncounterBGM*, can be modified to determine which music plays in the background when each trainer class is encountered in the overworld. For example, this table causes all *Youngsters* to have the same encounter music, without having to set the byte for each of them in their trainer data. If you wanted to change which encounter music *Youngsters* have, all you have to do is make the following change:

```
[CLASS_YOUNGSTER] = BGM_EYE_BOY, → [CLASS_YOUNGSTER] = BGM_EYE_GIRL,
```

Now, all *Youngsters* will play the girl encounter when they spot the player.

If you do not want to use this feature, the **switch** statement in **SetUpTrainerEncounterMusic** found in **src/overworld.c** will need to be modified to add new encounter song ids in.

Class Based Battle Music

The table, *gClassBasedBattleBGM*, can be modified to determine the song that plays during trainer battles against certain classes. For example, if I wanted to make all *Team Rocket* battles play the *Gym Leader* theme, I would add the following onto the end of the table:

```
[CLASS_ELITE_4] = BGM_BATTLE_GYM_LEADER,  
[CLASS_TEAM_ROCKET] = BGM_BATTLE_GYM_LEADER,  
};
```

In a *Multi Battle*, if either trainer has custom battle music, their theme will play. If both trainers have custom battle music, then the theme for the trainer on the right (first opponent) will play.

Wild Species Based Battle Music

The table, *gClassBasedTrainerEncounterBGM*, can be modified to determine the song that plays during wild battles against certain species of Pokémon. For example, to make all *Rattata* battles play the *Deoxys* theme, I would add the following onto the end of the table:

```
[SPECIES_DEOXY] = BGM_BATTLE_DEOXY,  
[SPECIES_RATTATA] = BGM_BATTLE_DEOXY,  
};
```

Victory Music

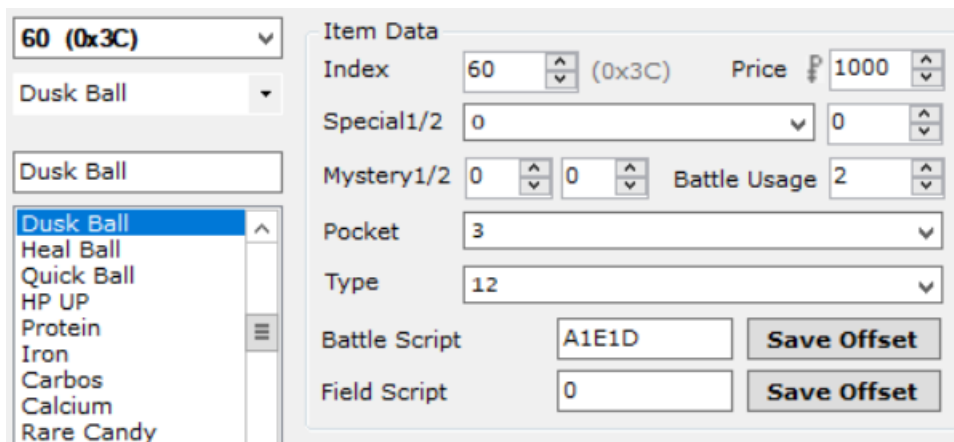
In the file **src/end_battle.c** is a function called **HandleEndTurn_BattleWon**. In this function is a switch statement (look for **VICTORY_MUSIC_SELECTION**) which controls the songs that play when defeating certain trainer classes. See the example from *Pokémon Unbound* (include in the file) for how to append new classes to this switch statement.

Poke Balls

Several new Poké Balls have been added to the engine, in addition to the Trainer Class Based Poké Ball hack.

Adding Support for Added Balls

Although catching data has been added in for the new balls, item data has not. This means that if you want to give the player a certain ball, you'll need to add in item data for it. Adding a new Poké Ball follows the following format in G3T:



The screenshot shows the G3T Item Data editor. On the left, a list of items is shown with 'Dusk Ball' selected. The main panel displays the following fields:

- Index: 60 (0x3C)
- Price: 1000
- Special1/2: 0
- Mystery1/2: 0 0
- Battle Usage: 2
- Pocket: 3
- Type: 12
- Battle Script: A1E1D
- Field Script: 0

Buttons for 'Save Offset' are present next to the Battle Script and Field Script fields.

Regarding the *Type* field, this related to the *Ball Type* of the given ball. So looking in **include/new/catching.h**, it can be seen that *Dusk Ball*'s type is 12.

Class Based Poke Balls

Loosely based on the hack created by [Sagiri](#), if **TRAINER_CLASS_POKE_BALLS** is defined, this implements the feature from Gen 7 where certain Trainer classes always send out Pokémon in a specific type of Poké Ball.

To modify the trainer class balls, open **src/Tables/class_based_poke_ball_table.c**. All the trainer classes have been preloaded into the table, but if you would like to change a trainer class name to your own custom name, do so in **include/constants/trainer_classes.h** and then update the table accordingly. The ball type defines that can be used can be found in **include/new/catching.h**. So, for example, to change the *Youngster*'s ball to a *Great Ball*, make the following change:

```
[CLASS_YOUNGSTER] = BALL_TYPE_POKE_BALL, → [CLASS_YOUNGSTER] = BALL_TYPE_GREAT_BALL,
```

And then the next time the player battles a *Youngster*, all their Pokémon will be sent out in *Great Balls*!

See [Adding New Poke Balls](#) to add new Poke-Ball types.

Trainers With EVs

Loosely based on [DoesntKnowHowToPlay's hack](#), if `TRAINERS_WITH_EVS` is defined, this feature allows trainer Pokémon to have certain EV spreads. Preloaded spreads can be found in `src/Tables/trainers_with_evs_table.h`. New spreads should also be added in here as well.

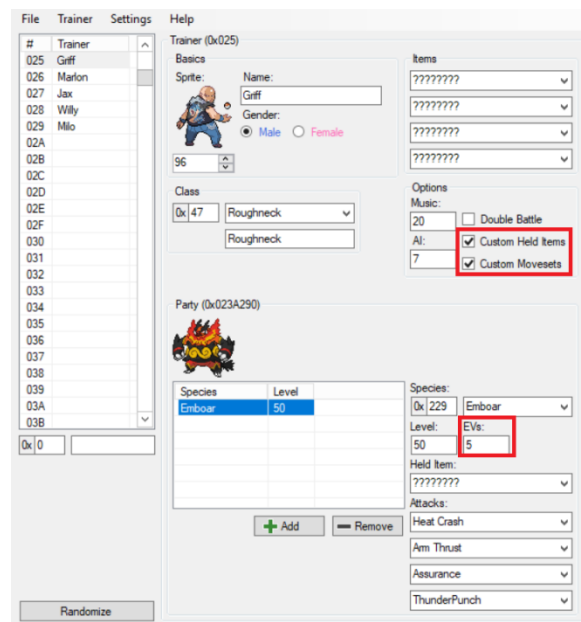
Note that since this is a `.h` file, if you want your changes to compile, you will need to make a change to the file `src/build_pokemon.c` so the compiler recompiles the Trainer's With EVs file. The change can be as simple as adding a whitespace character and then removing it. **Don't use the undo button to remove the change!** The change needs to be removed manually by using the backspace character so the editor saves the file with an updated timestamp.

Assigning a Spread

In order for a Pokémon to use a spread, the following must be done:

- The Pokémon must have a custom moveset.
- The Pokémon must have a custom item (this item can still be set to 0).
- The Pokémon must have an IV (labeled "EVs" in most Trainer editors) value greater than 1 (the 0th spread is left empty for this reason).

For example, looking in the file, you can see that spread 5 is predefined as a "Bulky Physical Attacker". If I wanted to assign this to a Pokémon, the layout would look like this (in [Hopeless Trainer Editor](#)):



As can be seen in the image, both *Custom Movesets* and *Custom Hold Items* have been selected, and the *EVs* value has been set to 5. Note that the hold item has been set to "????????", meaning that *Emboar* is not actually holding an item (this is still fine).

Creating a New Spread

If you want to create a new spread, add it to the end of the table and start counting at 31. The table can at most hold up to the 255th spread. Spreads have the following members:

- `.nature`: The nature of the Pokémon. Natures can be found in **`include/pokemon.h`**.
- `.ivs`: The IVs for the Pokémon. All stats are set to this value (meaning custom *Hidden Powers* are not possible).
- `.hpEv`: The number of *HP* EVs the Pokémon has.
- `.atkEv`: The number of *Attack* EVs the Pokémon has.
- `.defEv`: The number of *Defense* EVs the Pokémon has.
- `.spAtkEv`: The number of *Special Attack* EVs the Pokémon has.
- `.spDefEv`: The number of *Special Defense* EVs the Pokémon has.
- `.spdEv`: The number of *Speed* EVs the Pokémon has.
- `.ball`: If set to `TRAINER_EV_CLASS_BALL` and `TRAINER_CLASS_POKE_BALLS` is defined, then the ball loaded will be the one for the trainer class. Otherwise look in **`include/new/catching.h`** for a list of available Poké Ball types.
- `.ability`: The ability number of the Pokémon. Can be set to:
 - `Ability_1`: The Pokémon will have its first ability.
 - `Ability_2`: The Pokémon will have its second ability.
 - `Ability_Hidden`: The Pokémon will have its [hidden ability](#).
 - `Ability_Random_1_2`: The Pokémon will have one of its primary abilities.
 - `Ability_RandomAll`: The Pokémon will have one of its possible abilities.

Example:

```
[31] =  
{  
    .nature = NATURE_DOCILE,  
    .ivs = 31,  
    .hpEv = 128,  
    .atkEv = 252,  
    .spdEv = 128,  
    .ball = BALL_TYPE_CHERISH_BALL,  
    .ability = Ability_RandomAll,  
},
```

In the above example, the nature for the spread is set to *Docile*, each stat has an IV of 31, the *HP* stat has 128 EVs, the *Attack* stat has 252 EVs, and the *Speed* stat has 128 EVs. All EV stat not referenced in this spread will be set to a default value of 0. Any Pokémon using this spread will also be released in a *Cherish Ball* and have any one of its possible abilities (including its hidden ability).

Battle Frontier

There is support for four unique battle facilities included in this engine: the *Battle Tower*, *Battle Sands*, *Battle Mine*, and *Battle Circus* (names can be changed - see below for descriptions). This is an advanced option and will not be explained in great detail. The various battle facilities and necessary setup will be explained here, but script making will not be detailed. Use the documentation provided for the frontier specials to come up with your own unique battle facility scripts.

The basic setup for any battle facility is to first configure the configuration vars for the Battle Frontier. *Special 0xF5* can be used to select Pokémon to participate in a battle facility, but the Battle Frontier vars must be configured first and `FLAG_BATTLE_FACILITY` must be set prior. Don't forget to use *Special 0x27* to back up the player's party before the battle, and use *Special 0x28* to restore the original party after all the battles have completed. See the example script for *Special 0x73* for an example on how to properly enter the player's Pokémon.

When saving the game, make sure the player has their original team or it will be lost if they rage quit. Once the Pokémon are entered, simply use the trainer generator specials before initiating a battle.

When the player enters a battle facility, the var *0x403A* (a temporary var usually used for elevator scripts) should be set to one of the following values:

- In Battle Tower: 0
- In Battle Sands: 1
- In Battle Mine: 2
- In Battle Circus: 3

This will help the game understand which streaks to load among other things.

Each battle facility has its own set of predefined tiers.

The *Battle Tower* tiers can be modified by editing *gBattleTowerTiers*. Note that *Monotype* (Lv. 100 only here) shares a record with *Little Cup* (Lv. 5 only), so either both or neither must be present in the list. The *Battle Tower* can have at most six tiers (seven if *Monotype* + *Little Cup* are used). The *Battle Sands* uses the same tiers as the *Battle Tower*.

The *Battle Mine* has three formats, each with their own list of tiers: *gBattleMineFormat1Tiers*, *gBattleMineFormat2Tiers*, and *gBattleMineFormat3Tiers*. As many tiers may be added to each format as you'd like.

The *Battle Circus* can have as many tiers added to *gBattleCircusTiers* as you'd like.

Battle Tower

The Battle Tower should be used as the most basic of Battle Facilities. Nothing is special about each battle.

Battle Sands

For all battles, the player's Pokémon is controlled by some AI character. Each battle, a random stat boost is applied to one Pokémon on each side of the field. These stat boosts increase in intensity the higher the win streak. The AI character chosen to control the player's Pokémon should ideally be chosen by using *sp06D_LoadFrontierMultiTrainerById* with the random input.

Battle Mine

Each battle has a random format determined before the battle which randomizes level, party size, inverse settings, and tier (varies depending on Battle Mine format). The selection of options can get more harrowing the higher the win streak. See the function [sp070_RandomizeBattleMineBattleOptions](#) in **src/frontier.c** for the logic of choosing the battle settings. Make sure to call the aforementioned special before battles, as well as call [sp071_LoadBattleMineRecordTier](#) after battles to revert to the original chosen tier (*Battle Mine Format 1, 2, or 3*).

Battle Circus

Each battle has a random permanent effect chosen before battle. These effects increase in intensity and amount per battle the higher the win streak. See the function [sp072_LoadBattleCircusEffects](#) for the logic of choosing the effects. The effects can be:

- Permanent Electric Terrain
- Permanent Grassy Terrain
- Permanent Misty Terrain
- Permanent Psychic Terrain
- Permanent Rain
- Permanent Sun
- Permanent Sandstorm
- Permanent Hail
- Permanent Fog
- Permanent Trick Room
- Permanent Magic Room
- Permanent Wonder Room
- Permanent Gravity
- Permanent Ion Deluge
- Permanent Fairy Lock
- Permanent Mud Sport
- Permanent Water Sport
- Permanent Sea of Fire
- Permanent Rainbow
- Permanent Swamp (basically useless)
- Permanent Confusion
- Permanent Taunt
- Permanent Torment
- Permanent Torment
- Permanent Heal Block
- Permanent Throat Chop
- Permanent Ability Suppression
- Always land critical hits
- Never land critical hits

Trainers

Pre-defined Trainers for the battle frontier can be found in *gTowerTrainers* in the file `src/tables/battle_frontier_trainers.c`. By default they are hidden behind a `#ifdef UNBOUND`, but this can easily be fixed by changing the line to `#ifndef UNBOUND`. Below is an explanation of each member of *struct BattleTowerTrainer*:

- *owNum*: The OW sprite id of the trainer. See `include/constants/event_objects.h`.
- *trainerClass*: The trainer class of the trainer. See `include/constants/trainers.h`.
- *trainerSprite*: The in-battle sprite of the trainer. See `include/constants/trainers.h`.
- *gender*: Either `BATTLE_FACILITY_MALE` or `BATTLE_FACILITY_FEMALE`.
- *preBattleText*: A pointer to a string that is displayed before the battle.
- *playerWinText*: A pointer to a string that is displayed when the player wins.
- *playerLoseText*: A pointer to a string that is displayed when the player loses.

The struct *struct SpecialBattleFrontierTrainer* can be used for both [Frontier Brains](#) and [Special Trainers](#). For the Frontier Brain of the Battle Mine, see the example (Pablo) from *Pokémon Unbound*. An explanation of each member:

- *owNum*: The OW sprite id of the trainer. See `include/constants/event_objects.h`.
- *trainerClass*: The trainer class of the trainer. See `include/constants/trainers.h`.
- *trainerSprite*: The in-battle sprite of the trainer. See `include/constants/trainers.h`.
- *gender*: Either `BATTLE_FACILITY_MALE` or `BATTLE_FACILITY_FEMALE`.
- *name*: A pointer to a string that's the name of the trainer.
- *regularSpreads*: An array of Pokémon spreads for any tier that uses standard Pokémon.
- *legendarySpreads*: An array of Pokémon spreads for any tier that uses legendary (banned in the *Standard* tier and allowed in *Ubers/No Restrictions*) Pokémon.
- *middleCupSpreads*: An array of Pokémon spreads for any tier that uses Pokémon that have evolved once and can still evolve.
- *littleCupSpreads*: An array of Pokémon spreads for any tier that uses Little Cup Pokémon.
- *regSpreadSize*: The number of regular Pokémon spreads.
- *legSpreadSize*: The number of legendary Pokémon spreads.
- *mcSpreadSize*: The number of Middle Cup spreads.
- *lcSpreadSize*: The number of Little Cup spreads.

The struct *struct MultiBattleTowerTrainer* is for trainers that can be teamed up with in *Multi Battles*. Their struct is very similar to the *Special Trainer* struct, with one key member addition:

- *otId*: The trainer Id assigned to each of the multi trainer's Pokémon.

Other than that one difference, this struct is set up in almost the same way (see the example from *Pokémon Unbound* for more details).

Any trainer text should be defined in `strings/frontier_trainer_text.string`.

Any trainer names should be defined in `strings/frontier_trainer_names.string`.

Spreads

What would a Battle Frontier be without Pokémon to battle against? In the file **src/tables/battle_tower_spreads.h** (linked to the compilation of **src/build_pokemon.c**) there are several tables of generic Pokémon spreads (each is self-explanatory by its name). By default they are hidden behind a **#ifdef UNBOUND**, but this can easily be fixed by changing the line to **#ifndef UNBOUND**.

Most of the members of *struct BattleTowerSpread* are very self-explanatory, but there are a few members that should be explained:

- *ball*: The Poké Ball the Pokémon is in. See **include/new/catching.h** for ball types.
- *shiny*: **TRUE** if the Pokémon is shiny. **FALSE** or don't include the line if it is not.
- *forSingles*: The Pokémon can be generated in a single battle.
- *forDoubles*: The Pokémon can be generated in a double battle.
- *modifyMovesDoubles*: Certain moves of the Pokémon will change in a double battle (eg. *Stone Edge* becomes *Rock Slide*).

Special spreads for *Special Trainers* and *Frontier Brains* should be defined in the very, very large file **src/tables/frontier_special_trainer_spreads.h**. (linked to the compilation of **src/tables/battle_frontier_trainers.c**). When designing teams for these trainers, take care to always make sure a team of 6 with unique items, unique species, and no Mega Pokémon can be created in any tier the spread set is usable in.

Upgraded TM/HM Expansion

This engine includes a vastly improved TM/HM system over those created in the past. This version allows the hacker to use ANY item id for their TMs. None of this item ID sorting or item gap nonsense! It also fixes the graphical problems with high TM numbers in the TM case.

To set this feature up, set the *Mystery 2* byte of the item to the TM Id, starting at 01 for TM01. The HMs must start at a TM Id **after** the max number of TMs you plan on including. For example, if you want 120 TMs and 8 HMs, HM01 would be given a TM Id of 121. Otherwise, the items are set up as normal (see image ->). The item sprite and palette don't matter, as the disk is loaded outside of the item data, and the TM animation must be removed for the expansion to work correctly.

This engine does not mess with any of the original tables to allow a hack in progress to keep all its data. Therefore, once all your item data is set up, you must set up the *TM Move Table* and *TM Compatibility Table*. The *TM Move Table* has a pointer at 0x125A8C and is simply a list of move IDs corresponding with the TM ID. The *TM Compatibility Table* is more complicated. Its pointer lies at 0x43C68 and is 16 bytes per species. It is explained how to modify it in *Step 3* from [this tutorial](#).

The screenshot shows a software interface for configuring TM/HM data. On the left is a scrollable list of item names, with 'TM52' selected. On the right is a configuration panel for the selected item. The 'Item Data' section includes fields for Index (227), Price (0), Special1/2 (0), Mystery1/2 (0), Battle Usage (0), Pocket (4 TM Case), Type (1 Out of battle), Battle Script (0), and Field Script (0). There are 'Save Offset' buttons for Battle Script and Field Script. The 'Description' section has a text field containing '3DB020' and a 'Save Offset' button. The 'Unknown Bytes' section has two input fields set to 0. The 'Extra ?' section has an input field set to 00000000 and a 'Save Extra' button. The 'Sprite' section has a question mark icon, two input fields set to E87028 and E870A0, and 'Save Sprite' and 'Save Palette' buttons. At the bottom is a 'TM/HM Move Name' dropdown menu.

If you're using the *Dynamic Pokémon Expansion*, simply modify the provided TM files to add/remove any TM data. The compiler will take care of the rest! Make sure you read the readme on how to set up the TM data first, though!









































Reusable TMs

If you've defined *REUSABLE_TMS* in **src/config.h**, you will then also need to set the *Mystery 1* byte to 1 for each of your TMs. This will tell the game to treat the item like an HM (except extra code has been written to sort TMs/HMs by the TM index. For more info see [TM/HM Expansion](#)), preventing you from giving it to a Pokemon to hold, selling it, etc. That's it!

Pickup³

The items found by the ability [Pickup](#) can be modified in `src/Tables/item_tables.c`. Modify *sPickupCommonItems* and *sPickupRareItems* to change the items that appear. **DO NOT** add any new items to the tables; only change the pre-existing items. The default values were chosen based on the following table (common is highlighted in blue, rare is highlighted in red):

Pokémon Omega Ruby and Alpha Sapphire [\[edit\]](#)

Level	30%	10%	4%	1%
1-10				
11-20				
21-30				
31-40				
41-50				
51-60				
61-70				
71-80				
81-90				
91-100				

Select from PC Hack

If `SELECT_FROM_PC` is defined, the player can select Pokémon directly from the PC to modify certain data. To initiate, set *pcSelect_StateTracker* (defined in `asm_defines.s`) to 1 before using Special 0x3C. The hack will store the box number and slot to *Var 0x8000* and *Var 0x8001*, respectively. All data retrieval/manipulation specials will be able to access the selected PC Pokémon by setting *Var 0x8003* to 1. For example, nicknaming a Pokémon in the PC:

```
#define pcSelect_StateTracker 0x203B7AC

#org @start
writebytetooffset 0x1 pcSelect_StateTracker
special 0x3C 'Select boxed mon, box stored to Var8000, slot to Var8001
waitstate
writebytetooffset 0x0 pcSelect_StateTracker
compare LASTRESULT 0x7F
if 0x1 goto @NothingSelected 'User cancelled out of the PC menu
bufferpokemon 0x0 0x8002
setvar 0x8003 0x1 'Data source is in the PC Box
special 0x7d
compare LASTRESULT 0x1
if 0x1 goto @traded
special 0x9e
waitstate
end
```

³ Credits to [Sagiri](#) for the original code.

Time of Day Based Wild Encounters

In order to use this feature, `TIME_ENABLED` must be defined. Once it is open the file `src/Tables/wild_encounter_tables.c`. Find where it says `//Modify this section`. This is the data that will be modified.

Example: Modifying the Night Data

For this example, HOOTHOOT will be added onto ROUTE 1 in place of PIDGEY.

First, the wild data needs to be created. If you open up AdvanceMap to the wild data for ROUTE 1, you should see this:

Type:

Encounter ratio: 8%

	Min/max level:	Pokémon:	No:	Chance:
Pokémon 1:	3 3	PIDGEY	16	20%
Pokémon 2:	3 3	RATTATA	19	20%
Pokémon 3:	3 3	PIDGEY	16	10%
Pokémon 4:	3 3	RATTATA	19	10%
Pokémon 5:	2 2	PIDGEY	16	10%
Pokémon 6:	2 2	RATTATA	19	10%
Pokémon 7:	3 3	PIDGEY	16	5%
Pokémon 8:	3 3	RATTATA	19	5%
Pokémon 9:	4 4	PIDGEY	16	4%
Pokémon 10:	4 4	RATTATA	19	4%
Pokémon 11:	5 5	PIDGEY	16	1%
Pokémon 12:	4 4	RATTATA	19	1%

Now that you know what the data looks like, convert it into a C structure (see image below) and paste it at the top file, directly under the line `"#ifndef UNBOUND //Modify this section"`, or under previously created wild data structures. No matter what, it must be placed above the line `"const struct WildPokemonHeader gWildMonMorningHeaders[] ="`:

```
const struct WildPokemon gRoute1_LandMonsNight[] =
{
    {3, 3, PKMN_PIDGEY},
    {3, 3, PKMN_RATTATA},
    {3, 3, PKMN_PIDGEY},
    {3, 3, PKMN_RATTATA},
    {2, 2, PKMN_PIDGEY},
    {2, 2, PKMN_RATTATA},
    {3, 3, PKMN_PIDGEY},
    {3, 3, PKMN_RATTATA},
    {4, 4, PKMN_PIDGEY},
    {4, 4, PKMN_RATTATA},
    {5, 5, PKMN_PIDGEY},
    {4, 4, PKMN_RATTATA},
};

const struct WildPokemonInfo gRoute1_LandMonsInfoNight = {21, gRoute1_LandMonsNight};
```

^ This line is very important and must be added in as well. Make sure the label correctly matches the name for your newly created wild data structure. The 21 is the encounter rate.

Now that the wild data has been copied from AdvanceMap, it's time to make modifications. Change all the "PKMN_PIDGEY" to "PKMN_HOOTHOOOT":

```
const struct WildPokemon gRoute1_LandMonsNight[] =
{
    {3, 3, PKMN_HOOTHOOOT},
    {3, 3, PKMN_RATTATA},
    {3, 3, PKMN_HOOTHOOOT},
    {3, 3, PKMN_RATTATA},
    {2, 2, PKMN_HOOTHOOOT},
    {2, 2, PKMN_RATTATA},
    {3, 3, PKMN_HOOTHOOOT},
    {3, 3, PKMN_RATTATA},
    {4, 4, PKMN_HOOTHOOOT},
    {4, 4, PKMN_RATTATA},
    {5, 5, PKMN_HOOTHOOOT},
    {4, 4, PKMN_RATTATA},
};

const struct WildPokemonInfo gRoute1_LandMonsInfoNight = {21, gRoute1_LandMonsNight};
```

The data for Route 1 at night is now complete. As this is data for night time, we need to add it to our night table, *gWildMonNightHeaders*. Make sure you leave the pre-existing entry at the bottom of the table:

```
const struct WildPokemonHeader gWildMonNightHeaders[] =
{
    {
        .mapGroup = MAP_GROUP(ROUTE_1),
        .mapNum = MAP_NUM(ROUTE_1),
        .landMonsInfo = &gRoute1_LandMonsInfoNight,
        .waterMonsInfo = NULL,
        .rockSmashMonsInfo = NULL,
        .fishingMonsInfo = NULL,
    },
    {
        .mapGroup = 0xFF,
        .mapNum = 0xFF,
        .landMonsInfo = NULL,
        .waterMonsInfo = NULL,
        .rockSmashMonsInfo = NULL,
        .fishingMonsInfo = NULL,
    }
};
```

The only thing left to do now is define "MAP_ROUTE_1". Go back to AdvanceMap and find the map bank and map number for ROUTE 1 (the map bank is 3, and the map number is 19). In **include/constants/maps.h**, add a line formatted like (if the define already exists edit the line):

```
#define MAP_NAME ((MAP_BANK << 8) | MAP_NUM)
#define MAP_ROUTE_1 ((3 << 8) | 19)
```

Make sure the map name matches what's in the brackets for ".mapGroup" and ".mapNum" (ie. ROUTE_1 became MAP_ROUTE_1).

Now wild night data has successfully been added for Route 1. Morning and Evening data follow the same pattern. Any route that doesn't have morning or night data defined will load the standard day data set in AdvanceMap.

For water, fishing, or Rock Smash data, follow the same steps, but look [here](#) to see how to structure those kinds of wild datasets.

If you followed everything correctly, here is what the file should look like now:

```
#include "..\\defines.h"

#ifdef UNBOUND //Modify this section

#define MAP_ROUTE_1 ((3 << 8) | 19)

const struct WildPokemon gRoute1_LandMonsNight[] =
{
    {3, 3, PKMN_HOOTHOOT},
    {3, 3, PKMN_RATTATA},
    {3, 3, PKMN_HOOTHOOT},
    {3, 3, PKMN_RATTATA},
    {2, 2, PKMN_HOOTHOOT},
    {2, 2, PKMN_RATTATA},
    {3, 3, PKMN_HOOTHOOT},
    {3, 3, PKMN_RATTATA},
    {4, 4, PKMN_HOOTHOOT},
    {4, 4, PKMN_RATTATA},
    {5, 5, PKMN_HOOTHOOT},
    {4, 4, PKMN_RATTATA},
};

const struct WildPokemonInfo gRoute1_LandMonsInfoNight = {21, gRoute1_LandMonsNight};

const struct WildPokemonHeader gWildMonMorningHeaders[] =
{
    {
        .mapGroup = 0xFF,
        .mapNum = 0xFF,
        .landMonsInfo = NULL,
        .waterMonsInfo = NULL,
        .rockSmashMonsInfo = NULL,
        .fishingMonsInfo = NULL,
    }
};

const struct WildPokemonHeader gWildMonEveningHeaders[] =
{
    {
        .mapGroup = 0xFF,
        .mapNum = 0xFF,
        .landMonsInfo = NULL,
        .waterMonsInfo = NULL,
        .rockSmashMonsInfo = NULL,
        .fishingMonsInfo = NULL,
    }
};

const struct WildPokemonHeader gWildMonNightHeaders[] =
{
    {
        .mapGroup = MAP_GROUP(ROUTE_1),
        .mapNum = MAP_NUM(ROUTE_1),
        .landMonsInfo = &gRoute1_LandMonsInfoNight,
        .waterMonsInfo = NULL,
        .rockSmashMonsInfo = NULL,
        .fishingMonsInfo = NULL,
    },
    {
        .mapGroup = 0xFF,
        .mapNum = 0xFF,
        .landMonsInfo = NULL,
        .waterMonsInfo = NULL,
        .rockSmashMonsInfo = NULL,
        .fishingMonsInfo = NULL,
    }
};

const struct SwarmData gSwarmTable[] =
{
    {
        .mapName = 0xFF,
        .species = 0xFFFF,
    },
};
```

Swarms

Also known as “[mass outbreaks](#)”, swarms are when a certain species of Pokémon that doesn’t normally appear in the wild, makes an appearance for a single day. To get the most out of this feature, the real-time clock is utilized to run a function daily that updates the swarms. To add a swarm, in the file **src/Tables/wild_encounter_tables.c**, search for the first occurrence of *gSwarmTable*. Each element in this table is a *struct* with the following members:

- **mapName**: The map name (or section) id where the swarming Pokémon will be located.
- **species**: The species to swarm in locations with the given mapName.

For example, to make *Sentret* swarm on *Route 1*, add the following to the table (the levels are loaded from *Route 1*’s wild data):

```
{
    .mapName = MAP_NAME_ROUTE_1,
    .species = SPECIES_SENTRET,
}
```

The species names can be found in **include/constants/species.h**. The map names are not predefined anywhere (use AdvanceMap 1.92’s world map editor as a map name reference), so before this entry is added, the following line needs to be added above the table (or in an external, included file):

```
#define MAP_NAME_ROUTE_1 0x65
```

To buffer swarm text so an NPC can tell the player what and where a Pokémon is swarming, see *special 0x58*.

Roaming Pokemon

[Roaming Pokémon](#) have been staples in the Pokémon games since Gen 2. This engine allows you have up to 10 roaming Pokémon at once, where, if one is encountered, can be trapped, battled, and captured by the player. To set up Pokémon that roam around, use *special 0x129* (see the example script for *special 0x59*).

In order to control where Pokémon can roam, open up **src/roamer.c** and look for *sRoamerLocations*. The roaming movement is divided into several sets of maps with each set containing at most seven different maps (each row in the table is a set). The different maps are and should be defined in **include/constants/maps.h**. Every time the player changes maps, the roamer moves as well. Most likely they will move within a given set, but they may change sets entirely. It’s recommended to only place closely related maps within each set. So, for example, if you had two regions, you’d want one set containing only maps from region A, and another set containing only maps from region B. As many sets as you’d like can be added to the table, however the table must be terminated with:

```
{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
```

Day & Night System

A brand new DNS has been included in the engine. It features dynamic palette fading throughout the day, as well as options to allow windows to light up at night. Both of these options can be customized by editing `include/new/dns_data.h`. Note that since this is a `.h` file, if you want your changes to compile, you will need to make a change to the file `src/dns.c` so the compiler recompiles the DNS file. The change can be as simple as adding a whitespace character and then removing it. **Don't use the undo button to remove the change!** The change needs to be removed manually by using the backspace character so the editor saves the file with an updated timestamp.

Changing Which Palettes Are Faded

Open up the `dns_data.h` file and search for `OW_BG_PAL_0`. Here is a listing of all palettes that can be faded. `OW_BG_PAL_0` through `OW_BG_PAL_15` represent the palettes of the background in the overworld. By default, OW BG palettes 0 - 12 are set to be faded, but 12 can be unfaded by making the following change:

```
#define OW_BG_PAL_12 TRUE → #define OW_BG_PAL_12 FALSE
```

Similarly, whether any other palette is faded can be changed by changing its value from `TRUE` to `FALSE` or vice versa. It is **NOT** recommended to fade palettes 13-15 as these are the colours used for the menus and text boxes.

Other than the OW backgrounds, the `OW_SPRITE_PAL` represent the sprites in the overworld, the `BBG_PAL` defines represent the background in battle, and the `B_SPRITE_PAL` defines represent the sprites in battle.

Changing the Colours Faded Throughout the Day

In the same file is a table representing the actual fading colours (*gDNSNightFadingByTime*). Currently the table only has entries from 12 AM - 7:59 AM and 5 PM - 11:59 PM. The rest of the day no fading changes can be seen. If you would like to add fading for more time during the day, simply look for the line *Day has no fade* and start adding new entries there. For example, adding an entry for 8 AM - 8:59 AM:

```
[8] = {
    {RGB(0, 24, 16), 1}, //8:00 AM
    {RGB(0, 24, 16), 1}, //8:10 AM
    {RGB(0, 24, 16), 1}, //8:20 AM
    {RGB(0, 24, 16), 1}, //8:30 AM
    {RGB(0, 24, 16), 1}, //8:40 AM
    {RGB(0, 24, 16), 1}, //8:50 AM
},
```

The colour can change every ten minutes, so each of those colour indices represent a colour at that ten minute period.

Light Up Windows

There are currently two ways to handle light up windows. The first is more tedious. It involves leaving `OW_BG_PAL_12` as `FALSE` (see *Changing Which Palettes Are Faded*) and make all tiles you'd like to light up use that palette. Then, use an on-entry script in AdvanceMap to do setmaptiles and place your light up windows only when the time is night. This can be an extreme annoyance and a time consumer, which is why the second, new method was developed.


This new method uses a table to fade certain palette colours if it nighttime.

1. To start off, find the offset of the tileset containing the palette you wish to fade. For this example, I'll be fading the windows of the player's door in *Palette Town*. Looking in AdvanceMap, the player's door uses tiles from *Tileset 1*, which has an offset of `0x82D4AAC`.

Map footer:
18 00 00 00 14 00 00 00 F8 D0 2D 08 00 D1 2D 08
94 4A 2D 08 AC 4A 2D 08 02 02 00 00

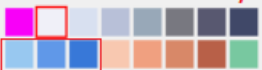
Width:	<input type="text" value="24"/>	Height:	<input type="text" value="20"/>
Border offset:	<input type="text" value="\$002DD0F8"/>	Map offset:	<input type="text" value="\$002DD100"/>
No of tileset part 1:	<input type="text" value="0"/>	No of tileset part 2:	<input type="text" value="1"/>
Offset of tileset part 1:	<input type="text" value="\$002D4A94"/>	Offset of tileset part 2:	<input type="text" value="\$002D4AAC"/>
Border width:	<input type="text" value="2"/>	Border height:	<input type="text" value="2"/>

2. Now open the tile viewer and determine which colours are the ones you wish to fade. For my example, I've determined that the player's door uses palette 8, and the colours used are in indices 8, 9, and 10:

Down/Up  Save

Palette form ✕

Palette: Palette 8 ▾ Apply

 Select color

8 9 10

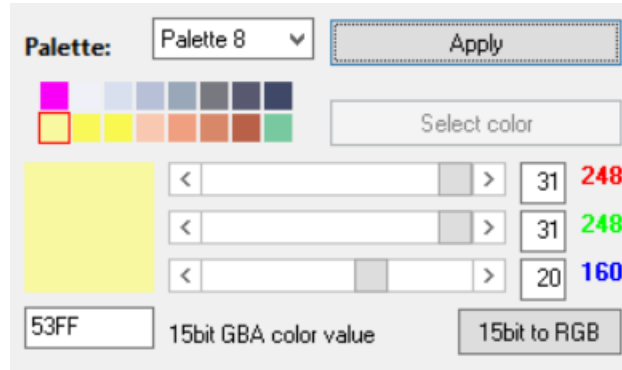
< > 240

< > 240

< > 248

7FDE 15bit GBA color value 15bit to RGB

- Now determine the colours you wish to change them to. It's okay to play around with these colours in AdvanceMap. Hitting *Apply* won't save anything permanently unless you exit out of the block editor and choose to save. For my example, I could go to change the window to a yellowy colour:



These colours can be represented by the RGB values of (31, 31, 20), (31, 31, 11), and (31, 31, 10).

- Putting it all together now:

```
{ //Palette Town - Player's Door
  .tilesetPointer = 0x82D4AAC, //Tileset 1
  .paletteNumToFade = 8,
  .paletteIndicesToFade =
  {
    {8, RGB(31, 31, 20)},
    {9, RGB(31, 31, 11)},
    {10, RGB(31, 31, 10)},
    TILESET_PAL_FADE_END
  },
},
```

In the image *.tilesetPointer* is set to the offset of the tileset found in step 1, *.paletteNumToFade* is set to 8 representing the 8th palette. *.paletteIndicesToFade* has entries for index 8, 9, and 10, each with the corresponding colour they should be faded. After all that, is the line **TILESET_PAL_FADE_END**. **DO NOT FORGET THIS LINE!**

Some more examples have been left in the file to help guide you further if you need it.



Pre-Battle Mugshots

The battle transition from the Elite Four has been modified for Fire Red. [Originally created by Jambo51](#), the updated feature allows toggling of different transition styles with vars, as well as use custom images in place of the trainer's front sprite. To use this feature, you must define var values for `VAR_PRE_BATTLE_MUGSHOT_STYLE` and `VAR_PRE_BATTLE_MUGSHOT_SPRITE`. The following table details the types of configurations you can use with these vars.

<code>VAR_PRE_BATTLE_MUGSHOT_STYLE</code>	
0	One Big Bar. Full Trainer Sprites
1	One Small Bar. Half Trainer Sprites
2	Two Small Bars. Half Trainer Sprites

<code>VAR_PRE_BATTLE_MUGSHOT_SPRITE</code>	
0	Player's Front Picture
1	"VS" Symbol instead of Player

To load a custom image instead of the opponent's trainer front sprite, set the flag `FLAG_LOAD_MUGSHOT_SPRITE_FROM_TABLE`. The table `sPreBattleMugshotSprites` in `src/Tables/Mugshot_Tables.c` allows you to set the custom image to load for that given trainer's front sprite index. Two examples exist to show you how to add one in for a different trainer picture.

As per Jambo's original routine, the unused 3rd argument in the trainerbattle script command lets you turn on the pre-battle mugshot. Changing the value to `0x0Y00` lets you select different palettes for each transition tilemap (for a total of 15 different palettes). The palettes associated with each can be modified by changing the palettes of the images found in the directories `graphics/Battle_Mugshots/Big` and `graphics/Battle_Mugshots/DP`.

Finally, if you are using the Two Bar mugshot style, the 3rd trainerbattle argument only changes the palette of the opponent's tilemap. The value of `VAR_MUGSHOT_PLAYER_PAL` can be changed to load a different palette for the player's side as well.

Example Script:

```
#define VAR_PRE_BATTLE_MUGSHOT_STYLE 0x5038
#define VAR_PRE_BATTLE_MUGSHOT_SPRITE 0x5039
```

```
#org @RivalBattle
msgbox @BattleMe 0x6
setvar VAR_PRE_BATTLE_MUGSHOT_STYLE 0x2
setvar VAR_PRE_BATTLE_MUGSHOT_SPRITE 0x0
trainerbattle 0x3 0x146 0x100 @LossMessage
end
```



New Field Moves

The engine allows you to (relatively) easily add new field moves! Rock Climb, Dive, and Defog are already added for you. Note that this feature may require some coding on your behalf.

If you want to only adjust the added field moves, you can adjust the badge requirement in the table *gFieldMoveBadgeRequirements* in **src/party_menu.c** (0 means no badge requirement).

Adding New Field Moves

1. Open up **src/party_menu.c** and find the section “Field Moves”
2. Add your field move to the *enum FieldMoveIDs* section (always include **FIELD_MOVE_COUNT** at the end!). For example, let’s add **FIELD_MOVE_SUNNY_DAY**.
3. Add your field move to the list of party menu options, *gPartyMenuCursorOptions* as so:

```
[MENU_FIELD_MOVES + FIELD_MOVE_SUNNY_DAY] = {gMoveNames[MOVE_SUNNY_DAY], CursorCb_FieldMove},
```

4. Create the appropriate callback for your field move inside *gFieldMoveCursorCallbacks*:

```
[FIELD_MOVE_SUNNY_DAY] = {SetUpFieldMove_SunnyDay, 0x0D},
```

Where **SetUpFieldMove_SunnyDay** is a function you will have to create (see step 8).

5. Add the field move description to *gFieldMoveDescriptions*:

```
[FIELD_MOVE_SUNNY_DAY] = gText_FieldMoveDesc_SunnyDay,
```

Where *gText_FieldMoveDesc_SunnyDay* can be defined in **strings/party_menu.string** and declared at the top of the “Field Moves” section of **src/party_menu.c** like:

```
extern const u8 gText_FieldMoveDesc_SunnyDay[];
```

6. Link the field move and move ID inside *gFieldMoves* (keep **FIELD_MOVE_COUNT** at the end!)

```
[FIELD_MOVE_SUNNY_DAY] = MOVE_SUNNY_DAY,
```

7. Add the badge requirement of your new field move to *gFieldMoveBadgeRequirements*:

```
[FIELD_MOVE_SUNNY_DAY] = 6 // Requires badge 6 to use, for example
```

8. Now we must include the logic for seeing if we can use our field move! This is dependent on what the new field move is. Ostensibly, Sunny Day can only be used outside, which can be checked the same way as fly, for our intents and purposes. We might also want to make sure it isn’t already sunny via **GetCurrentWeather**. So, our function might look like (feel free to adjust):

```

static bool8 SetUpFieldMove_SunnyDay(void)
{
    if (GetCurrentWeather() == WEATHER_SUNNY)
        return FALSE;
    else if (Overworld_MapTypeAllowsTeleportAndFly(gMapHeader.mapType))
    {
        gFieldCallback2 = FieldCallback_PrepareFadeInFromMenu;
        gPostMenuFieldCallback = FieldCallback_SunnyDay;
        return TRUE;
    }
    return FALSE;
}

```

These functions return TRUE if the field move is usable and FALSE if not. Most also set up some callback to run a script upon exiting the party menu. In our case we want to run `FieldCallback_SunnyDay`, which we can emulate from `FieldCallback_RockClimb` as so:

```

static void FieldCallback_SunnyDay(void)
{
    ((u32*) gFieldEffectArguments)[0] = GetCursorSelectionMonId();
    ScriptContext1_SetupScript(SystemScript_SunnyDay);
}

```

We will then have to create our script of `SystemScript_SunnyDay`, either in XSE or in the CFRU engine via **assembly/overworld_scripts/system_scripts.s** (see Step 9) and declare it above with either:

Compiled with XSE: `#define SystemScript_SunnyDay (const u8*) 0x8XXXXXXX` (0x8XXXXXXX is a pointer to where the script was compiled)

Compiled in CFRU: `extern const u8 SystemScript_SunnyDay[];`

9. Our script will only run if the field move is usable, so we don't have to worry about adding a ton of logic into our script. For *Sunny Day*, all we need to do is display the HM bar animation and then set the new weather, like so:

```

.global SystemScript_SunnyDay
SystemScript_SunnyDay:
    lockall
    doanimation 0x28
    waitstate
    setweather 0x1
    doweather
    msgbox gText_BecameSunny MSG_KEEPOPEN @define this in strings/overworld_strings.string
    releaseall
    end

```

And that's all!

Other Features Included

Save Expansion⁴

The default save space has been expanded tremendously to allow for:

- 4096 new Flags (0x900 - 0x18FF)
- 200 new Vars (0x5000 - 0x51FF)
- 10 new PC Boxes (for a total of 24)
- Up to 10 Roaming Pokémon
- Up to 778 unique items in the bag
- 4 New Step Counters

Updated & New Item Effects

Several new items effects have been added such as effects for the [Power Items](#) and the [Shiny Charm](#), a complete set of battle effects, as well as some other items like the [Black Flute](#) and [White Flute](#) (which have been updated to the standards from ORAS).

Trainer Face Fix

The player will face trainers before battle.

DexNav⁵

A simplified [DexNav](#) system for Fire Red is included in the engine. It replaces the *POKEDEX* option in the Start Menu with *TOOLS*, which can either load a multichoice menu containing the *Pokédex* and *DexNav*, or an entirely new start menu. Selecting the *DexNav* opens the graphical user interface (GUI) to allow the player to view which Pokémon they have caught/seen on the current map, and either press the *R-Button* or *A-Button* on a given Pokémon to search for it on the map. The *R-button* will save the Pokémon information to `VAR_DEXNAV` and allow the player to search for that Pokémon via the *R-button* from the overworld.

Dynamic Overworld Palettes⁶

Overworld sprites are now loaded in dynamically, allowing for more freedom with creating new overworld sprite palettes.

Ability Pop-Ups⁷

Whenever a Pokémon's ability activates in battle, a pop-up will appear showing the ability akin the Gen 5+ games.

⁴ Credits to [FBI](#) for this feature.

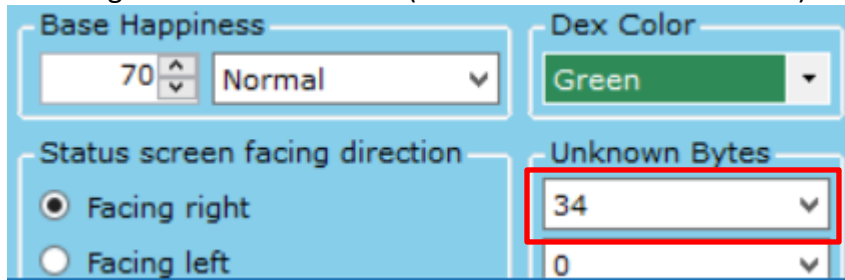
⁵ Credits to [FBI](#) and the [Community Hack](#) for this feature.

⁶ Credits to [Navenatox](#) for this feature.

⁷ Credits to [DizzyEgg](#) for this feature.

Hidden Abilities⁸

Hidden abilities have been implemented using one of the empty bytes in the Pokémon data. It is the 27th byte (0x1A from the start, including 0) of the pokemon base stats data structure. You can assign it in G3T as follows: (in decimal! Not hexadecimal!) see [include/constants/abilities.h](#)



Expanded Text Names

The following text names can be expanded by replacing the first four characters in the name with a pointer to a longer text string:

- Trainer Class Names
- Item Names

Additionally, ability names have had their name length changed from 12 to 16.

Pokédex Screen Stats⁹

The boring old size comparison when viewing a Pokémon's pokedex data has been replaced with a routine by that displays the Pokémon's stats and abilities instead.

Turbo Boost

A pipelined routine has been written to replace the old main loop with one that now allows the game to run at over 1000% speed using the fast-forward button.

Various Customizable Updates

The file **bytereplacement** contains a list of optional byte changes:

- Don't count eggs while healing at the Pokémon Centre.
- Extend number of direct sound tracks to 12.
- Fix for movement type 0xC (Hidden).
- Remove *National Dex* evolution limiters.
- Remove plot related trade restrictions.
- Fix for *Pokédex* species name issue.
- Display foreign Pokémon's id without *National Dex*.

⁸ Credits to [azurile13](#) for this feature.

⁹ Credits to [DoesntKnowHowToPlay](#) and [Squeetz](#) for this feature.

¹⁰Credits to [Diegoisawesome](#) for this feature.

- PC boxes use more wallpapers.
- Max money increased to 9999999.

To remove any of these changes, simply add a # symbol at the beginning of the line or delete the byte change lines entirely.

Triple Layer Tiles¹⁰

Taken from [this source](#), blocks can now have three layers on top of each other. See the original post for details on using this feature.

Expanded Coins

The expanded coins option also comes with rewritten coin-related scripting commands. The original scripting commands only allow an argument of up to *0xFFFF*, but if we would've expanded coins up to 999,999,999, the command would need to be repeated multiple times. Now, using the coin commands (*checkcoins*, *givecoins*, *removecoins*) with argument *0xFFFF*, the value will be loaded from *Var8000* and *Var8001*. The upper halfword will be the value in *Var8000*, and the lower halfword will load from *Var8001*.

Example Script:

```
#org @giveLotsOfCoins
lock
setvar 0x8000 0x1234
setvar 0x8001 0x5678
givecoins 0xFFFF 'The player will receive 0x12345678 (or 305,419,896) coins
release
end
```

Multichoice Windows

Static Multichoice Windows

Like in JPANs Fire Red Hacked Engine, it is possible to script up custom multichoice boxes for players to select from various options. The hack uses [Special 0x24](#) or [Special 0x25](#) to add list items, and the *multichoice* scripting command to display the list. The first two arguments in this command are the (x, y) coordinates on the screen. The third argument is the number of list items you have (0x20 is 2 options, 0x25 is 7 options, the maximum). The final argument is either 0 if B allows the player to cancel, or 1 if the player cannot cancel with the B button. Here is a scripting example using special 0x25:

```
#org @StaticMultichoiceExample
setvar 0x8006 0x0 'First list item
loadpointer 0x0 @text1
special 0x25
setvar 0x8006 0x1 '2nd list item
loadpointer 0x0 @text2
special 0x25
setvar 0x8006 0x2 '3rd list item
loadpointer 0x0 @text3
special 0x25
preparemsg @msg 'String to display in the message box
waitmsg
multichoice 0x0 0x0 0x21 0x1 'Multichoice at (0x0), 3 options, 'B' CANNOT cancel.
compare LAST_RESULT 0x0
if 0x1 goto @firstOption
'etc...
```

Scrolling Multichoice Windows

Ported from [FBIs asm routine](#) and improved slightly, this feature allows you to include multichoice boxes with more than 6 options at once, such as in the example to the right. To set these up, go to the bottom of **src/scripting.c** (or jump there by searching “Scrolling Multichoice”). You can define new multichoice lists by adding `static const u8* sMultichoiceSet3[] = {...}`, where the brackets are filled with pointers to the text you want to add. You can include strings inside the engine by emulating the example strings from the string file **strings/scrolling_multichoice.string**, eg.

```
#org @sExampleText_11
```

Example 11

And then add `extern const u8 sExampleText_11[];` underneath *Text Declarations* in **src/scripting.c**.

Next, you must add the pointer to your new multichoice list to the set of multichoice lists, *gScrollingSets*, defined below the multichoice sets. You can add your new list in the same format as the examples given: `{sMultichoiceSet3, ARRAY_COUNT(sMultichoiceSet3)}`,

Finally, here’s how you would display your new list in a script:

```
#org @ScrollingMultichoiceExample
```

```
setvar 0x8000 0x2 'Choose the 3rd multichoice list in gScrollingSets
```

```
setvar 0x8001 0x5 'Display 5 text items at once (defines the box height - not total num)
```

```
special 0x158
```

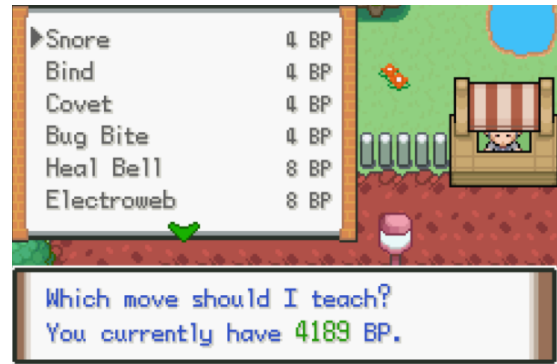
```
waitstate
```

```
compare LAST_RESULT 0x7F
```

```
if 0x1 goto @cancelled
```

```
compare LAST_RESULT 0x0
```

```
if 0x1 goto @firstOption
```



Tile Interaction Scripts

sMetatileInteractionScripts, located in **src/overworld.c** contains scripts linked to tile behaviours (what the tile does when the A-button is pressed next to it).

1. You can add new interaction scripts by defining/using the metatile behaviour byte in **include/constants/metatile_behaviors.h** (eg. a library book - start with 0xA7).

```
#define MB_CYCLING_ROAD_PULL_DOWN_GRASS 0xD1
```

```
#define MB_LIBRARY_BOOK 0xA7
```

2. Link it to a script inside *sMetatileInteractionScripts* via:

```
[MB_LIBRARY_BOOK] = EventScript_LibraryBook,
```
3. Make the script inside **assembly/overworld_scripts/system_scripts.s**. Don’t forget to declare the script above the table and add the needed string to **strings/overworld_strings.string**!

```
.global EventScript_LibraryBook
EventScript_LibraryBook:
    msgbox gText_BookText MSG_SIGN
end
```

Script Specials

Several new scripting specials have been added to the engine. Many have been ported from JPAN's hacked engine, and thus will work similar to how they worked there.

If a special is shown to have a var (such as *Var 0x8000*) as an input, set that var to the required data. If **SELECT_FROM_PC** is defined, remember to keep track of the Pokémon source from *Var 0x8003* before calling specials that manipulate Pokémon attributes.

If a special is shown to have a return value, it must be called with the **special2** scripting command.

Pokémon Specials

The following specials check or change Pokémon attributes. If **SELECT_FROM_PC** is defined, *Var 0x8003* will allow you to check/change data from PC boxed Pokémon if it is set to 1. Otherwise it will check/change from a party Pokémon.

Special 0x7 – EV/Contest Stat Checker

Details: Checks a party/boxed Pokémon's EVs or Contest stats.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: Stat to check:

HP EV	0x0
Attack EV	0x1
Defense EV	0x2
Speed EV	0x3
Special Attack EV	0x4
Special Defense EV	0x5
Coolness	0x6
Beauty	0x7
Cuteness	0x8
Smartness	0x9
Toughness	0xA
Luster	0xB

Returns: Stat value to given var.

Example Script:

```
setvar 0x8003 0x0 'Select from party
setvar 0x8004 0x3 '4th Pokémon in party
setvar 0x8005 0x1
special2 LAST_RESULT 0x7
buffernumber 0x0 LAST_RESULT 'Buffer Attack EV stat into [buffer1]
```

Special 0x8 – Pokémon IV Checker

Details: Checks a party/boxed Pokémon's IVs

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: IV stat to check:

HP IV	0x0
Attack IV	0x1
Defense IV	0x2
Speed IV	0x3
Special Attack IV	0x4
Special Defense IV	0x5

Returns: IV stat value to given var.

Example Script (with PC Selection Hack):

```
writebytooffset 0x1 0x0203b7ac 'Or whatever pcSelect_StateTracker is set to
special 0x3C 'Select boxed mon, box stored to var8000, slot to var8001
waitstate
compare LAST_RESULT 0x7F 'Player exited without selecting
if 0x1 goto @DidNotSelect
setvar 0x8003 0x1 'From boxed mon
setvar 0x8005 0x1 'Check attack IV
special2 LAST_RESULT 0x8
buffernumber 0x0 LAST_RESULT 'Buffer attack IV to [buffer1]
```

Special 0x9 – Pokémon Ribbon Checker

Details: Checks a Pokémon's ribbons. The first 5 ribbons have values from 1 - 4 for Normal, Super, Hyper, and Master rank, so be sure to reference the correct bit value(s). See [this page](#) for more info. These values cannot be checked from the PC as they are removed to compress box Pokémon size.

Input:

Var 0x8004: Holds the party slot number.

Var 0x8005: Ribbon bit to check. Here are the possible bits and known ribbon values.

Ribbon	Bit(s)/Rank	Var 0x8005 Val
Cool Ribbons	Normal (1) = bit 1 (0001)	0
	Super (2) = bit 2 (0010)	1
	Hyper (3) = bits 1,2 (0011)	0 & 1
	Master (4) = bit 3 (0100)	2
Beauty Ribbons	Normal (1) = bit 4	3
	Super (2) = bit 5	4
	Hyper (3) = bit 4,5	3 & 4
	Master (4) = bit 6	5
Cute Ribbons	Normal (1) = bit 7	6
	Super (2) = bit 8	7
	Hyper (3) = bit 7,8	6 & 7
	Master (4) = bit 9	8
Smart Ribbons	Normal (1) = bit 10	9

	Super (2) = bit 11 Hyper (3) = bit 10,11 Master (4) = bit 12	10 9 & 10 11
Tough Ribbons	Normal (1) = bit 13 Super (2) = bit 14 Hyper (3) = bit 13,14 Master (4) = bit 15	12 13 12 & 13 14
Champion	Bit 16	15
Winning	Bit 17	16
Victory	Bit 18	17
Artist	Bit 19	18
Effort	Bit 20	19
Special 1	Bit 21	20
Special 2	Bit 22	21
Special 3	Bit 23	22
Special 4	Bit 24	23
Special 5	Bit 25	24
Special 6	Bit 26	25
Special 7	Bit 27	26
Fateful Encounter	Bits 28-30	27 - 30
Obedience (Mew/Deoxys)	Bit 31	31

NOTE: the “hyper” status is a bit more challenging to check for, as the input is a bit number and these ribbon statuses require checking two bits. If this is something you are interested in implementing, you would need to remove the Normal bit upon receiving Super status, and then check for both bits with two separate special calls to determine the Hyper status.

Returns: 1 if the ribbon flag is set, 0 if not.

Example Script:

```

setvar 0x8004 0x2 '3rd mon in party
setvar 0x8005 15 'Check Champion ribbon
special2 LAST_RESULT 0x9
buffernumber 0x0 LAST_RESULT
compare LAST_RESULT 0x1
if 0x1 goto @IsAChampion
'Else, Pokémon was not in the hall of fame

```


Special 0xA – Pokérus Timer Checker

Details: Checks the [Pokérus](#) virus timer on a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: Pokérus time left to given var.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x0 'First party Pokémon
special2 LAST_RESULT 0xA
buffernumber 0x0 LAST_RESULT 'Pokérus timer into [buffer1]
bufferpartypokemon 0x1 0x0 'Buffer first poke name into [buffer2]
msgbox @timeLeft 0x6
```

#org @timeLeft

= [buffer2] is sick for [buffer1] more cycles!

Special 0xB – Poké Ball Checker

Details: Check the Poké Ball type of a Pokémon. The ball Ids can be found in **include/new/catching.h**.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: Poké Ball type to given var.

Example Script:

```
writebytetoffset 0x1 0x0203B7AC 'Select from PC hack
Special 0x3C 'Store box/slot into vars 0x8000, 0x8001, respectively
waitstate
Compare LAST_RESULT 0x7F
If 0x1 goto @didNotSelect 'Player cancelled without selection
setvar 0x8003 0x1 'From box
special2 LAST_RESULT 0xB
buffernumber 0x1 LAST_RESULT 'Buffer item number to [buffer2]
special 0x7C 'Buffer boxed mon nickname to [buffer1]
msgbox @ball 0x6
```

#org @ball

= [buffer1] is inside a [buffer2]! How fortunate!

Special 0xC – Check Capture Location

Details:

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: Capture Location Id to given var.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x5 'Last party mon
special2 LAST_RESULT 0xC
buffernumber 0x0 LAST_RESULT 'Buffer capture location to [buffer1]
```

Special 0xD – Happiness Checker

Details: Check the number of happiness points for a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: Happiness value (0-255) to given var.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x2 '3rd Pokémon
special2 LAST_RESULT 0xD
buffernumber 0x0 LAST_RESULT 'Buffer happiness to [buffer1]
compare LAST_RESULT 255
if 0x1 goto @maxedHappiness
```

Special 0xE – Hold Item Checker

Details: Check hold item value of a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: Hold Item Id to given var.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x3 '4th Pokémon
special2 LAST_RESULT 0xE
buffernumber 0x0 LAST_RESULT 'Buffer to [buffer1]
```

Special 0xF – Add/Subtract to EVs

Details: Add or subtract values to Pokémon EVs (between 0 and 252).

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: Stat for math (see Special 0x7 for indices).

Var 0x8006: Value to add. 0x01YY to subtract YY, 0x00ZZ to add ZZ.

Returns: Nothing.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x1 '2nd party Pokémon
setvar 0x8005 0x3 'Speed EV
setvar 0x8006 0x0150 'Subtracting 0x50, or 80 speed EVs
special 0xF
```

Special 0x10 – Set IVs

Details: Set IV values for a Pokémon. No math here, just setting to a specific value.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: IV stat to change (see Special 0x8 for indices), between 0 and 31 (0x1F).

Var 0x8006: IV value to set.

Returns: Nothing.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x0 'First Pokémon
setvar 0x8005 0x0 'HP IV
setvar 0x8006 31 'Value to set
special 0x10 'Maximize first party Pokémon's HP IV
```

Special 0x11 – Set Ribbons

Details: Set or clear a Pokémon's ribbon flag.

Input:

Var 0x8004: Holds the party slot number.

Var 0x8005: Ribbon flag to set (see Special 0x9 for indices/values).

0x00XX will set a ribbon, 0x01YY will clear a ribbon.

Returns: Nothing.

Example Script:

```
setvar 0x8004 0x2 '3rd Pokémon
setvar 0x8005 15 'Set Champion ribbon
Special 0x11 'Set the ribbon
```

Special 0x12 – Set Pokérus

Details: Set a Pokérus timer of a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: Number of cycles, 0x0 to 0xF, 0x10 to “cure”.

Returns: Nothing.

Example Script:

```
setvar 0x8003 0x0 'From party.  
setvar 0x8004 0x0 'First mon.  
setvar 0x8005 0x10 'Cure Pokémon's Pokérus.  
special 0x12
```

Special 0x13 – Change Happiness

Details: Add or subtract to a Pokémon's happiness.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: Amount to add/subtract.

0x01YY will subtract YY from happiness; 0x00XX will add XX to happiness.

Returns: Nothing.

Example Script:

```
setvar 0x8003 0x0 'From party  
setvar 0x8004 0x1 '2nd party Pokémon  
setvar 0x8005 0x0150 'Subtracting 0x50, or 80 friendship points  
special 0x13
```

Special 0x14 – Change Pokeball

Details: Set the ball type of a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: Ball ID to set.

Returns: Nothing.

Example Script:

```
setvar 0x8003 0x0 'From party  
setvar 0x8004 0x0 'First Pokémon  
setvar 0x8005 0x1 'Set to Master Ball  
special 0x14
```

Special 0x15 – Change Hold Item

Details: Set the hold item of a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: Item Id to set.

Note that if the Pokémon is already holding an item, this will not change the item, unless *Var 0x8005* is set to 0 to remove the item. Then you can call it again to set a new hold item.

Returns: 0 to LAST_RESULT if successful item change, 1 if not.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x5 'Last mon
setvar 0x8005 0x0 'Remove a hold item first
special 0x15
setvar 0x8005 ITEM_SILKSCARF 'Item to give
special 0x15 'Give silk scarf
compare LAST_RESULT 1
if 0x1 goto @Failed
```

Special 0x16 – Change Species

Details: Change the species of a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: Species to change to.

Returns: Nothing.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x0 'First Pokémon
setvar 0x8005 PKMN_CHARMANDER 'Set to Charmander
special 0x16
```

Special 0x17 – Change Attacks

Details: Set or remove a move for a Pokémon. This one does not work with the PC Selection Hack.

Input:

Var 0x8004: Pokémon Slot (0-5)

Var 0x8005: Move Slot (0-3 for moves 1-4, respectively)

Var 0x8006: Move Id (0 to clear move slot)

Returns: Nothing.

Example Script:

```
Special 0x9F 'Select a Pokémon from the menu, store slot to Var 0x8004
waitstate
compare LAST_RESULT 0x6
if 0x4 goto @Cancelled
setvar 0x8005 0 'First move
setvar 0x8006 MOVE_HYPERBEAM 'Teach Hyper Beam in slot 0
special 0x17
```

Special 0x18 – Check Species

Details: Check the species of a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: Species Id to given var.

Example Script:

```
setvar 0x8003 0x0 'From party
setvar 0x8004 0x0 'Check first Pokémon
special2 LAST_RESULT 0x18
compare LAST_RESULT PKMN_RATTATA 'Check if first Pokémon if Rattata
If 0x0 goto @NotCorrect
```

Special 0x19 – Check Attack PP

Details: Check a Pokémon move's PP.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Var 0x8005: Move slot (0-3).

Returns: PP left to given var.

Example Script:

```
setvar 0x8003 0x0 'From Party
setvar 0x8004 0x0 'First Pokémon
setvar 0x8005 0x0 'First move
special2 LAST_RESULT 0x19
buffernumber 0x0 LAST_RESULT
compare LAST_RESULT 0x0
if 0x1 goto @NoPPLeft
```

Party Specials

Special 0x62 – Erase Pokémon

Details: Erase a Pokémon from your party, or the entire party.

Input:

Var 0x8004: Slot to erase (0xF for entire party).

Returns: Nothing.

Example Script:

```
setvar 0x8004 0xF 'Erase entire party  
Special 0x62
```

Special 0x63 – Status Checker

Details: Check the primary status of a Pokémon.

Input:

Var 0x8004: Pokémon Slot

Returns: Status inflicted to given var.

Status	Bits	Hex Value
Sleep	1	0x1
	2	0x2
	3	0x4
Poison	4	0x8
Burn	5	0x10
Frozen	6	0x20
Paralyzed	7	0x40
Badly Poisoned	8	0x80

Example Script:

```
setvar 0x8004 0x0 'First party Pokémon  
special2 0x8004 0x63 'Get statuses to var 0x8004  
setvar 0x8005 0x80 'Badly poisoned  
special2 LAST_RESULT 0x42 'Var 0x8004 & Var 0x8005  
compare LAST_RESULT 0x1 '& will return 1 if the Pokémon has this status  
If 0x1 goto @BadlyPoisoned
```

Special 0x64 – Status Inducer

Details: Inflict a primary status on a party Pokémon.

Input:

Var 0x8004: Pokémon slot, or *0xF* for entire party.

Var 0x8005: Status(es) to induce (see Special 0x63 for values).

Var 0x8006: *1* if status should only be given to Pokémon that can be afflicted with it (ie. No paralysis on Electric-types), *0* otherwise.

Returns: Nothing.

Example Script:

```
setvar 0x8004 0xF 'First party Pokémon
setvar 0x8005 0x20 'Freeze entire party
setvar 0x8006 0x1 'Don't freeze Ice-types of Pokémon with Magma Armor
special 0x64
```

Special 0x65 – Check Pokémon HP

Details: Check the amount of HP remaining for a party Pokémon.

Input:

Var 0x8004: Pokémon slot.

Returns: HP to given var.

Example Script:

```
setvar 0x8004 0x0 'First Pokémon
special2 LAST_RESULT 0x65
compare LAST_RESULT 0x0
if 0x1 goto @DeadPoke
```

Special 0x66 – Inflict Party Damage or Heal

Details: Inflict damage on/heal a Pokémon, or entire party.

Input:

Var 0x8004: Pokémon slot, *0xF* for entire party.

Var 0x8005: Damage to inflict/heal.

Var 0x8006: *1* to heal, otherwise inflict damage.

Returns: Nothing.

Example Script:

```
setvar 0x8004 0xF 'Entire party
setvar 0x8005 20 '20 damage to party
setvar 0x8006 0x0 'Damage party
Special 0x66
```


Special 0xB0 – Load Party Pokémon Types

Details: Gets the types of the requested party Pokémon.

Input:

Var 0x8000: Party Pokémon number.

Returns:

Var 0x8000: Type 1.

Var 0x8001: Type 2.

Example Script:

```
Special 0x9F 'Select a Pokémon from the menu, store slot to Var 0x8004
waitstate
compare LAST_RESULT 0x6
if 0x4 goto @Cancelled
copyvar 0x8000 LAST_RESULT
special 0xB0
```

Special 0xB2 –Pokémon Type In Party

Details: Checks if a specific Pokémon type can be found among the party Pokémon.

Input:

Var 0x8000: Pokémon type.

Returns: *Last Result: Party Index* if type was found in party. 6 otherwise.

Example Script:

```
setvar 0x8000 0xA 'Check Fire-type in party
special 0xB2
compare LAST_RESULT 0x6
if == goto @NotFound
```

Special 0xCC –Pokémon in Party that Can Learn Draco Meteor

Details: Checks if any Pokémon in the player's party can learn Draco Meteor. It does not include a friendship check. That is handled in the tutor part.

Input: None.

Returns: *Last Result: TRUE* if any Pokémon can learn Draco Meteor. 0 otherwise.

Example Script:

```
special 0xB2
compare LAST_RESULT 0x0
if == goto @NoDragonTypesInParty
```

Special 0xD0 - Pokémon in Party that Can Learn TM/HM

Details: Checks if any Pokémon in the player's party can learn the given TM/HM number.

Input: *Var8000:* The TM/HM id to check (1 to *NUM_TM* + *NUM_HMS*).

Returns: *Given Var: Slot Id* of the Pokémon the can learn the TM/HM. 6 if none can learn.

Example Script:

```
setvar 0x8000 0x1 'Check Focus Punch
special2 LAST_RESULT 0xD0
compare LAST_RESULT 0x0
if == goto @NoFocusPunchPotentialInParty
```

Key Specials

Special 0x2B – Check AB Buttons

Details: Check if *A* or *B* has been pressed

Input: Nothing.

Returns: To the given var:

0x0: Neither pressed.

0x1: *A* was pressed.

0x2: *B* was pressed.

0x3: Both *A* & *B* were pressed.

Example Script:

```
#org @Loop
special2 LAST_RESULT 0x2B
compare LAST_RESULT 0x1 'Check for A
if 0x0 goto @Loop 'Player cannot continue onwards until they press A
Continue
```

Special 0x2C – Check D-Pad

Details: Check *D-Pad* presses.

Input: Nothing.

Returns: To given var:

0x0: No direction is pressed.

0x1: *Up* is pressed.

0x2: *Left* is pressed.

0x3: *Down* is pressed.

0x4: *Right* is pressed.

0x5: *Up-left* is pressed.

0x6: *Up-right* is pressed.

0x7: *Down-left* is pressed.

0x8: *Down-right* is pressed.

Example Script: Wait Until Player Presses Down Button

```
#org @start
goto @loop

#org @loop
special2 LAST_RESULT 0x2C
compare LAST_RESULT 0x3 'Down pressed
if 0x1 goto @continue 'exit infinite loop if player pressed down
goto @loop
```

Special 0x2D – Check Start/Select

Details: Check if *Start/Select* are pressed

Input: Nothing

Returns: To given var:

0x0: Nothing pressed.

0x1: *Select* pressed.
0x2: *Start* pressed.
0x3: Both *Start* & *Select* pressed.

Example Script:

```
special2 LAST_RESULT 0x2D  
compare LAST_RESULT 0x2 'Start pressed  
If 0x1 goto @PressedStart
```

Special 0x2E – Check L/R

Details: Check if *L/R* are pressed.

Input: Nothing.

Returns: To given var:

0x0: Nothing pressed.
0x1: *R* pressed.
0x2: *L* pressed.
0x3: *L* & *R* pressed.

Example Script:

```
special2 LAST_RESULT 0x2E  
compare LAST_RESULT 0x1 'R pressed  
if 0x1 goto @PressedR
```

Special 0x2F – Dump Keys

Details: Dump any and all keys that have been pressed.

Input: Nothing.

Returns: Key presses to Var 0x800D (LASTRESULT):

Key	Bit	Hex
A	1 = 0001	0x1
B	2 = 0010	0x2
Select	3 = 0100	0x4
Start	4 = 1000	0x8
Right	5 = 0001 0000	0x10
Left	6 = 0010 0000	0x20
Up	7 = 0100 0000	0x40
Down	8 = 1000 0000	0x80
R	9 = 0001 0000 0000	0x100
L	10 = 0010 0000 0000	0x200

Example Script: Wait until player presses A

```
#org @loop  
special 0x2F  
Compare LAST_RESULT 0x1 'A pressed  
If 0x1 goto @pressedA  
goto @loop
```

Special 0xC9 – Force Key Input

Details: Force a key input from the user.

Input:

Var 0x8004: Key(s) to force (bitfield) (see *Special 0x2F* for bits).

Var 0x8005: Number of times to press it. For held buttons, a good rule of thumb is 0x10 times the number of times you want to move (see Script Example)

Returns: Nothing.

Example Script:

```
setvar 0x8004 0x10 'Right
setvar 0x8005 0x30 'Step Right 3 Times
special 0xC9 'Force player to move right 3 steps
```

Special 0xCA – Prevent Key Press

Details: Prevent player from being able to press button(s).

Input:

Var 0x8004: Key(s) to prevent (bitfield). 0 to allow all keys.

Returns: Nothing.

Example Script:

```
setvar 0x8004 0x104 'Prevent R and Select from doing anything
special 0xCA
```

Special 0xCB – Assign Key Script

Details: assign a specific script to a key

Input:

Var 0x8004: Key to assign script to (0 to remove).

Loadpointer 0x0: Script pointer.

Returns: Nothing.

Example Script:

```
setvar 0x8004 0x100 'Assign to R
loadpointer 0x0 @DoSomething
special 0xCB 'Now, when in the overworld, pressing R will launch @DoSomething
```

Variable Math Specials

Special 0x3E – Add Variables

Details: Add the values of two variables together.

Input:

Var 0x8004: Variable holding the first value.

Var 0x8005: Variable holding the second value.

Returns:

Variable Inside Var 0x8004: Sum of two values.

Given Var: 1 if sum overflows 0xFFFF, 0 otherwise.

Example Script:

```
setvar 0x4050 50
setvar 0x4051 25
setvar 0x8004 0x4050
setvar 0x8005 0x4051
special2 LAST_RESULT 0x3E 'Addition result to first variable, 0x4050
compare LAST_RESULT 0x1
if 0x1 goto @overflow
```

Special 0x3F – Subtract Variables

Details: Subtract the values inside two variables.

Input:

Var 0x8004: Variable holding first value.

Var 0x8005: Variable holding second value.

Returns:

First Variable: Difference of values.

Given Var: 1 if difference underflows 0x0, 0 otherwise.

Example Script:

```
setvar 0x8004 0x4059
setvar 0x8005 0x4050
special2 LAST_RESULT 0x3F '[var 0x4059] = [var 0x4059] – [var 0x4050]
compare LAST_RESULT 0x1
If 0x1 goto @overflowed '[var 0x4059] > [var 0x4050]
```

Special 0x40 – Multiply Variables

Details: Multiply the values of two variables together.

Input:

Var 0x8004: Variable holding first value.

Var 0x8005: Variable holding second value.

Returns:

First Variable: $[Var\ 0x8004] \times [Var\ 0x8005]$.

Given Var: 1 if product overflows 0xFFFF, 0 otherwise.

Example Script:

```
setvar 0x8006 400
setvar 0x8007 200
setvar 0x8004 0x8006
setvar 0x8005 0x8007
special2 LAST_RESULT 0x40 '400*200 = 80000 = 0x13880 = (0xFFFF) + 0x3881
```

Special 0x41 – Divide Variables

Details: Divide the values in two variables.

Input:

Var 0x8004: Numerator.

Var 0x8005: Denominator.

Returns:

Var 0x8004: Integer result of Var8004 / Var8005

Given Var: Remainder (modulus) of the division.

Example Script:

```
setvar 0x8004 50
setvar 0x8005 6
special2 0x8006 0x41 'Var8004 = 50 / 6 = 8
buffernumber 0x0 0x8006 'Remainder = 2
```

Special 0x42 – AND Variables

Details: [Bitwise AND](#) two variables.

Input:

Var 0x8004: First value.

Var 0x8005: Second value.

Returns: AND result of two variables to given var.

Example Script:

```
setvar 0x8004 0xCB '1100 1011
setvar 0x8005 0xAA '1010 1010
special2 0x8004 0x42 '[1100 1011] & [1010 1010] = 1000 1010 = 0x8A
```

Special 0x43 – OR Variables

Details: [Bitwise OR](#) two variables

Input:

Var 0x8004: First value.

Var 0x8005: Second value.

Returns: OR result of two variables to given variable

Example Script:

```
setvar 0x8004 0x4 '0000 0100
setvar 0x8005 0x10 '0001 0000
special2 0x8004 0x43 'Var8004 = [0000 0100] | [0001 0000] = 0001 0100 = 0x14
```

Special 0x44 – XOR Variables

Details: [Bitwise XOR](#) two variables.

Input:

Var 0x8004: First value.

Var 0x8005: Second value.

Returns: XOR result of two variables to given variable

Example Script:

```
setvar 0x8004 0x12 '0001 0010
setvar 0x8005 0x18 '0001 1000
special2 0x8007 0x44 'Var8007 = [0001 0010] ^ [0001 1000] = 0000 1010 = 0xA
```

Frontier Specials

Special 0x52 – Generate Frontier Trainer Id

Details: Generates a random battle facility id in `gTowerTrainers` or `gSpecialTowerTrainers`, or a preset frontier brain in `gFrontierBrains` found in `src/Tables/Frontier_Trainers.c`, and a name for the trainer.

Input:

`Var 0x8000: 0` = Load data for trainer Opponent 1.
 `1` = Load data for trainer Opponent 2.
 `2` = Load data for partner Trainer.
`Var 0x8001: 0` = Load from `gTowerTrainers`.
 `1` = Load from `gSpecialTowerTrainers`.
 `2` = Load from `gFrontierBrains`.
`Var 0x8002: 0` = If `Var 0x8001` is 2, then the frontier brain id num.

Returns: To given var the OW sprite Id of the chosen trainer.

Also buffers the trainer name to `gStringVar1`.

Example Script: See below.

Special 0x53 – Load Frontier Intro Battle Message

Details: Loads the battle intro message of the requested trainer.

Input:

`Var 0x8000: 0` = Load data for trainer Opponent 1.
 `1` = Load data for trainer Opponent 2.
`Var 0x8001: 0` = Load from `gTowerTrainers`.
 `1` = Load from `gSpecialTowerTrainers`.
 `2` = Load from `gFrontierBrains`.

Returns: Nothing

Example Script: For battle against a normal single opponent:

```
#define VAR_RUNTIME_CHANGEABLE 0x4080

setvar 0x8000 OPPONENT_1 '0
setvar 0x8001 REGULAR_TRAINER '0
special2 VAR_RUNTIME_CHANGEABLE SPECIAL_GENERATE_TOWER_TRAINER '0x52
reappear FOE_NPC_ID 'The Person Id of the dynamic overworld person event
setvar 0x8000 OPPONENT_1 '0
setvar 0x8001 REGULAR_TRAINER '0
special BUFFER_TOWER_TRAINER_INTRO_MSG '0x53
callstd MSG_NORMAL 'Displays the buffered intro text
```

Special 0x54 – Get Frontier Streak

Details: Gets the streak for the requested Frontier format.

Input:

Var 0x8000: 0 = Current Streak.
1 = Max Streak.

Var 0x8001: 0xFFFF = Load style from var VAR_BATTLE_FACILITY_BATTLE_TYPE.
0 = Single battle.
1 = Double battle.
2 = Multi battle.
3 = Link multi battle.

Var 0x8002: 0xFFFF = Load tier from var VAR_BATTLE_FACILITY_TIER.
0 - 0xFF = See VAR_BATTLE_FACILITY_TIER description for available tiers.

Var 0x8003: 1 - 5 = Party size option 1.
6 = Party size option 2.

Var 0x8004: 0 = Load level from VAR_BATTLE_FACILITY_POKE_LEVEL.
1-MAX_LEVEL = Level bracket 1.
MAX_LEVEL = Level bracket 2.

Returns: The requested streak to the given var.

Example Script:

```
setvar 0x8000 CURRENT_STREAK '0
setvar 0x8001 VAR_LOAD_STYLE_FROM '0xFFFF
setvar 0x8002 VAR_LOAD_TIER_FROM '0xFFFF
setvar 0x8003 FLAG_LOAD_INVERSE_FROM '2
setvar 0x8004 VAR_LOAD_LEVEL_FROM '0
special2 LASTRESULT SPECIAL_GET_TOWER_STREAK '0x54
```

Special 0x55 – Update Current Frontier Streak

Details: Updates the streak for the current Frontier format.

Input:

Var 0x8000: 0 = Increment streak by 1.
1 = Reset streak.

Returns: Nothing.

Example Script:

```
setvar 0x8000 INCREMENT_STREAK_BY_1 '0
special SPECIAL_UPDATE_TOWER_STREAK '0x55
```


Special 0x56 – Determine Battle Points To Give

Details: Gets the number of battle points to give for the player's previous frontier win.

Input: None.

Returns: [Given var](#):

Streak Length	Num Battle Points
1 - 10	2
11 - 19	3
20	20 (Against Brain)
21 - 30	4
31 - 40	5
41 - 49	6
50	50 (Against Brain)
51+	7

Example Script:

```
special2 LASTRESULT SPECIAL_DETERMINE_BATTLE_POINTS '0x56  
'Add vars here to update total BP amount
```

Special 0x6B – Replace Player Team With Multi Trainer Team

Details: Temporarily replaces the player's team with the team of the given multi battle trainer to allow the player to choose which Pokémon they want the partner to use.

Input:

[Var 0x8000](#): Given multi trainer id of trainer in [gFrontierMultiBattleTrainers](#).

Returns: Nothing.

Example Script: See *Special 0x6C*.

Special 0x6C – Splice Frontier Multi Trainer Team With Player's Team

Details: To be used after special 0x6B. Merges the player's choice of partner Pokemon onto their team.

Input: None.

Returns: Nothing.

Example Script:

```
setvar 0x8000 0x0 'First trainer in gFrontierMultiBattleTrainers  
special 0x6B 'Replace player's team  
special 0xF5 'Choose Pokémon for battle  
waitstate  
special 0x6C 'Splice result, even if player cancelled.
```

Special 0x6D – Load Frontier Multi Trainer Data By Id

Details: Loads any relevant multi trainer data by the given Id value.

Input:

[Var 0x8000](#): [0-0xFE](#): Given multi trainer id of trainer in [gFrontierMultiBattleTrainers](#).
[0xFF](#): Random multi trainer id.

Returns: [Given Var](#): The OW sprite Id of the chosen trainer.

Also buffers the multi trainer name to [gStringVar2](#) ([BUFFER2] / bufferstring 0x1).

Example Script:

```
setvar 0x8000 0x0 'First trainer in gFrontierMultiBattleTrainers  
special2 VAR_RUNTIME_CHANGEABLE 0x6D
```

Special 0x6E – Buffer Battle Sands Records

Details: Buffers text relating to Battle Sands records.

Input:

Var 0x8000: 0 = Previous Streak.
1 = Max Streak.

Returns: *Last Result:* TRUE if the requested record exists.

Also buffers strings to the following:

gStringVar1: Tier name.
gStringVar2: Battle format name.
gStringVar3: Level.
gStringVar7: Inverse on or off.
gStringVar8: Species 1.
gStringVar9: Species 2.
gStringVarA: Species 3.
gStringVarB: Streak length.
gStringVarC: "previous" or "max"

Example Script:

```
setvar 0x8000 0 'Previous record
special 0x6E 'Buffer records
msgbox @TellRecords MSG_NORMAL
```

Special 0x6F – Can Team Participate in Battle Mine

Details: Checks if the player's team can enter the Battle Mine. Also sets the *VAR_BATTLE_FACILITY_TIER* to the chosen tier.

Input:

Var 0x8000: 0 = Check Battle Mine Format 1.
1 = Check Battle Mine Format 2.
2 = Check Battle Mine Format 3.

Returns: *LastResult:* TRUE if the team can participate.

Example Script:

```
setvar 0x8000 0 'Check eligibility in Battle Mine Format 1
special 0x6F
compare LASTRESULT 0x0
if == goto @CantParticipate
```

Special 0x70 – Randomize Battle Mine Options

Details: Randomizes various battle options for a battle in the Battle Mine.

Input: None.

Returns: Buffers strings to the following:

gStringVar7: Tier name.
gStringVar8: Battle format name.
gStringVar9: Level.
gStringVarA: Party size.
gStringVarB: Inverse on or off.

Example Script: special 0x70

Special 0x71 – Load Battle Mine Record Tier

Details: Sets the `VAR_BATTLE_FACILITY_TIER` var to the correct tier the Battle Mine streaks are recorded in. To be used after a battle.

Input: None.

Returns: Nothing.

Example Script: special 0x71

Special 0x72 – Load Battle Circus Effects

Details: Loads random effects for Battle Circus battles.

Input: None.

Returns: `Last Result`: TRUE if all effects loaded.

`gStringVarC`: A description of the effect just activated.

Example Script:

```
setweather 0x0 'Remove overworld weather data
```

```
#org @SetEffectsForBattle
special 0x72 'Try to add effect
compare LASTRESULT 0x0
if != goto @Return
msgbox @BattleWillTakePlaceIn MSG_NORMAL
doweather 'Try to update overworld weather
goto @SetEffectsForBattle
```

```
#org @Return
Return
```

Special 0x73 – Modify Team for Battle Facility

Details: Modifies the entered Pokémon's levels to be used in the Battle facility.

Input: None.

Returns: Nothing.

Example Script:

```
special 0x27 'Backup original party
special 0xF5 'Choose Pokémon to fight with
waitstate
special 0x28 'Need to have original team when saving game
special 0x23 'Save game
waitstate
special 0x73 'Modify chosen team
```

Special 0x67 – Generate Random Battle Frontier Team

Details: Generates a random Battle Frontier ready team using `VAR_BATTLE_FACILITY_POKE_LEVEL`. Teams are generated based on Pokémon found in `src/Tables/Frontier_Spreads.h`.

Input: Technically `VAR_BATTLE_FACILITY_POKE_LEVEL`.

Returns: Nothing.

Example Script: special 0x67

Battle Specials

Special 0x51 – Can Team Participate in a Sky Battle

Details: Checks if the player has at least one Pokémon on their team that can participate in a [Sky Battle](#). The Pokémon species banned from participating can be found under [gSkyBattleBannedSpeciesList](#) in `src/Tables/Pokémon_Tables.c`.

Input: None.

Returns: To given var [0](#) if team can't participate, [1](#) if it can.

Example Script:

```
special2 0x51
compare LASTRESULT 0x0
if == goto @CantParticipate
setflag FLAG_SKY_BATTLE 'Defined in config.h
trainerbattle 0x3 0x20 0x0 @lose
```

Special 0x58 – Buffer Swarm Text

Details: Buffers the map name where there is currently a swarm to *buffer1* and the species name where there is currently a swarm to *buffer2*.

Input: None.

Returns: Nothing.

Example Script:

```
special 0x58
msgbox @Saw 0x6 '[buffer1]! They said there's\na whole bunch of [buffer2] there!
```

Special 0x59 – Buffer Species Roaming Text

Details: Buffers the map name where the given roamer can be found to *buffer1*, and the species name of the roamer to *buffer2*.

Input:

[Var 0x8000](#): Species

Returns: To given var [0](#) if the requested species isn't found roaming. [1](#) otherwise.

Example Script:

```
setvar 0x8000 PKMN_CHARMANDER 'Charmander should roam
setvar 0x8001 25 'Level 25
setvar 0x8002 0x1 'Can roam on land
setvar 0x8003 0x0 'Cannot roam on water
special 0x129 'Create roaming Pokemon
compare LASTRESULT 0x0
if 0x1 goto @TooManyRoamers
setvar 0x8000 PKMN_CHARMANDER 'Find map where Charmander is roaming
special2 LASTRESULT 0x59 'Buffer roaming text
compare LASTRESULT 0x0
if 0x1 goto @NotRoaming
msgbox @Saw 0x6 '[buffer1]! They said a\n[buffer2] appeared there!
```

Special 0x5A – Wild Data Switch

Details: Overwrites the wild data on all maps until *special 0x5B* is used.

Input:

Loadpointer 0x0: New wild data header pointer.

Returns: Nothing.

Example Script:

```
loadpointer 0x0 0x83C9CB8 'Pointer to Route 1 wild data
special 0x5B
```

Special 0x5B – Cancel Wild Data Switch

Details: Cancels the wild data switch set by *special 0x5A*.

Input: None.

Returns: Nothing

Example Script:

```
special 0x5C
```

Special 0x97 – Random Grass Battle

Details: Initiates a random grass battle from the map wild data. Does nothing if no land data exists.

Input: None.

Returns: Nothing.

Example Script: See *Special 0x98*.

Special 0x98 – Random Sea Battle

Details: Initiates a random water battle from the map wild data. Does nothing if no water data exists.

Input: None.

Returns: Nothing.

Example Script:

```
#org @start
special 0x8F 'The trainer position special
special2 LAST_RESULT 0x7F 'Returns 1 if ground battles, 2 if water battle
compare LAST_RESULT 0x1
if 0x1 goto @grass
compare LAST_RESULT 0x2
if 0x1 goto @water
release
end

#org @grass
special 0x97 'Generates a random grass battle.
release
end

#org @water
special 0x98 'Generates a random water battle.
release
end
```

Special 0x156 – Ghost Battle

Details: Initiate a ghost battle with a given Pokémon, level, and held item.

Input:

Var 0x8004: Ghost species. Setting to 0 initiates a battle with default Marowak.

Var 0x8005: Ghost level.

Var 0x8006: Ghost hold item.

Returns: Nothing.

Example Script:

```
setvar 0x8004 PKMN_CHARIZARD
```

```
setvar 0x8005 100
```

```
setvar 0x8006 ITEM_LEFTOVERS
```

```
Special 0x156
```

Special 0xAC - Load Second Trainer Defeat Message

Details: If a battle against two opponents is being started from a flag, this special will load in the defeat text for the second trainer.

Input:

Loadpointer 0x0: Pointer to defeat text.

Returns: Nothing

Example Script: See [here](#).

Special 0x9C – Old Man Battle

Details: Initiate an old man battle with a specific Pokémon species and level.

Input:

Var 0x8004: Species.

Var 0x8005: Level.

Returns: Nothing.

Example Script:

```
setvar 0x8004 PKMN_BEEDRILL
```

```
setvar 0x8005 50
```

```
special 0x9C
```

```
waitstate
```



Timer Specials

Another feature from JPANs engine, which allows the player to utilize the game timer for timed events.

Special 0x46 – Start Timer

Details: Start the timer. If called after it started running, it resets the timer.

Input: None.

Returns: Nothing.

Example Script: special 0x46

Special 0x47 – Pause Timer

Details: Pauses the already-started timer. Stores the timer value to *gTimerValue*

Input: None.

Returns: Nothing.

Example Script: special 0x47

Special 0x48 – Resume Timer

Details: Resume a paused timer.

Input: None.

Returns: Nothing.

Example Script: special 0x48

Special 0x49 – Stop Timer

Details: Stops the timer and returns the value. Timer needs to be started anew, resuming the timer will cause the value to be inaccurate.

Input: None.

Returns: The timer value to the given var.

Example Script: special2 LAST_RESULT 0x49

Special 0x4A – Get Timer Value

Details: Just return the time on the timer.

Input: None.

Returns: The timer value to the given variable.

Example Script: special2 LAST_RESULT 0x4A

Special 0x4B – Stop and Update Playtime

Details: Stop the timer and update playtime value.

Input: None.

Returns: Nothing.

Example Script: special 0x4B

Special 0x4C – Update Playtime

Details: Update the playtime. This is meant for functions that take a while to process that cause delay in playtime.

Input: None.

Returns: Nothing.

Example Script: special 0x4C

Special 0x4D – Check Timer Value

Details: Check if timer has reached a value stored in *Var 0x8010*.

Input:

Var 0x8010: Value to check against.

Returns: 1 if timer is greater or equal, 0 otherwise.

Example Script:

```
setvar 0x8010 100
special2 LAST_RESULT 0x4D
Compare LAST_RESULT 0x1 'Is timer >=
If 0x1 goto @timeReached
```

Special 0x4E – Save Timer Value

Details: Store the timer value to a free RAM address, *gTimerValue* to allow you to later reset it to this value.

Input: None.

Returns: Nothing.

Example Script: special 0x4E

Special 0x4F – Start Timer at a Time

Details: Restart the timer at the value stored with Special 0x4E (value in *gTimerValue*).

Input: None.

Returns: Nothing.

Example Script: special 0x4F

Special 0x50 – Store Timer Value to Variable

Details: Store the timer value from *gTimerValue* to a given variable

Input:

Var 0x8006: Variable to store timer value to

Returns: Nothing.

Example Script:

```
Special 0x4e 'save timer value to gTimerValue
Setvar 0x8006 0x8000
special 0x50 'store timer value to variable 0x8000
```

Special 0x61 – Load Timer Value from Variable

Details: Set the timer value RAM, *gTimerValue*, from a variable

Input:

Var 0x8006: Variable holding timer value to set

Returns: Nothing.

Example Script:

```
Setvar 0x8000 50
setvar 0x8006 0x8000
special 0x61 'set gTimerValue to 50
```


Safari Specials

Special 0x86 – Get Safari Balls

Details: Check Safari Ball quantity.

Input: None.

Returns:

Given Var: Number of Safari Balls

Example Script:

```
special2 LAST_RESULT 0x86
buffernumber 0x0 LAST_RESULT
msgbox @numSafariBalls 0x6
```

#org @numSafariBalls

= You have [buffer1] Safari Balls remaining!

Special 0x87 – Change Safari Balls

Details: Increase or decrease the safari ball count, maximum up to **MAX_SAFARI_BALLS**.

Input:

Var 0x8004: Number to increase/decrease by (up to 100)

0x1XX decreases by XX, *0x1YY* increases by YY.

Returns: Nothing.

Example Script:

```
setvar 0x8004 0x011E 'Remove 30 safari balls
special 0x87
```

Special 0x88 - Get Safari Pedometer

Details: Get the value of the safari pedometer.

Input: None.

Returns: Safari pedometer value to given variable.

Example Script:

```
special2 LAST_RESULT 0x88
compare LAST_RESULT 50 'Check if taken at least 50 steps
if 0x4 goto @OverFifty
```

Special 0x89 – Set Safari Pedometer

Details: Set a specific number of steps in the safari.

Input:

Var 0x8004: Pedometer Value to set.

Returns: Nothing.

Example Script:

```
special2 LAST_RESULT 0x88 'Get current pedometer
Compare LAST_RESULT 50
If 0x3 goto @Continue
setvar 0x8004 50 'Pedometer can only reach 50 steps
Special 0x89 'Set the safari step counter to 50
```

Walking Specials

Special 0x7E – Get Tile Number

Details: Get the tile number at a specified location on the current map.

Input:

Var 0x8004: Tile x-coordinate.

Var 0x8005: Tile y-coordinate.

Returns: Tile number to the given var.

Example Script:

```
getplayerpos 0x8004 0x8005 'Player's current position
addvar 0x8004 0x2 'Check tile 2 steps to the right of the player
special2 LAST_RESULT 0x7E
compare LAST_RESULT SOME_TILE_NUMBER
if 0x1 goto @CorrectTile
```

Special 0x7F – Get Tile Behaviour

Details: Get a specific tile set of attributes

Input:

Var 0x8004: Tile x-coordinate.

Var 0x8005: Tile y-coordinate.

Returns:

Var 0x8004: Tile background byte.

Var 0x8005: Tile behaviour bytes.

Given Var: Tile background byte.

Example Script:

```
Getplayerpos 0x8004 0x8005
special2 LAST_RESULT 0x7F 'Get tile attributes on player's current tile
```

Special 0x81 – Set Walking Script

Details: Load a walking script to run each step. Setting to zero removes any walking script.

Input:

Loadpointer 0x0: Script pointer.

Returns: Nothing.

Example Script:

```
Loadpointer 0x0 @WalkingMsg
Special 0x81
release
end
```

#org @walkingMsg

Msgbox @msg 0x6

end

#org @msg

= This msgbox will play every single step.

Special 0x8A – Read Pedometer Value

Details: This special is not in JPAN's original engine. It reads the value of one of the extra pedometers included in the engine, which are set with flags (see customization).

Input:

Var 0x8004: Pedometer to read

- 0: always active pedometer (32bit)
- 1: large valued-pedometer (32bit)
- 2: medium valued-pedometer (16bit)
- 3: first small pedometer (8bit)
- 4: second small pedometer (8bit)

Returns: Pedometer value to the given variable.

Example Script:

```
setvar 0x8004 0x0 'Pedometer that's always on  
special2 LAST_RESULT 0x8A 'Get number of steps player has walked  
buffernumber 0x0 LAST_RESULT
```

PC Selection Specials

A couple new specials are added, as well as a few existing specials changed to allow data manipulation of boxed Pokémon.

Special 0x1A – Store/Return Party Pokémon Data

Details: Save or Return party/boxed Pokémon Data

Input:

Var 0x8002:

- 0 For store to free ram.
- 1 For return to party from free ram.
- 2 For store from free ram to box.
- 3 For store from box to free ram.

Var 0x8005: Party slot number (for special 0xFE inputs).

Returns: 0 or 1 to LAST_RESULT for success/failure, respectively

Script Example:

```
setvar 0x8002 0 'Store
setvar 0x8005 0 'Save first party mon data
Special 0x1A    'First party Pokémon data now in Enemy data slot 5
```

Special 0x1B – Swap Party/Boxed Pokémon Data

Details: Swap party and box data

Input:

Var 0x8000: Box Number .

Var 0x8001: Box Position.

Var 0x8002: 0 for withdraw from box, 1 for store to box.

Var 0x8005: Party slot number (for special 0xFE inputs).

Returns: 0 or 1 to LAST_RESULT for success/failure, respectively.

Script Example – Swap Party and Boxed Mon

```
Msgbox @ask 0x6 'Select party mon to deposit
Special 0x9F
waitstate
copyvar 0x8005 0x8004
setvar 0x8002 0 'From party to free ram
Special 0x1A ' Store from party to free ram
msgbox @ask2 0x6 'Select boxed mon to withdraw
writebytetooffset 0x1 0x0203b7ac
special 0x3C 'Select boxed mon, box stored to Var8000, slot to Var8001
waitstate
setvar 0x8002 0x0 'Withdraw
special 0x1B 'Selected boxed mon to selected party slot (Var8005)
setvar 0x8002 0x2 'Free ram to box
special 0x1A 'Free ram (eg. Original selected party mon) to same box slot
```

NOTE: Rather than use a single special for this, the dynamic inputs of these specials allow for swapping party/boxed mon, trading, and more.

Special 0x7C – Buffer nickname

Details: Buffer a Pokémon's nickname to [buffer1].

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: Nothing.

Example Script: See *Special 0x9E*.

Special 0x7D – Check Traded Pokémon

Details: Check if Pokémon is traded.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: *LastResult*: 0 if traded, 1 if not.

Example Script: See *Special 0x9E*.

Special 0x9E – Nickname Pokémon

Details: Nickname a Pokémon.

Input:

Var 0x8003: From party (0), or box (1).

From Box: *Var 0x8000*, *Var 0x8001* hold the box num/slot, respectively.

From Party: *Var 0x8004* holds the party slot number.

Returns: *LastResult*: 0 if traded, 1 if not.

Example Script – Nickname a Boxed Pokémon

writebytetooffset 0x1 0x0203B7AC

special 0x3C 'Select boxed mon, box stored to Var8000, slot to Var8001

waitstate

setvar 0x8003 0x1

special 0x7C 'Buffer nickname

Msgbox @AskNickname 0x5 'Nickname [buffer1]?

Compare LAST_RESULT 0x1

If 0x0 goto @Nope

Special 0x7D 'Check traded mon

Compare LAST_RESULT 0

If 0x1 goto @Traded

Special 0x9E 'Nickname boxed mon

waitstate

Other Specials

Special 0x24 – Add Multichoice Text By Variable

Details: Add a dynamic multichoice option by variables

Input:

Var 0x8004: Upper halfword of pointer.

Var 0x8005: Lower halfword of pointer.

Var 0x8006: Multichoice Index.

Returns: Nothing.

Example Script:

```
setvar 0x8004 0x0890
```

```
setvar 0x8005 0x5040
```

```
setvar 0x8006 0x0
```

```
Special 0x24 'Multichoice index 0 is string pointer 0x08905040
```

NOTE: Special 0x25 is easier to use, you don't have to worry about upper/lower sections of a word.

Special 0x25 – Add Multichoice Text by Pointer

Details: Add a dynamic multichoice option by a pointer.

Input:

Var 0x8006: Multichoice Index.

Loadpointer 0x0: Pointer to string.

Returns: Nothing.

Example Script:

```
setvar 0x8006 0x0
```

```
loadpointer 0x0 @option1
```

```
special 0x25
```

```
setvar 0x8006 0x1
```

```
loadpointer 0x0 @option2
```

```
special 0x25
```

```
preparemsg @msg
```

```
waitmsg
```

```
multichoice 0x0 0x0 0x20 0x0 'See note below
```

```
compare LAST_RESULT 0x0
```

```
if 0x1 goto @selectedOption1
```

```
compare LAST_RESULT 0x1
```

```
if 0x1 goto @selectedOption2
```

NOTE: multichoice 0xX 0xY 0xWW 0xZ

0xX: X position of box.

0xY: Y position of box.

0xWW: Multichoice box index.

0x20: 2 options (min).

...

0x25: 7 options (max).

0xZ: 0x0 if B can cancel box, 0x1 if not

Special 0x74 – Is Pokémon Storage System Full

Details: Checks if there is space in the PC for a new Pokémon.

Input: None.

Returns: **Given Var:** 1 if no space left in PC. 0 otherwise.

Example Script:

```
special 0x74
compare LAST_RESULT 0x0
if == goto @YesSpace
```

Special 0x75 – Buffer Species

Details: Buffer a Pokémon's species to [buffer3] and size to [buffer1].

Input:

Var 0x8005: Holds the variable that stores measurements.

Var 0x8006: Species to evaluate.

Returns: Nothing.

Example Script: See below

Special 0x76 – Measure Pokémon

Details: Play the measure Pokémon game.

Input:

Var 0x8004: Party slot of Pokémon.

Var 0x8005: Holds the variable that stores measurements.

Var 0x8006: Species to evaluate.

Returns: **Given var:**

1: Pokémon is not of selected type.

2: Pokémon is smaller.

3: Pokémon is bigger, also stores biggest value in variable in *Var 0x8005*.

4: Sizes are equal.

Special 0x18B – Show Fossil Image

Details: Load a custom image into the fossil image window.

Pointer Table - Defined in src/config.h		
Table can also be generated by the engine in src/script_specials.c . Search for <i>gFossilImageTable</i> .		
Name	Bytes	Description
Fossil Pointer	4	Pointer to a section of data that has the needed information for the function to work.
Palette Pointer	4	A simple uncompressed palette must be on the other end.
Fossil Data (Pointed to from "Fossil Pointer")		
Image Pointer	4	Pointer to the actual image. Uncompressed, 64x64 pixel, so each should take 4kb worth of space.
Constant bytes	4	Don't know what they do, but when changed they mess up the whole picture. Must be 0008581b.
Null bytes	8	8 zeros. Changing them will result in the same as messing with the above bytes.

Input:

Var 0x8004: Image number.

Var 0x8005: X coordinate on screen.

Var 0x8006: Y coordinate on screen.

Returns: Nothing.

Example Script:

```
setvar 0x8004 0x2 'Show third image in table
setvar 0x8005 0x0 'At (0x0)
setvar 0x8006 0x0 'At (0x0)
special 0x18B
```

Special 0x9A – Stop Sounds

Details: Stops all sound effects currently playing

Input: None.

Returns: Nothing.

Example Script:

```
sound 0x15
special 0x9A 'Stops the sound
```

Special 0xAE – Dynamic Clearflag

Details: Clears the flag loaded.

Input:

Var 0x8000: Flag to clear.

Returns: Nothing.

Example Script:

```
setvar 0x8000 0x2D 'Clear flag 0x2D
special 0xAE
```

Special 0xAF – Dismount Bicycle

Details: If the player is on the bicycle, they dismount.

Input: None.

Returns: Nothing.

Example Script: special 0xAF

Special 0xB1 – Choose Item From Bag

Details: Opens the bag and lets the player select an item.

Input:

Var 0x8000: 0 = Any pocket, Open from Item's pocket.
1 = Any pocket, Open from Key Item's pocket.
2 = Any pocket, Open from Poke Ball pocket.
3 = Any pocket, Open from Item's pocket.
4 = Berry Pouch
5 = TM Case

Returns: *Var 0x800E:* The item the player chose. 0 if they chose nothing.

Example Script:

```
setvar 0x8000 0x4 'Choose berry
special 0xB1
```


Special 0xB3 – Do Choose Number Screen

Details: Opens the naming screen and allows the player to input a number.

Input: None.

Returns: [Last Result](#): The number entered. 0xFFFF if number was invalid.

Example Script:

```
special 0xB3
waitstate
compare LASTRESULT 0xFFFF
if == goto @NotValidNumber
```

Special 0x196 – Try To Copy TM/HM Name To Buffer1

Details: If the given item is a TM/HM, buffers its name.

Input:

[Var 0x8000](#): Item id of TM/HM.

Returns: [Given Var](#): 0 if couldn't copy name. 1 if name copied.

[gStringVar1](#): TM/HM Name.

Example Script:

```
setvar 0x8000 TM01
special 0x196
```

Special 0xD1 – Create Follower NPC

Details: Causes the requested NPC to begin following the player wherever they go. This feature will not work perfectly in every situation (NOT BUGS). Eg, the NPC will not spin like the player on spin pads, and jumping over ledges / going on side stairs have a slight delay while the player waits for it to finish the jump / exit the staircase. It also is set to not follow during scripts. Adding an NPC to the table [gFollowerAlternateSprites](#) in [src/follow_me.c](#), will allow you to set specific biking and surfing sprites.

Input:

[Var 0x8000](#): Id of NPC (or var containing id) to start following the player.

[Var 0x8001](#): Flags to determine follower properties (bitset):

0x1 = NPC has specific running frames like the player.

0x2 = Player is allowed to bike while with NPC. Changes NPC to its biking frames (if can).

0x4 = Player can use Fly, Teleport, Dig, or use an Escape Rope while with NPC.

0x8 = Player can surf while with NPC. Changes NPC to its surfing frames (if can).

0x10 = Player can use Waterfall while with NPC.

0x20 = Player can use Dive while with NPC.

0x40 = Player can use Rock Climb while with NPC.

Returns: Nothing.

Example Script:

```
setvar 0x8000 LAST_TALKED 'NPC player is currently talking to
setvar 0x8001 0x48 'Can Surf + Rock Climb (0x8 + 0x40)
special 0xD1 'Convert the NPC into a follower
```

Special 0xD2 – Destroy Follower NPC

Details: If *special 0xD1* was used to create a follower NPC prior, then this follower NPC will now disappear. Any flag given to the original NPC will also be set.

Input: None.

Returns: Nothing.

Example Script: special 0xD2

Scripted Field Move Specials

These specials can be used to help implement the [PokéRide](#) feature found in recent main series Pokémon games.

Special 0x100 – Can Player Use Flash In Current Location

Details: Checks if [Flash](#) can be used from the player's current location.

Input: None.

Returns: **LastResult:** 0 if can't use *Flash*, 1 if can.

Example Script: See below.

Special 0x101 – Can Player Use Fly In Current Location

Details: Checks if [Fly](#) can be used from the player's current location.

Input: None.

Returns: **LastResult:** 0 if can't use *Fly*, 1 if can.

Example Script: See below.

Special 0x102 – Is Player Facing Surfable Water

Details: Checks if [Surf](#) can be used from the player's current location.

Input: None.

Returns: **LastResult:** 0 if can't use *Surf*, 1 if can.

Example Script: See below.

Special 0x103 – Is Player Facing Climbable Waterfall

Details: Checks if [Waterfall](#) can be used from the player's current location.

Input: None.

Returns: **LastResult:** 0 if can't use *Flash*, 1 if can.

Example Script: See below.

Special 0x104 – Is Player on Diveable Water

Details: Checks if [Dive](#) can be used from the player's current location.

Input: None.

Returns: **LastResult:** 0 if can't use *Dive*, 1 if can.

Example Script: See below.

Special 0x105 – Is Player Facing Climbable Wall

Details: Checks if [Rock Climb](#) can be used from the player's current location.

Input: None.

Returns: **LastResult:** 0 if can't use *Rock Climb*, 1 if can.

Example Script: (Also for all above)

special 0x1XX 'Where XX is replaced with one of the above special numbers
compare LASTRESULT 0x0
if == goto @CantUse

Special 0x109 – Is Player Facing NPC With Overworld Sprite

Details: Checks if the player is facing an NPC with a given overworld sprite number.

Input:

Var 0x8000: Sprite Id of NPC to check for.

Returns: *LastResult:* 0 if not facing NPC with sprite Id, 1 if is facing.

Example Script:

```
setvar 0x8000 95 'Sprite Id of Cut tree
special 0x109
compare LASTRESULT 0x0
if == goto @NotFacing
```

Time-Based Specials

Special 0xA0 - Check And/Or Set Daily Event

Details: Checks if a daily event has been run. It can also simultaneously set a daily event to “done”.

Input:

Var 0x8000: The first of a pair of vars containing the daily event data. Note that the var after this var is used as well (hence why “pair” was mentioned).

Var 0x8001: Set to 0 if you just want to check if the event has been done. Any other value sets the daily event as “done”.

Returns: 0 if the event has already been completed. 1 otherwise.

Special 0xA1 - Update Time in Vars

Details: Updates the time stored in a pair of vars to the current time.

Input:

Var 0x8000: The first of a pair of vars containing the daily event data. Note that the var after this var is used as well (hence why “pair” was mentioned).

Special 0xA2 - Get Time Difference

Details: Gets the time difference between the data stored in a pair of vars and the current time. These vars should be set by *special 0xA1* or *special 0xA2*.

Input:

Var 0x8000: The first of a pair of vars containing the daily event data. Note that the var after this var is used as well (hence why “pair” was mentioned).

Var 0x8001: Set to one of the following values:

- 0 - Get the minute difference.
- 1 - Get the hour difference.
- 2 - Get the day difference.
- 3 - Get the month difference.
- 4 - Get the year difference.

Example Script:

```
#define SP_DAILY_EVENT 0xA0
#define SP_UPDATE_TIME_IN_VARS 0xA1
#define SP_GET_TIME_DIFFERENCE 0xA2
#define VAR_DAILY_EVENT 0x50D2 'Also uses 0x50D3
```

```
#dynamic 0x740000
#org @start
setvar 0x8000 VAR_DAILY_EVENT
setvar 0x8001 0x0 'Don't set daily event var to done
special2 LASTRESULT SP_DAILY_EVENT
compare LASTRESULT 0x0
if == goto @AlreadyDid
setvar 0x8000 VAR_DAILY_EVENT
special SP_UPDATE_TIME_IN_VARS
msgbox @havenot "I have not done it."
callstd MSG_FACEPLAYER
end
```

```
#org @AlreadyDid
setvar 0x8000 VAR_DAILY_EVENT
setvar 0x8001 0x0 'Minute difference
special2 LASTRESULT SP_GET_TIME_DIFFERENCE
buffernumber 0x0 LASTRESULT
msgbox @already "\h02 minutes ago I did it."
callstd MSG_FACEPLAYER
end
```

Special 0xAD - Get Time Of Day

Details: Gets the current time of day.

Input:

Var 0x8001: 0 = Distinct times.

1 = Merge morning, day, and evening into day.

Returns: *Given Var:*

0 = Morning.

1 = Day.

2 = Evening.

3 = Night.

Example Script:

special2 LASTRESULT 0xAD

compare LASTRESULT 0x0

if == goto @Morning

Special 0xD9 - Get Hour

Details: Gets the current hour in the day.

Input: None.

Returns: *Given Var:* Current hour of the day.

Example Script: special2 LASTRESULT 0xD9

Special 0xDA - Get Day of Week

Details: Gets the current day of week.

Input: None.

Returns: *Given Var:* Current day of week.

Example Script: special2 LASTRESULT 0xDA

Creating New Battle Mechanics

The following sections detail how to add new moves, abilities, poke balls, and items to the battle engine. Because there is no cookie-cutter methodology for writing code for each ability or item effect, these sections only tell you how to set up the necessary data. The actual logic and coding are up to the user.

Moves

1. Declaration

- a. Add a definition in **include/constants/moves.h** following the same format,

#define MOVE_<your move name>. It should ideally go after **Hold Hands** and before the Z-Moves.

```
#define MOVE_HOLDHANDS 0x2A5
#define MOVE_AURA_WHEEL 0x2A6
#define MOVE_BREAKNECK_BLITZ_P 0x2A7
```

- b. This index must stay the same in all table positions! This is easily done by not editing any move IDs and adding new moves after **MOVE_HOLDHANDS**. If adding new moves before the Z-Moves, make sure to change the defined indices for all Z-Moves as well.

2. Move Name

- a. Add to the end of **strings/attack_name_table.string** following the **#org**

@NAME_<your name here> format. The move name “table” is generated directly from the ordering of the **#org** declarations in this file, so make sure the ordering is correct!

```
#org @NAME_HOLD_HANDS
Hold Hands

#org @NAME_AURA_WHEEL
Aura Wheel
```

3. Move Description

- a. Add to the end of **strings/Attack_Descriptions.string** following the **#org @DESC_<your name here>** format, following the ordering of your move ID definitions from step 1!

```
#org @DESC_HOLDHANDS
The user and an ally hold hands. This makes them very happy.

#org @DESC_AURAWHEEL
Aura Wheel inflicts damage. Its type changes depending on Morpeko's current mode.
```

- b. Add a pointer in the **gMoveDescriptions** table in **assembly/data/attack_description_table.s**, following the **.word DESC_<your name here>** format, following the ordering of your move ID definitions from step 1!

```
.word DESC_HOLDHANDS
.word DESC_AURA_WHEEL
```

4. Animation

- a. Add the animation pointer in the respective location in **gAttackAnimations**, typically called **ANIM_<your move name>**. Follow the ordering of your move ID definitions from step 1!
- b. Create your animation. This is most easily done with animation command macros. See examples after the table in **assembly/data/attack_anim_table.s** as well as the macros in **Anim_Defines.asm**

```
.word ANIM_HOLDHANDS
.word ANIM_AURAWHEEL
.word ANIM_BREAKNECK_BLITZ
```

c. If you are adding new backgrounds:

i. Add the indexed image to

graphics/Backgrounds/Animation_Backgrounds/ for a 256x112 image.
256x256 may go in **Larger_Anim_BGs**.

ii. Add the new BG index to **Anim_Defines.asm**. It is easiest to add just after **BG_SNUGGLE_FOREVER**.

```
.equ BG_SNUGGLE_FOREVER, 0x46  
.equ BG_AURA_WHEEL, 0x47
```

iii. Add the definition to

assembly/data/anim_backgrounds_graphics_defines.s following the same format. This step isn't 100% necessary, but helps keep formatting

```
.equ BG_AURA_WHEEL_IMG, AuraWheelTiles  
.equ BG_AURA_WHEEL_PAL, AuraWheelPal  
.equ BG_AURA_WHEEL_RAW, AuraWheelMap
```

consistent. If your file name is *NewBg*, your image declaration would be *NewBgTiles*, your palette would be *NewBgPal*, and the Map would be *NewBgMap*.

iv. Add the background to the animation background table in

assembly/data/anim_background_table.s, following the ordering of

```
.word BG_SNUGGLE_FOREVER_IMG, BG_SNUGGLE_FOREVER_PAL, BG_SNUGGLE_FOREVER_RAW  
.word BG_AURA_WHEEL_IMG, BG_AURA_WHEEL_PAL, BG_AURA_WHEEL_RAW
```

your indices from step 4.b.ii. The table goes [IMG_PTR, PAL_PTR, MAP_PTR], so follow your definitions from 4.b.iii.

v. Your background can now be called in animations by your definition from 4.b.ii, eg. **loadBG1 BG_<your_bg_name>**

d. If you are adding new particles:

i. Add the indexed .png file to **graphics/Attack_Particles/**

ii. Define a new particle tag at the end of **Anim_Defines.asm**. It is easiest to keep all current definitions and add to the end after

ANIM_TAG_MUD_BOMB.

```
.equ ANIM_TAG_MUD_BOMB, 0x286F  
.equ ANIM_TAG_AURA_WHEEL, 0x2870
```

iii. Add the particle definitions

to **assembly/data/Anim_Particle_Graphics_Defines.s**. Similar to the backgrounds, this isn't necessary but keeps the naming convention consistent.

```
.equ MUD_BOMB_IMG, MudBombTiles  
.equ AURAWHEEL_IMG, AuraWheelTiles
```

For a file called *NewParticle.png*,

```
.equ MUD_BOMB_PAL, MudBombPal  
.equ AURAWHEEL_IMG, AuraWheelPal
```

your image header will be called *NewParticleTiles* and your palette will be *NewParticlePal*.

iv. Add the above definitions to *gBattleAnimPicTable* and

gBattleAnimPaletteTable in **assembly/data/anim_particle_table.s**. These are two separate tables, the palettes following the images. So make sure you add *animparticle <your_particle_IMG>*, (*<width> * <height>*) / 2, *ANIM_TAG_<your_particle_tag>* and *animparticlepal <your_particle_PAL>*, *ANIM_TAG_<your_particle_tag>*, 0x0 in the appropriate spots in each respective table.

```
animparticle MUD_BOMB_IMG, (64 * 64) / 2, ANIM_TAG_MUD_BOMB  
animparticle AURAWHEEL_IMG, (64 * 64) / 2, ANIM_TAG_AURA_WHEEL  
animparticlepal MUD_BOMB_PAL, ANIM_TAG_MUD_BOMB, 0x0  
animparticlepal AURAWHEEL_PAL, ANIM_TAG_AURA_WHEEL, 0x0
```

- v. You can now call your particles by the defined particle tag in any *objtemplate* (follow examples in **assembly/data/Attack_Anim_Table.s**).

5. Battle Data

- a. Add your static definition in the respective location inside *gBattleMoves*, found in **src/Tables/battle_moves.c**, following any existing move for each structure element definition. Here is a description of each element in the BattleMove structure:

Structure Element	Definition
effect	Battle effect, e.g. poisoning, atk up. See include/battle_move_effects.h
power	Base power (0-255)
type	Move type (eg. TYPE_NORMAL)
accuracy	Move base accuracy % (0-100)
pp	Move's base pp
secondaryEffectChance	Chance of activating the move's effect % (0-100)
target	Who the move targets: MOVE_TARGET_SELECTED – choose target MOVE_TARGET_DEPENDS – depends on the move/scenario (eg. counter) MOVE_TARGET_USER_OR_PARTNER – Acupressure, choose between self or partner MOVE_TARGET_RANDOM – random target, eg. Thrash MOVE_TARGET_BOTH – affects both opponents MOVE_TARGET_USER – targets self MOVE_TARGET_FOES_AND_ALLY – all adjacent targets, eg. Surf MOVE_TARGET_ALL – target all battlers MOVE_TARGET_OPPONENTS_FIELD – affect the opponent's field, eg. Spikes
priority	Move's priority. See here . Priority is a signed integer, so negative values would subtract from 256. Eg. Revenge has priority -4, or 252.
flags	FLAG_MAKES_CONTACT – contact move FLAG_PROTECT_AFFECTED – move blocked by protect FLAG_MAGIC_COAT_AFFECTED – Move is reflected by Magic Coat FLAG_SNATCH_AFFECTED – Move is affected by Snatch FLAG_MIRROR_MOVE_AFFECTED – Move is copyable by Mirror Move FLAG_KINGS_ROCK_AFFECTED – King's Rock works with the move FLAG_TRIAGE_AFFECTED – Move is affected by the ability Triage
z_move_power	Base power of the associated Z-move
split	SPLIT_PHYSICAL, SPLIT_SPECIAL, SPLIT_STATUS for move split type
z_move_effect	Move effect of the associated Z-move. See z_move_effects.h

See [here](#) to add your move to a table and give it special effects. You now have a fully defined move! You can add it to a Pokémon's learnset inside **src/Tables/level_up_learnsets.c**, give it to a trainer's Pokémon, add in some AI logic, and whatever else you desire.

Abilities

1. Ability ID – add your `#define ABILITY_<your_ability>` at the end of `include/constants/abilities.h`. Then adjust `ABILITIES_COUNT` to the last ability index +1.

```
#define ABILITY_GALVANIZE 0xED
#define ABILITY BALL_FETCH 0xEE
#define ABILITIES_COUNT (ABILITY BALL_FETCH + 1)
```
2. Ability Name – add `#org @NAME_<your_ability>` and the ability name in `strings/ability_name_table.string`. Make sure the positioning of your ability name matches the index in Step 1.

```
#org @NAME_GALVANIZE
Galvanize

#org @NAME BALLFETCH
Ball Fetch
```
3. Ability Description – add the header, `#org @DESC_<your_ability>` to `strings/ability_descriptions.string`, as well as the description itself.
4. Add a pointer to your description inside `gAbilityDescription`, found in `assembly/data/Ability_Description_Table.s`. Make sure the position matches the index defined in Step 1!

```
#org @DESC_GALVANIZE
Normal moves become Electric-type.

#org @DESC BALLFETCH
Collects first failed Pok\e-Ball.
```
5. Add your new ability to one of the ability tables (see [here](#)) above the `ABILITY_TABLES_TERMIN` to give it a certain property (such as being affected by *Mold Breaker*, being ignored by *Skill Swap*, etc.).
6. Your new ability is now defined and can be assigned to a Pokémon with the Ability ID defined in Step 1. Any ability coding/logic must be written by the user. If you wish to give the ability the same effect as a pre-existing ability, simply search for the ability name in the engine and add logic for the new ability as well. This is easiest when the pre-existing ability is checked for in a switch statement. Simply stack the new ability as a case on top. See [here](#) for an explanation of switch statements.

Poke Balls

1. Define a Ball ID – add a new ball type inside the *enum BallTypes* via `BALL_TYPE_<ball_name>` inside `include/new/catching.h`. You can only have up to 32 ball types, due to the restriction on bits.
 - a. Update `LAST_BALL_INDEX` and `NUM_BALLS` as needed.
2. Throwing Image – add the indexed 16x48 .png file to `graphics/Poke_Balls/`, and any new particles to `graphics/Poke_Balls/Opening_Particles/`.
 - a. Note that the bag image is excluded because item data is untouched in this engine.
3. Open `src/Tables/Ball_Graphics_Tables.c`.

```
BALL_TYPE_DREAM_BALL, //26
BALL_TYPE_LUDICROUS_BALL, //27
```

```
#define LAST_BALL_INDEX BALL_TYPE_LUDICROUS_BALL
#define NUM_BALLS (BALL_TYPE_LUDICROUS_BALL + 1)
```

- a. Declare the throwing image and palette with `extern const u8 gInterfaceGfx_<ball_name>Tiles[];` and `extern const u8 gInterfaceGfx_<ball_name>Pal[];`

```
extern const u8 gInterfaceGfx_LudicrousBallTiles[];
extern const u8 gInterfaceGfx_LudicrousBallPal[];
```

- b. Any new opening particles can be declared underneath the image/pal declarations,

eg. `extern const u8 gBattleAnimSpriteSheet_Particles<your_particle>Tiles/Pal[];`

You can define the desired starting frame (for particle animations) similar to

```
#define BALL_OPEN_PURPLE_CIRCLES 2
#define BALL_OPEN_BLUE_STICKS 0
```

`BALL_OPEN_WHITE_STARS`

```
extern const u8 gBattleAnimSpriteSheet_ParticlesLudicrousBallTiles[];
extern const u8 gBattleAnimSpriteSheet_ParticlesLudicrousBallPal[];
```

- c. Ball Tag – add `GFX_TAG_<ball_name>` at the end of *enum BallTags*
- d. Ball Particle Tags – add

```
GFX_TAG_DREAMBALL,
GFX_TAG_LUDICROUS_BALL,
};
```

`TAG_BALL_OPEN_<ball_name>` at the end of *enum BallOpenParticleTags*

- e. Add the sprite sheet info to `gBallSpriteSheets` following the same format as the other entries

```
TAG_BALL_OPEN_DREAM,
TAG_BALL_OPEN_LUDICROUS,|
};
```

- f. Add the sprite palette info to `gBallSpritePalettes`, again following the same table format.

```
[BALL_TYPE_DREAM_BALL] = {gInterfaceGfx_DreamBallTiles, (16 * 48) / 2, GFX_TAG_DREAMBALL},
[BALL_TYPE_LUDICROUS_BALL] = {gInterfaceGfx_LudicrousBallTiles, (16 * 48) / 2, GFX_TAG_LUDICROUS_BALL},
};
```

- g. Add the sprite template info to `gBallSpriteTemplates`, emulating another table entry. The *tileTag* and *paletteTag* should be the only elements that change.

```
[BALL_TYPE_DREAM_BALL] = {gInterfaceGfx_DreamBallPal, GFX_TAG_DREAMBALL},
[BALL_TYPE_LUDICROUS_BALL] = {gInterfaceGfx_LudicrousBallPal, GFX_TAG_LUDICROUS_BALL},
};
```

- h. Add an entry to the particle sprite sheet, `gBallOpenParticleSpritesheets`, following the format of the other entries.

```
[BALL_TYPE_LUDICROUS_BALL] =
{
    .tileTag = GFX_TAG_LUDICROUS_BALL,
    .paletteTag = GFX_TAG_LUDICROUS_BALL,
    .oam = sBallOamData,
    .anim = sBallAnimSequences,
    .images = NULL,
    .affineAnim = sBallAffineAnimSequences,
    .callback = SpriteCB_TestBallThrow,
},
```

gBattleAnimSpriteSheet_Particles is used for any existing opening particles (eg. the sticks/stars defined in *enum BallOpenParticles*). If you've included custom opening particles, assign them similar to the dusk ball or heal ball.

```
[BALL_TYPE_DREAM_BALL] = {gBattleAnimSpriteSheet_Particles, (8 * 64) / 2, TAG_BALL_OPEN_DREAM},
[BALL_TYPE_LUDICROUS_BALL] = {gBattleAnimSpriteSheet_Particles, (8 * 64) / 2, TAG_BALL_OPEN_LUDICROUS},|
```

- i. Add an entry for the particle palette table, *gBallOpenParticlePalettes*, again following the table format. *gBattleAnimSpritePalette_136* is for the given opening particles (sticks, stars, etc). Custom particle palettes can be defined similar to the dusk or heal ball.

```
[BALL_TYPE_DREAM_BALL] = RepeatBallOpenParticleAnimation,
[BALL_TYPE_LUDICROUS_BALL] = TimerBallOpenParticleAnimation,
```

- j. Add an entry to *gBallOpenParticleAnimNums*, linking your ball ID to the opening particle. It is easiest to assign existing particles, eg. *BALL_OPEN_STICKS*. Custom particles are defined similar to the dusk or heal ball, eg. follow *BALL_OPEN_PURPLE_CIRCLES* and its definitions/uses.

```
[BALL_TYPE_DREAM_BALL] = BALL_OPEN_HEARTS,
[BALL_TYPE_LUDICROUS_BALL] = BALL_OPEN_BLUE_STICKS,|
```

- k. Create a particle trajectory animation entry in *gBallOpenParticleAnimationFuncs*. The existing animation trajectories are hard to describe, just test them yourself.



- l. Add an entry to the *gBallOpenMonFadePal* table, which is the color the Pokémon fades to when it exits a Poké Ball. You can play around with RGB values in paint to find one you like. Keep the definitions after *BALL_TYPE_DREAM_BALL* at the end.

```
[BALL_TYPE_DREAM_BALL] = RGB(31, 12, 20), //Dream Ball - Deep Pink
[BALL_TYPE_LUDICROUS_BALL] = RGB(255, 255, 255), //Ludicrous Ball - White
RGB(0, 0, 0), //No idea what these lower values are for
RGB(1, 16, 0),
RGB(3, 0, 1),
RGB(1, 8, 0),
RGB(0, 8, 0),
RGB(3, 8, 1),
RGB(6, 8, 1),
RGB(4, 0, 0),
```

- m. Create a sprite template entry in *gBallParticleSpriteTemplates* following the structure of one of the new balls such as Dusk Ball or Dream Ball. Again, your *tileTag* and *paletteTag* should be the only differences between entries.

```
{ //GFX TAG_LUDICROUS_BALL
    .tileTag = TAG_BALL_OPEN_LUDICROUS,
    .paletteTag = TAG_BALL_OPEN_LUDICROUS,
    .oam = gUnknown_083AC9C8,
    .anim = gUnknown_0840C050,
    .images = NULL,
    .affineAnim = gDummySpriteAffineAnimTable,
    .callback = SpriteCallbackDummy,
},|
```

4. Ball Catch Multiplier

a. Open **src/catching.c** and find the function, **atkEF_handleballthrow**.

b. Add a *case* for your new ball inside the **switch (ballType)** **switch (ballType)** statement to include logic for your specific ball catch multiplier. A *ballMultiplier* value of 10 corresponds to a 1x catch multiplier, as the value is divided by 10 later on. You can also add or subtract values by modifying the variable *catchRate*, as in the *Heavy Ball's* case (it's weird - don't do it).

```
case BALL_TYPE_LUDICROUS_BALL:  
    ballMultiplier = 200;  
    catchRate += 50;  
    break;
```

5. You have now fully defined your new Poké Ball! You can now assign it to specific trainer's Pokémon (eg. Battle Frontier). You will still need to create the item data if you want the player to be able to use it. This is entirely up to the user, as this engine does not directly modify item data.

6. Please do not make a ball named Ludicrous Ball 😊

Items

NOTE: Defining a new battle item is easily done with this engine. The logic for the hold item effect is not generic and must be done by the user.

1. Define an Item ID – add a define: `#define ITEM_<your_item_name>` inside **include/constants/items.h**. This is your item's index. Update `ITEMS_COUNT`

```
#define ITEM_EJECT_PACK 0x2C0
#define ITEM_ROOM_SERVICE 0x2C1
#define ITEMS_COUNT (ITEM_ROOM_SERVICE + 1)
```

2. Define a hold item effect index - `#define ITEM_EFFECT_<your_effect_name>` in **include/constants/hold_effects.h**. Update `ITEM_EFFECT_COUNT` as needed.
3. Create your new item – this is handled entirely by the user, as item data is untouched in this engine. Just make sure the item index (step 1) matches. Assign the item the hold effect defined in Step 2.

```
#define ITEM_EFFECT_ROOM_SERVICE 133
#define ITEM_EFFECT_COUNT (ITEM_EFFECT_ROOM_SERVICE + 1)
```

4. There is no hold effect information needed. The user must find an appropriate location to include logic for their new battle item effect. The function `ItemId_GetHoldEffect` is an effective way to obtain the Pokémon in question's hold item effect from its hold item ID. The same logic with adding new ability effects to switch statements applies here as well for item effects.

Code Files

Below is a list of all code files which can be found in **src**, along with features contained in each file. Each file's respective header file can be found in **include/new**.

File	Description
ability_battle_effects.c	Functions that introduce or modify battle effects via abilities or otherwise. Includes terrain effects and ability pop-up
accuracy_calc.c	Rewrites how accuracy is calculated, including all relevant abilities, effects, etc. Includes protection logic
attackcanceler.c	Handles any logic for discerning if a move can be used or is effective, eg. mold breaker, flinch status, truant, etc.
battle_anims.c	Functions and structures to modify attack animations.
battle_contoller_opponent.c	Handles the functions responsible for the user moving between battle menus, choosing moves, etc.
battle_script_util.c	General functions that aide in battle scripting via callasm.
battle_start_turn_start.c	Handles the logic for determining which pokemon attacks first. Also handles setting up battle data.
battle_strings.c	Modifies the strings displayed in battle.
battle_terrain.c	Functions responsible for checking/loading/removing battle terrain.
battle_transition.c	Handles the transition into battle, eg. trainer mugshots.
battle_util.c	General functions for aiding in battle logic for everything.
build_pokemon.c	Modifies the data that is set for generated pokemon, eg. for battle tower/frontier team generation and others.
catching.c	Handles the catch probability logic, expands pokeballs, etc.
character_customization.c	Functions for altering the player's sprite based on the current sprite/palette selections
cmd49.c	Handles a ton of battle logic at the end of each turn
damage_calc.c	Functions responsible for calculating damage,

	including modifications from abilities, effects, etc...
daycare.c	Functions that handle all daycare functions, including attribute inheritance and step counts.
dexnav.c	Functions for the simplified dexnav system.
dns.c	Handles functions and palette changes for the day, night, and seasons feature.
dynamic_ow_pals.c	Handles the dynamic loading and tracking of overworld sprite palettes.
end_battle.c	Handles all battle termination logic and data resetting/saving.
end_turn.c	Handles all effects that happen at the end of each turn.
evolution.c	Handles old and new evolution methods.
exp.c	Functions that handle the exp share and exp gain.
follow_me.c	Functions relating to an NPC following around the player.
form_change.c	Functions/structures/arrays that handle species that change form.
frontier.c	All supporting and master functions for developing a battle frontier.
frontier_records.c	Handle player's status in the battle frontier.
general_bs_commands.c	Functions that support the battle scripting in assembly/battle_scripts.
item.c	Handles all item related functions, such as returning hold effects, tm/hm expansion, etc.
item_battle_effects.c	Handles functions that deal with battle effects of specific items.
learn_move.c	Handles functions for pokemon trying to learn moves.
link.c	Handles data transfer.
mega.c	Functions that support mega evolution logic and execution.
move_menu.c	Functions for displaying move data and z moves in the battle menu.
multi.c	Handles partner battle logic.
new_bs_commands.c	Functions for any additional battle scripting commands that are used inside battle scripts.
options_menu.c	Functions for secondary options menu.
overworld.c	Functions for anything regarding the overworld, such as trainer spotting, whiteout,

	step counters, etc.
party_menu.c	Handles anything related to the party menu, such as field moves, new party menu GUIs, etc...
pokemon_storage_system.c	Handles pokemon storage expansion and related functions
read_keys.c	Emulated JPANs keypad hacks, allowing the designer to force key presses, prevent them, or map functions onto them, among other uses.
roamer.c	Handles roaming pokemon and relevant functions/hooks.
save.c	Handles save block expansion functions/structures.
scripting.c	Handles all scripting specials or other functions associated with scripts.
set_effect.c	Handles move effects.
set_z_effect.c	Handles the logic for z move effects, including special z moves.
start_menu.c	Functions to redo how the start menu is generated, and associated functions such as safari steps/ball count.
stat_buffs.c	Adjusts stat-related functions to include abilities and effects that change their operations (eg. contrary).
switching.c	Handles battle switching logic.
trainer_sliding.c	Handles mid-battle trainer sliding and related message.
util.c	General utility functions.
wild_encounter.c	Handles functions related to wild encounter probability and associated features.

The following files can be found in **src/Battle_AI** are contain code for the updated battle AI.

File	Description
ai_advanced.c	Advanced logic for the AI, including move prediction and fight classes
ai_master.c	The master function(s) for the AI logic
ai_negatives.c	All possible subtractions to an AIs move viability.
ai_partner.c	Partner AI logic function(s)
ai_positives.c	All possible additions to an AIs move viability
ai_util.c	Commonly used functions in AI logic

The following files can be found in **src/Tables** and contain various data tables for the user to modify as they see fit.

File	Description
attack_data_table.c	Contains data for all the different battle moves.
ball_graphics_tables.c	Contains data for new in-battle Poke Ball graphics.
class_based_poke_ball_table.c	Used in conjunction with the definition TRAINER_CLASS_POKE_BALLS to assign unique Poké Balls to each trainer class.
experience_tables.c	Contains larger Exp yield data for each species (see GEN_7_BASE_EXP_YIELD). Also contains experience per level tables to help account for a possible increased max level.
frontier_trainers.c	Contains data for various trainers which can appear in <i>Battle Facilities</i> .
learnsets.c	Contains data for the level-up movepool of each Pokémon to support the increased number of moves the engine has to offer.
music_tables.c	Contains data to set trainer encounter music by class (see ENCOUNTER_MUSIC_BY_CLASS), battle music by class, and battle music by wild species.
pickup_items.c	Contains the different items that can be picked up using the ability Pickup .
pokemon_tables.c	Contains data tables for alternate species height and weights (for example, if a mega form is a different size), and several Pokémon ban lists.
terrain_table.c	Contains data for each battle background and how it influences various effects in battle such as <i>Camouflage</i> , <i>Nature Power</i> , <i>Secret Power</i> , and <i>Burmy</i> .
wild_encounter_tables.c	Contains data for setting up time-based wild encounters and swarms .

The following files can also be found in **src/Tables** and also contain various data tables for the user to modify, however, as they are header files, a change must be made in the file that includes them in order for them to recompile without cleaning the whole engine.

File	Description
battle_tower_spreads.h	Set up EVs, IVs, ability types, items, moves, ball type, etc, for trainers in the battle tower.

frontier_multi_spreads.h	Set up battle frontier multi battle spreads: EVs, IVs, ability type, item, moves, pokeball type, and more!
frontier_special_trainer_spreads.h	Set up battle frontier species for special trainers, including EVs, IVs, nature, ability, ball type, etc!
frontier_trainer_names.h	Set up tables of names for battle frontier trainers
trainers_with_evs_table.h	Defining the EV/IV/ability/ball type to be assigned to trainer pokemon

The following files can be found in **assembly/data** and contain various data tables for the user to modify as they see fit.

File	Description
ability_description_table.s	Contains a table with pointers to the different ability descriptions.
ability_tables.s	Contains several tables with abilities mainly used by the battle engine to see if certain abilities are in a given list.
anim_background_table.s	Contains a table of all the in-battle animation backgrounds.
anim_backgrounds_graphics_defines.s	Contains defines for assembly/data/Anim_Background_Table.s
anim_particle_graphics_defines.s	Contains defines for assembly/data/Anim_Particle_Table.s
anim_particle_table.s	Contains a table of all the in-battle attack particles.
attack_anim_table.s	Contains a table and data for all the new attack animations.
attack_description_table.s	Contains a table with pointers to the attack descriptions.
battle_script_commands_table.s	Contains both the original battle scripting command table, as well as a second battle scripting command table.
item_tables.s	Tables with items that are checked during the execution of certain moves.
move_effect_table.s	Contains a table with the battle scripts for each move effect, as well as certain tables of move effects which are used by the AI.
move_tables.s	Contains several tables with moves mainly used by the battle engine to see if certain moves are in a given list.
special_battle_anim_table.s	Contains a table and data for special in-battle

	animations.
trainer_backsprite_table.s	Contains various tables relating the trainer backsprites shown at the start of battle.
type_tables.s	Contains tables for move type effectiveness and graphics data for the different type icons.
z_move_name_table.s	Contains a table with pointers to the names of all the different Z-Moves.

The following files can be found in **assembly/battle_scripts** and contain several different scripts that run in-battle.

File	Description
ability_battle_scripts.s	Contains battle scripts for various ability effects.
attackcanceler_battle_scripts.s	Contains battle scripts for the battle scripting command <i>attackcanceler</i> . See src/attackcanceler.c .
battle_start_turn_start_battle_scripts.s	Contains battle scripts that can run at the beginning of the battle or each turn. See src/battle_start_turn_start.c .
cmd49_battle_scripts.s	Contains battle scripts for the battle scripting command <i>cmd49 (moveend)</i> . See src/CMD49.c .
et_battle_scripts.s	Contains battle scripts that can run at the end of each round. See src/end_turn.c .
fainting_battle_scripts.s	Contains the battle scripts that run when a Pokémon faints.
general_attack_battle_scripts.s	Contains various battle scripts for the different move effects.
item_battle_scripts.s	Contains various battle scripts for item effects. See src/item_battle_effects.c .
mega_battle_script.s	Contains battle scripts for Mega Evolution, Primal Reversion, and Ultra Burst. See src/mega.c .
move_menu_battle_scripts.s	Contains battle scripts which print strings for the player while they are selecting moves. See src/move_menu.c .
set_effect_battle_scripts.s	Contains battle scripts for various secondary move effects. See src/set_effect.c .
standard_damage_battle_script.s	Contains battle scripts that are called for nearly every damaging move and moves that fail.

switch_battle_scripts.s	Contains battle scripts that run when Pokémon are switched out. See src/switching.c .
trainer_sliding_battle_scripts.s	Contains battle scripts for when the opposing trainer sliding back onto the screen to give a message. See src/trainer_sliding.c .
z_effect_battle_scripts.s	Contains battle scripts for various secondary Z-Status move effects. See src/set_z_effect.c .

The following files can be found in **assembly/overworld_scripts** and contain a few different scripts that run in the overworld.

File	Description
system_scripts.s	Contains scripts that are run by the engine rather than as an event script (such as overworld poison, repels, etc.).
trainer_battle_scripts.s	Contains scripts that help initiate new types of trainer battles.

The following files can be found in **assembly/hooks** and contain different sets of hooks for updated functions.

File	Description
bag_expansion_hooks.s	Contains hooks relating to expanding the bag size. See src/item.c .
follow_me_hooks.s	UNUSED
general_hooks.s	Contains various general hooks that didn't fit into one specific category.
illusion_hooks.s	Contains hooks for the ability Illusion . See src/ability_battle_effects.c .
mega_hooks.s	Contains hooks for Mega Evolution, Primal Reversion, and Ultra Burst. See src/mega.c .
multi_hooks.s	Contains hooks that help implement <i>multi battles</i> against two opponents. See src/multi.c and src/battle_controller_opponent.c .
multi_partner_hooks.s	Contains hooks that help implement <i>multi battles</i> with a partner See src/multi.c .
pokemon_data_hooks.s	Contains hooks for the alteration of various Pokémon data.

	See src/build_pokemon.c .
--	----------------------------------

The following files can be found in **assembly** and contain different assembly routines.

File	Description
main.s	Similar to assembly/hooks/general_hooks.s , this file was mainly used by ghouslash , while the former was mainly used by Skeli .
pokedex_screen_stats.s	Contains a routine the replace the Pokédex size comparison with a screen showing the stat values of the Pokémon.
rtc.s	Contains a routine originally created by prime-dialga that implements the real-time clock using whatever hardware the emulator is running on.
thumb_compiler_helper.s	Contains various functions to helper the C compiler.

Table Compendium

Below is a list of all tables/arrays found in the engine that may be modified:

Table Name	File Name	Description
gAbilityDescriptions	assembly/data/ability_description_table.s	Pointers to descriptions of each ability, ordered by ability index
gRolePlayBannedAbilities	assembly/data/ability_tables.s	Banned abilities to be copied by <i>Role Play</i>
gRolePlayAttackerBannedAbilities	assembly/data/ability_tables.s	Banned abilities for the user of <i>Role Play</i>
gSkillSwapBannedAbilities	assembly/data/ability_tables.s	Banned abilities for <i>Skill Swap</i>
gWorrySeedGastroAcidBannedAbilities	assembly/data/ability_tables.s	<i>Worry Seed</i> and <i>Gastro Acid</i> ban list
gEntrainmentBannedAbilitiesAttacker	assembly/data/ability_tables.s	<i>Entrainment</i> attacker ability ban list
gEntrainmentBannedAbilitiesTarget	assembly/data/ability_tables.s	<i>Entrainment</i> target ability ban list
gSimpleBeamBannedAbilities	assembly/data/ability_tables.s	<i>Simple Beam</i> ban list
gReceiverBannedAbilities	assembly/data/ability_tables.s	<i>Receiver</i> ban list
gTraceBannedAbilities	assembly/data/ability_tables.s	<i>Trace</i> ban list
gAnimationBackgrounds	assembly/data/anim_background_table.s	Pointer to image, palette, and map for each animation background, ordered by background index
gBattleAnimPicTable	assembly/data/anim_particle_table.s	Images of animation particles
gBattleAnimPaletteTable	assembly/data/anim_particle_table.s	Palettes of animation particles
AttackAnimationTable	assembly/data/attack_anim_table.s	Attack animations for each move (ordered by move index)
AttackDescriptionTable	assembly/data/attack_description_table.s	Descriptions for each attack
gBattleScriptingCommandsTable	assembly/data/battle_script_command_table.s	Battle script commands
gBattleScriptingCommandsTable2	assembly/data/battle_script_command_table.s	Extra battle script commands
gNaturalGiftTable	assembly/data/item_tables.s	Natural gift effects
ConsumableItemEffectTable	assembly/data/item_tables.s	List of consumable items
gBattleScriptsForMoveEffects	assembly/data/move_effect_tables.s	Battle scripts for each move effect
gSetStatusMoveEffects	assembly/data/move_effect_tables.s	Move effects that set a status
gStatLoweringMoveEffects	assembly/data/move_effect_tables.s	Move effects that lower a stat
gConfusionMoveEffects	assembly/data/move_effect_tables.s	Move effects that confuse
gMoveEffectsThatIgnoreWeaknessResistance	assembly/data/move_effect_tables.s	Move effects that ignore weakness or resistance
gSheerForceBoostedMoves	assembly/data/move_tables.s	Moves boosted by <i>Sheer Force</i>
gRecklessBoostedMoves	assembly/data/move_tables.s	Moves boosted by <i>Reckless</i>
gPunchingMoves	assembly/data/move_tables.s	List of <i>Punching</i> moves
gPulseAuraMoves	assembly/data/move_tables.s	List of <i>Pulse</i> and <i>Aura</i> moves
gBitingMoves	assembly/data/move_tables.s	List of <i>Biting</i> moves
gBallBombMoves	assembly/data/move_tables.s	List of <i>Ball</i> and <i>Bomb</i> moves
gDanceMoves	assembly/data/move_tables.s	List of <i>Dance</i> moves
gGravityBannedMoves	assembly/data/move_tables.s	List of moves that can't be used under intense <i>Gravity</i> .

gMeFirstBannedMoves	assembly/data/move_tables.s	List of moves <i>Me-First</i> can't call.
gCopycatBannedMoves	assembly/data/move_tables.s	List of moves <i>Copycat</i> can't call.
gInstructBannedMoves	assembly/data/move_tables.s	List of moves <i>Instruct</i> can't call.
gMetronomeBannedMoves	assembly/data/move_tables.s	List of moves <i>Metronome</i> cannot become
gAssistBannedMoves	assembly/data/move_tables.s	List of moves <i>Assist</i> can't call
gMimicBannedMoves	assembly/data/move_tables.s	List of moves <i>Mimic</i> can't copy
gPowderMoves	assembly/data/move_tables.s	List of <i>Powder</i> -based moves
gSoundMoves	assembly/data/move_tables.s	List of <i>Sound</i> -based moves
gSubstituteBypassMoves	assembly/data/move_tables.s	List of moves that bypass <i>Substitute</i>
gTypeChangeExceptionMoves	assembly/data/move_tables.s	List of moves that can't have their types changed by <i>Electrify</i>
gSkyBattleBannedMoves	assembly/data/move_tables.s	List of moves banned in <i>Sky Battles</i>
gIgnoreStatChangesMoves	assembly/data/move_tables.s	List of moves that ignore stat changes
gHighCriticalChanceMoves	assembly/data/move_tables.s	List of moves with a high critical hit chance
gAlwaysCriticalMoves	assembly/data/move_tables.s	List of moves that always land a critical hit
gSleepTalkBannedMoves	assembly/data/move_tables.s	List of moves <i>Sleep Talk</i> can't call
gMovesThatCallOtherMoves	assembly/data/move_tables.s	List of moves and that call other moves
gMovesThatRequireRecharging	assembly/data/move_tables.s	List of moves and that require recharging
gAlwaysHitWhenMinimizedMoves	assembly/data/move_tables.s	List of always and hit when that target is minimized
gMoldBreakerMoves	assembly/data/move_tables.s	List of moves that ignore the target's ability like <i>Mold Breaker</i>
gFlinchChanceMoves	assembly/data/move_tables.s	List of moves that have a chance to flinch the target
gParentalBondBannedMoves	assembly/data/move_tables.s	List of moves that only strike once with <i>Parental Bond</i>
gMovesCanUnfreezeAttacker	assembly/data/move_tables.s	List of moves and can unfreeze the attacker
gMovesCanUnfreezeTarget	assembly/data/move_tables.s	List of moves and can unfreeze the target
gMovesThatChangePhysicality	assembly/data/move_tables.s	List of moves that are either <i>Physical</i> or <i>Special</i> depending on the higher stat
gTwoToFiveStrikesMoves	assembly/data/move_tables.s	List of two-five strikes moves
gTwoStrikesMoves	assembly/data/move_tables.s	List of two strikes moves
gThreeStrikesMoves	assembly/data/move_tables.s	List of three strikes moves
gPercent25RecoilMoves	assembly/data/move_tables.s	Moves that recoil 25% HP.
gPercent33RecoilMoves	assembly/data/move_tables.s	Moves that recoil 33% HP.
gPercent50RecoilMoves	assembly/data/move_tables.s	Moves that recoil 50% HP.
gPercent66RecoilMoves	assembly/data/move_tables.s	Moves that recoil 66% HP.
gPercent75RecoilMoves	assembly/data/move_tables.s	Moves that recoil 75% HP.
gPercent100RecoilMoves	assembly/data/move_tables.s	Moves that recoil 100% HP.

gIgnoreInAirMoves	<i>assembly/data/move_tables.s</i>	List of moves that ignore when are Pokémon mid-air
gIgnoreUndergroundMoves	<i>assembly/data/move_tables.s</i>	List of moves that ignore when are Pokémon are underground
gIgnoreUnderwaterMoves	<i>assembly/data/move_tables.s</i>	List of moves that ignore when are Pokémon are underwater
gAlwaysHitInRainMoves	<i>assembly/data/move_tables.s</i>	List of moves that always hit in rain
gSpecialAttackPhysicalDamageMoves	<i>assembly/data/move_tables.s</i>	List of <i>Special</i> moves that do physical damage
gSpecialWholeFieldMoves	<i>assembly/data/move_tables.s</i>	List of moves that “target” the user but really affect the whole field
gAromaVeilProtectedMoves	<i>assembly/data/move_tables.s</i>	List of moves blocked by <i>Aroma Veil</i>
gMovesThatLiftProtectTable	<i>assembly/data/move_tables.s</i>	List of moves that lift protect
gBattleAnims_General	<i>assembly/data/special_battle_anims_table.s</i>	List of special battle animations that can be called from the <i>playanimation</i> battle script command
gTypeEffectiveness	<i>assembly/data/type_tables.s</i>	Type effectiveness table. The format goes: {[attacking type], [defending type], [damage multiplier]}
gMoveMenuInfoIcons	<i>assembly/data/type_tables.s</i>	Type icon sizes
gZMoveNames	<i>assembly/data/z-move_name_table.s</i>	List of Z-Move names
sSmartWildAITable	<i>src/battle_ai/ai_master.c</i>	Link specific **Wild** species with intelligent AI usage. Terminate the table with {0xffff, 0}
gTrainerBackPicPaletteTable	<i>src/tables/back_pic_tables.c</i>	Link trainer back pic indices to specific palettes. Follow the given format.
gTrainerBackAnimsPtrTable	<i>src/tables/back_pic_tables.c</i>	Table for the animation frames for each backsprite
gTrainerBackPicCoords	<i>src/tables/back_pic_tables.c</i>	Offset of each backsprite
gSpriteTemplateTable_TrainerBackSprites	<i>src/tables/back_pic_tables.c</i>	Sprite templates for each backsprite
gBallSpriteSheets	<i>src/tables/ball_graphics_tables.c</i>	Throwing images for each Poké Ball type
gBallSpritePalettes	<i>src/tables/ball_graphics_tables.c</i>	Throwing palettes for each Poké Ball type
gBallSpriteTemplates	<i>src/tables/ball_graphics_tables.c</i>	Sprite templates for each Poké Ball type
gBallOpenParticleSpritesheets	<i>src/tables/ball_graphics_tables.c</i>	Poké Ball type particle images
gBallOpenParticlePalettes	<i>src/tables/ball_graphics_tables.c</i>	Poké Ball type particle palettes
gBallOpenParticleAnimNums	<i>src/tables/ball_graphics_tables.c</i>	Poké Ball particle animation numbers
gBallOpenParticleAnimationFuncs	<i>src/tables/ball_graphics_tables.c</i>	Poké Ball particle animation functions (aka particle trajectories)
gBallOpenMonFadePal	<i>src/tables/ball_graphics_tables.c</i>	Pokémon fade color by Poké Ball type (when exiting Poké Ball)

gBallParticleSpriteTemplates	src/tables/ball_graphics_tables.c	Sprite templates for Poké Ball particles
gBattleMoves	src/tables/battle_moves.c	Attack data for each Pokémon move
gClassPoké Balls	src/tables/class_based_poké_ball_table.c	Assign a Poké Ball type to each trainer class
gBaseExpBySpecies	src/tables/experience_tables.c	Base experience values for each species (gen 7)
gExperienceTables	src/tables/experience_tables.c	Experience values for each level
sPickupCommonItems	src/tables/item_tables.c	Common items in pickup table
sPickupRareItems	src/tables/item_tables.c	Rare items in pickup table
gFlingTable	src/tables/item_tables.c	Damage and effects from Fling
gConsumableItemEffects	src/tables/item_tables.c	Set of item effects that are consumable (see IsConsumable in src/end_battle.c)
gItemsByType	src/tables/item_tables.c	Sorting items by their type
gLevelUpLearnsets	src/tables/level_up_learnsets.c	Learnset pointers for each Pokémon
sPreBattleMugshotSprites	src/tables/mugshot_tables.c	Custom mugshot sprites. Mugshot image replaces the player sprite
sMugshotsBigPals	src/tables/mugshot_tables.c	Palettes for the Big Mugshot Style
sMugshotsDpPals	src/tables/mugshot_tables.c	Palettes for the DP Mugshot Style
sMugshotsTwoBarsPals	src/tables/mugshot_tables.c	Palettes for the Two Bar Mugshot Style
sMugshotPlayerPals	src/tables/mugshot_tables.c	Palettes for the Player Mugshot Style
gClassBasedTrainerEncounterBGM	src/tables/music_tables.c	Encounter music for each trainer class
gClassBasedBattleBGM	src/tables/music_tables.c	Battle music for each trainer class
gWildSpeciesBasedBattleBGM	src/tables/music_tables.c	Battle music for specific wild Pokémon species
gRandomBattleMusicOptions	src/tables/music_tables.c	Random battle music possibilities
gAlternateSpeciesSizeTable	src/tables/pokemon_tables.c	Defines the height and weight for alternate forms
gTelekinesisBanList	src/tables/pokemon_tables.c	Species that cannot be affected by Telekinesis
gSkyBattleBannedSpeciesList	src/tables/pokemon_tables.c	Species that cannot enter a Sky Battle
gUltraBeastList	src/tables/pokemon_tables.c	Ultra Beasts ids
gDeerlingForms	src/tables/pokemon_tables.c	Deerling form ids
gSawsbuckForms	src/tables/pokemon_tables.c	Sawsbuck form ids
gVivillonForms	src/tables/pokemon_tables.c	Vivillon form ids
gFlabebeForms	src/tables/pokemon_tables.c	Flabebe form ids
gFloetteForms	src/tables/pokemon_tables.c	Floette form ids
gFlorgesForms	src/tables/pokemon_tables.c	Florges form ids
gFurfrouForms	src/tables/pokemon_tables.c	Furfrou form ids
gPikachuCapForms	src/tables/pokemon_tables.c	Pikachu-Cap form ids
gSetPerfectXlvsList	src/tables/pokemon_tables.c	Species created with perfect lvs (if

		CREATE_WITH_X_PERFECT_IV S defined)
gRandomizerSpeciesBanList	src/tables/pokemon_tables.c	Species banned from being created by the randomizer
gRandomizerAbilityBanList	src/tables/pokemon_tables.c	Abilities banned from being assigned in the randomizer
gTerrainTable	src/tables/terrain_tables.c	Define parameters for each battle terrain
gAttackTerrainTable	src/tables/terrain_tables.c	Terrain move background info
gCamouflageColours	src/tables/terrain_tables.c	Colors for camouflage based on battle terrain
gTrainersWithEvsSpreads	src/tables/trainers_with_evs_table.h	Ev/Iv spreads to be assigned to enemy Pokémon
gWildMonMorningHeaders	src/tables/wild_encounter_tables.c	Assign morning wild Pokémon data by map
gWildMonEveningHeaders	src/tables/wild_encounter_tables.c	Assign evening wild Pokémon data by map
gWildMonNightHeaders	src/tables/wild_encounter_tables.c	Assign nighttime wild Pokémon data by map
gSwarmTable	src/tables/wild_encounter_tables.c	Swarm species by map
gAbilityRatings	src/ability_battle_effects.c	Rate abilities for Battle Frontier/random assignment
gMoldBreakerIgnoredAbilities	src/ability_battle_effects.c	Add abilities to be ignored by mold breaker
sImmunities	src/build_pokemon.c	Link immunities to specific types for building battle frontier party
sCharacterPalSwitchTable	src/character_customization.c	General character customization struct for Pokemon Unbound. Can emulate for your own game
gOverworldTableSwitcher	src/character_customization.c	Pointers for each overworld table
sPlayerAvatarGfxIds	src/character_customization.c	Object Ids for each player avatar state
gEndBattleFlagClearTable	src/end_battle.c	List of flags to clear at the end of each battle
gFollowerAlternateSprites	src/follow_me.c	Sprites to change the npc follower to in specific cases: {WALKING/RUNNING SPRITE ID, BIKING SPRITE ID, SURFING SPRITE ID}
gMiniorCores	src/form_change.c	Ids for Minior cores
sBannedBackupSpecies	src/form_change.c	Ids not to revert to after battle
sEntryHazardsStrings	src/general_bs_commands.c	Strings for each entry hazard
sKeystoneTable	src/mega.c	Item Ids for Mega Rings
sTypeIconPositions	src/move_menu.c	Position of healthbar type icons
sTypeIconPicTable	src/move_menu.c	Define sprites for healthbar type icons
gDefaultWalkingScripts	src/overworld.c	Pointers to custom walking scripts
sMetatileInteractionScripts	src/overworld.c	Scripts to run when interacting with certain tiles/behaviour bytes
sRoamerLocations	src/roamer.c	Maps that include roaming

		pokemon
gFossilImageTable	src/scripting.c	Include images to show with the fossil image hack
sMultichoiceSetX	src/scripting.c	Lists of strings for scrolling multichoice set X
gScrollingSets	src/scripting.c	Scrolling multichoice sets
sMoveEffectsThatIgnoreSubstitute	src/set_effect.c	Moves that pass through substitute
sShieldDustIgnoredEffects	src/set_effect.c	Effects that are ignored by <i>Shield Dust</i>
gTrappingMoves	src/set_effect.c	Wrap-type moves
gWrappedStringIds	src/set_effect.c	Strings to display on wrap effect
sSpecialZMoveTable	src/set_z_effect.c	Link special z moves to the respective species
sStartMenuItems	src/start_menu.c	Items of the start menu
sStartMenuDescriptionItems	src/start_menu.c	Descriptions for each start menu item
sTrainerSlides	src/trainer_sliding.c	Trainer sliding conditions/messages

Battle Tower/Frontier Tables

Table Name	File Name	Description
gFrontierSpreads	src/Tables/battle_tower_spreads.h	Basic enemy data for the <i>Battle Frontier</i>
gFrontierLegendarySpreads	src/Tables/battle_tower_spreads.h	Legendary enemy data for the <i>Battle Frontier</i>
gMiddleCupSpreads	src/Tables/battle_tower_spreads.h	Middle Cup Pokémon data for the Battle Frontier
gLittleCupSpreads	src/Tables/battle_tower_spreads.h	Little Cup Pokémon data for the Battle Frontier
gArceusSpreads	src/Tables/battle_tower_spreads.h	Pokémon spreads for Arceus
gPikachuSpreads	src/Tables/battle_tower_spreads.h	Pokémon spreads for Pikachu
gWormadamSpreads	src/Tables/battle_tower_spreads.h	Pokémon spreads for Wormadam
gRotomSpreads	src/Tables/battle_tower_spreads.h	Pokémon spreads for Rotom
gOricorioSpreads	src/Tables/battle_tower_spreads.h	Pokémon spreads for Oricorio
gTowerTrainers	src/Tables/battle_frontier_trainers.c	Trainer information for the <i>Battle Frontier</i>
gSpecialTowerTrainers	src/Tables/battle_frontier_trainers.c	<i>Special Trainer</i> info for Battle Frontier
gFrontierBrains	src/Tables/battle_frontier_trainers.c	Smart, advanced trainer info for <i>Battle Frontier</i>
gFrontierMultiBattleTrainers	src/Tables/battle_frontier_trainers.c	<i>Multi Battle Trainer</i> info for <i>Battle Frontier</i>
gMaleFrontierNamesTable	src/Tables/frontier_trainer_names.h	Array of male trainer names for <i>Battle Frontier</i>
gFemaleFrontierNamesTable	src/Tables/frontier_trainer_names.h	Array of female trainer names for <i>Battle Frontier</i>
gBattleTowerStandardSpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban for the standard battle tier
gGSCup_LegendarySpeciesList	src/Tables/Pokemon_Tables.c	Legendary species allowed to participate in the <i>GS Cup</i> tier
gSmogonOU_SpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban list for <i>Smogon OU</i> Singles
gSmogonOUDoubles_SpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban list for <i>Smogon OU</i> doubles
gSmogonOU_AbilityBanList	src/Tables/Pokemon_Tables.c	Ability ban list for <i>Smogon OU</i> singles
gSmogonOUDoubles_AbilityBanList	src/Tables/Pokemon_Tables.c	Ability ban list for <i>Smogon OU</i> doubles
gBattleTowerStandard_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban list for the standard tier

gSmogonOU_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban list for <i>Smogon OU</i> singles
gSmogonOUDoubles_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban list for <i>Smogon OU</i> doubles
gSmogonOUDoubles_MoveBanList	src/Tables/Pokemon_Tables.c	Moves ban list for <i>Smogon OU</i> doubles
gSmogon_MoveBanList	src/Tables/Pokemon_Tables.c	Move ban list for general <i>Smogon</i> tiers
gSmogonLittleCup_SpeciesList	src/Tables/Pokemon_Tables.c	Allowed species list for <i>Little Cup</i>
gSmogonLittleCup_MoveBanList	src/Tables/Pokemon_Tables.c	Move ban list for <i>Little Cup</i>
gSmogonLittleCup_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban list for <i>Little Cup</i>
gMiddleCup_SpeciesList	src/Tables/Pokemon_Tables.c	Allowed species list for <i>Middle Cup</i>
gMiddleCup_AbilityBanList	src/Tables/Pokemon_Tables.c	Ability ban list for <i>Middle Cup</i>
gMiddleCup_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban list for <i>Middle Cup</i>
gSmogonMonotype_SpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban list for <i>Monotype</i>
gSmogonMonotype_AbilityBanList	src/Tables/Pokemon_Tables.c	Ability ban list for <i>Monotype</i>
gSmogonMonotype_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban list for <i>Monotype</i>
gSmogonCamomons_SpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban list for <i>Camomons</i>
gSmogonAverageMons_SpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban list for <i>Averagemons</i>
gSmogonAverageMons_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban for <i>Averagemons</i>
gSmogonAverageMons_MoveBanList	src/Tables/Pokemon_Tables.c	Move ban list for <i>Averagemons</i>
gSmogonAverageMons_AbilityBanList	src/Tables/Pokemon_Tables.c	Ability ban list for <i>Averagemons</i>
gSmogon350Cup_SpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban list for <i>350 Cup</i>
gSmogon350Cup_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban list for <i>350 Cup</i>
gSmogon350Cup_AbilityBanList	src/Tables/Pokemon_Tables.c	Ability ban list for <i>350 Cup</i>
gSmogonScalemons_SpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban list for <i>Scalemons</i>
gSmogonScalemons_ItemBanList	src/Tables/Pokemon_Tables.c	Item ban list for <i>Scalemons</i>
gSmogonScalemons_AbilityBanList	src/Tables/Pokemon_Tables.c	Ability ban list for <i>Scalemons</i>
gSmogonBenjaminButterfree_SpeciesBanList	src/Tables/Pokemon_Tables.c	Species ban list for <i>Benjamin Butterfree</i>
gBattleTowerTiers	src/frontier.c	Tiers available in the <i>Battle Tower</i>
gBattleMineFormat1Tiers	src/frontier.c	Tiers available in <i>Battle Mine Fmt 1</i>
gBattleMineFormat2Tiers	src/frontier.c	Tiers available in <i>Battle Mine Fmt 2</i>
gBattleMineFormat3Tiers	src/frontier.c	Tiers available in <i>Battle Mine Fmt 3</i>
gBattleMineTiers	src/frontier.c	Tiers available in the overall <i>Battle Mine</i>
gBattleCircusTiers	src/frontier.c	Tiers available in the <i>Battle Circus</i>
gBattleFrontierTierNames	src/frontier.c	Names of <i>Battle Frontier</i> tiers
gBattleFrontierFormats	src/frontier.c	Names of <i>Battle Frontier</i> formats
sBattleCircusEffectDescriptions	src/frontier.c	Descriptions for <i>Battle Circus</i> effects

Engine Scripts

Clean.py

Clean.py's role is to clean up the repository by removing generated files. The following commands can be used from the command line:

Command	Description
<code>python scripts/clean.py</code>	Removes all object files (not including those from images), generated repoints, generated offsets, and generated roms.
<code>python scripts/clean.py all</code>	Removes all object files, generated repoints, generated offsets, and generated roms.
<code>python scripts/clean.py build</code>	Removes all object files (not including those from images), generated offsets, and generated roms.
<code>python scripts/clean.py graphics</code>	Removes all object and header files from images.
<code>python scripts/clean.py file FILE_PATH</code>	Removes only the object file for the given file path. Make sure "/" and not "\" is used to separate the directory names.

String.py

String.py's role is to compile *.string* files (found in the **strings** directory). The following rules apply to compiling *.string* files:

- Placing `MAX_LENGTH=XX` at the top of the file where `XX` is a number of your choosing will force all strings compiled in that file to have a maximum length of `XX` (not including the terminator character, `0xFF`).
- When used in conjunction with `MAX_LENGTH`, placing `FILL_FF=True` at the top of the file will force all strings shorter than `MAX_LENGTH` to have FFs appended onto the end to make them the correct size. This is how arrays of strings are made possible (such as *gMoveNames* and *gAbilityNames*).
- Each string name starts with the directive `#org @` and is followed by the title of the string. Several `#org`'s made be piled on top of one another to make several string defines point to the same string. For example:

```
#org @gAbilityNames
#org @NAME_ABILITY_NONE
-----
```

- The line after the `#org` contains the string. It can be written in plain text with no quotation marks on either side and with no terminator character at the end. The string may be spread onto multiple lines if you so wish. The string is terminated when the next `#org` is parsed.

- There are different escape characters which can be used with the string files:

Character	Becomes
\n	Newline
\p	New textbox (display arrow)
\l	Scroll line
\e	é
\\$	¢
\"	"

- Text buffers can be used too (see [SpecialBuffers](#) in **scripts/string.py** for all available):

Buffer	Description
[.]	...
[BUFFER]	Start a buffer. Usually followed by a hex buffer.
[ATTACKER]	IN-BATTLE: Loads the attacker's name (found in gBankAttacker).
[TARGET]	IN-BATTLE: Loads the target's name (found in gBankTarget).
[EFFECT_BANK]	IN-BATTLE: Loads the bank found in gEffectBank .
[SCRIPTING_BANK]	IN-BATTLE: Loads the bank found in gBattleScripting->bank .
[CURRENT_MOVE]	IN-BATTLE: Loads the move found in gCurrentMove .
[LAST_ITEM]	IN-BATTLE: Loads the item found in gLastUsedItem .
[LAST_ABILITY]	IN-BATTLE: Loads the ability found in gLastUsedAbility .
[ATTACKER_ABILITY]	IN-BATTLE: Loads the attacker's ability (from gBankAttacker).
[TARGET_ABILITY]	IN-BATTLE: Loads the target's ability (from gBankTarget).
[SCRIPTING_BANK_ABILITY]	IN-BATTLE: Loads the ability of gBattleScripting->bank .
[PLAYER_NAME]	IN-BATTLE: Loads the player's name.
[PLAYER]	OVERWORLD: Prints the player's name.
[RIVAL]	OVERWORLD: Prints the rival's name.
[WHITE]	OVERWORLD: Changes the text colour to white.
[BLACK]	OVERWORLD: Changes the text colour to black.
[GRAY]	OVERWORLD: Changes the text colour to gray.
[RED]	OVERWORLD: Changes the text colour to red.
[ORANGE]	OVERWORLD: Changes the text colour to orange.
[GREEN]	OVERWORLD: Changes the text colour to green.
[LIGHT_GREEN]	OVERWORLD: Changes the text colour to light green.
[BLUE]	OVERWORLD: Changes the text colour to blue.
[LIGHT_BLUE]	OVERWORLD: Changes the text colour to light blue.
[SHRINK]	OVERWORLD: Uses a smaller font for the following characters.
[ALIGN][XX]	OVERWORLD: Adds XX whitespace characters.
[PAUSE_UNTIL_PRESS]	OVERWORLD: Stops printing text until a key is pressed.
[XX] Eg. [FA] or [52]	Any two hex characters to represent that byte exactly (hex buffer). Can be used in conjunction with [BUFFER]. For example, in the overworld, [BUFFER][02] is the same as saying [buffer1] in XSE.

Credits

Main Contributors:

Skeli

Ghoulslash

Code:

Lixdel - Attack Animations

Pret - PokeRuby, PokeFireRed, PokeEmerald

Sagiri - Trainer Class Poke Balls, Pickup Update, Move Item, Summary Screen Wrapping

DizzyEgg - Emerald Battle Engine Upgrade V1 & V2, Dizzy's Emerald Hacked Engine

FBI - Expanded Saveblock, Dexnav

Touched – Follow Me, Mega Evolution source

Navenatox – Dynamic Overworld Palettes

Doesntknowhowtoplay & Squeetz - Pokedex Screen Stats

Azurile13 - Hidden Abilities

Squeetz - Footstep Noises

JPAN – Fire Red Hacked Engine Source

Diegoisawesome – Triple Layer Tiles

Jiangzhengwenjz – Linux compatibility

Graphics:

Golche - Attack Particles, Battle Backgrounds, Other Graphics

Criminon - Ultra Burst indicator

Bela - Poke Balls

Solo993 - Backsprites

canstockphoto.ca - Battle Backgrounds

Testers:

Criminon

Dionen

Gail

Recko Juice

Patrickz

If you actually read all the pages up until this point you deserve a medal. No, wait, a cookie. You deserve a cookie. Good job!



THE END