Save data structure (Generation III)

This article is incomplete.
Please feel free to edit this article to add missing information and complete it.

This article does not yet meet the quality standards of Bulbapedia. Please feel free to edit this article to make it conform to Bulbapedia norms and conventions.

The **save data structure** for Generation III is stored in the cartridge's battery-backed RAM chip (SRAM), or as a ".sav" file from most emulators. The structure consists of 128 KB of data, though not every byte is used. Emulators may append additional data for the purposes of maintaining real-time clock operations.

The integrity of most of the file is validated by checksums.

Data types

Unless otherwise noted, integer values occupy the specified number of bytes, and are little-endian and either unsigned or two's complement.

Text data is stored in a proprietary encoding. Strings in fixed-length fields are terminated with 0xFF with any remainder padded with 0x00.

File structure

The Generation III save file is broken up into two game save blocks, each of which is broken up into 4 KB sections. Each 4 KB section is independently validated by checksum.

One block of game save data represents the most recent game save, and the other block represents the previous game save.

- If any checksum for the most recent game save fails, the game will use the previous game save instead.
- If any checksum for the previous game save fails, the player will be forced to start a new game.

Offset	Size	Contents
0x000000	57344	Game save A
0x00E000	57344	Game save B
0x01C000	8192	Hall of Fame
0x01E000	4096	*Mystery Gift/e-Reader*
0x01F000	4096	*Recorded Battle*

Game save A, Game save B

14 sections representing a game save.

Exactly which game save is the most recent and which one is the previous depends on the relative values of the *save index* field.

The game will alternate which region of the save file it writes to each time the game is saved. For example, if the most recent save was *Game Save A*, then the next time the game is saved, it will be written to *Game Save B*.

Hall of Fame, Mystery Gift/e-Reader, Recorded Battle

The first time each of these features are used on a new game, the data is wiped. For example, When entering the Hall of Fame for the first time, the entire contents of its sections are set to 0x00.

Section format

All sections contain the same general format: 3968 bytes of data, followed by a footer.

Offset	Size	Contents
0x0000	3968	Data
0x0FF4	2	Section ID
0x0FF6	2	Checksum
0x0FF8	4	Signature
0x0FFC	4	Save index

Data

The checksum-able data within the section. The actual number of bytes checked is determined by *Section ID*.

Section ID

Specifies the save data being represented, as well as how many bytes to validate for the checksum. This is a 16-bit unsigned field and must have one of the following values:

ID	Size	Contents
0	3884	Trainer info
1	3968	Team / items
2	3968	Game State
3	3968	Misc Data
4	3848	Rival info
5	3968	PC buffer A
6	3968	PC buffer B
7	3968	PC buffer C
8	3968	PC buffer D
9	3968	PC buffer E
10	3968	PC buffer F
11	3968	PC buffer G
12	3968	PC buffer H
13	2000	PC buffer I

All 14 sections must be present exactly once in each game save block.

Signature

Each section contains 4 bytes in between the Checksum and Save Index. This serves as a file signature that holds the magic number **0x08012025** in little-endian format (i.e. bytes **0x25 0x20 0x01 0x08**). If the signature holds a different value, the section, and, by extension, the save block, is treated as invalid.

Checksum

Used to validate the integrity of saved data.

A 16-bit checksum generated by adding up bytes from the section. The algorithm is as follows:

- Initialize a 32-bit checksum variable to zero.
- Read 4 bytes at a time as 32-bit word (little-endian) and add it to the variable. The number of bytes to process in this manner is determined by *Section ID*.
- Take the upper 16 bits of the result, and add them to the lower 16 bits of the result.
- This new 16-bit value is the checksum.

Save index

Every time the game is saved, its *Save Index* value goes up by one. This is true even when starting a new game: it continues to count up from the previous save.

All 14 sections within a game save must have the same *Save Index* value. The most recent game save will have a greater *Save Index* value than the previous save.

Pokédex data

The Pokédex has multiple data units stored in various parts of the game save. *Editor's note: Research into this is still ongoing, but all testing has turned up the following results.*

RS			E			FRLG			
Section	**Offset**	**Size**	**Section**	**Offset**	**Size**	**Section**	**Offset**	**Size**	**Contents**
0	0x0019	2	0	0x0019	2	0	0x001B	1	National Pokédex A
0	0x0028	49	0	0x0028	49	0	0x0028	49	Pokédex owned
0	0x005C	49	0	0x005C	49	0	0x005C	49	Pokédex seen A
1	0x0938	49	1	0x0988	49	1	0x05F8	49	Pokédex seen B
2	0x03A6	1	2	0x0402	1	2	0x0068	1	National Pokédex B
2	0x044C	2	2	0x04A8	2	2	0x011C	2	National Pokédex C
4	0x0C0C	49	4	0x0CA4	49	4	0x0B98	49	Pokédex seen C

Pokédex owned, Pokédex seen

Represents the specific Pokédex entries that have been either seen or owned during gameplay.

It is not clear why there are three copies of which Pokémon have been seen (it may be related to how the game loads game save sections), but all three need to have the same data.

Pokémon are indexed by their usual Pokédex order, meaning the order is the same as in the National Pokédex. However, indexes begin counting at 0, rather than 1.

1 bit is used to represent whether a given Pokémon has been seen/owned. Bits are ordered within bytes from lowest to highest, starting with the first byte. Therefore, the exact bit can be extracted from the list using the following formula:

```
Bit = ( Data[ RoundDown(PokeNum / 8) ] / 2 ^ (PokeNum Mod 8) ) AND 1
```

Or in C-style code (shift occurs before other bitwise operations):

```
Bit = Data[PokeNum >> 3] >> (PokeNum & 7) & 1;
```

Example

Let us say that we want to know whether #286 Breloom has been seen/owned:

- PokeNum becomes 285, since it is 0-based.
- The byte of the list in which bit 285 is located is = 285 / 8 = 35
- The bit within that byte is = 285 Mod 8 = 5
- Dividing the byte value by 2 ^ Bit, or shifting right by Bit, moves the bit to the least-significant position
- Performing a bitwise AND with 1 will remove all but the least-significant bit

National Pokédex

Various fields are used to indicate that the National Pokédex has been unlocked, and all fields must be configured correctly. *Editor's note: It's not certain whether these rules always apply to 100% of game saves, but they have for every save I've come across.*

Field A

In Ruby, Sapphire and Emerald, these two bytes have the following values:

- 0x00 0x00 - National Pokédex is **not** unlocked
- 0x01 0xDA - National Pokédex is unlocked

In FireRed and LeafGreen, this one byte has the following values:

- 0x00 - National Pokédex is **not** unlocked
- 0xB9 - National Pokédex is unlocked

Field B

One bit in this field is used to determine whether the National Pokédex has been unlocked, as follows:

- Bit 6 - Ruby, Sapphire and Emerald
- Bit 0 - FireRed and LeafGreen

If the National Pokédex has been unlocked, the value of the bit will be 1.

Field C

In Ruby, Sapphire and Emerald, these two bytes have the following values:

- 0x00 0x00 - National Pokédex is **not** unlocked
- 0x02 0x03 - National Pokédex is unlocked

In FireRed and LeafGreen, these two bytes have the following values:

- 0x00 0x00 - National Pokédex is **not** unlocked
- 0x58 0x62 - National Pokédex is unlocked

Section 0 - Trainer Info

This section contains information regarding the player character.

Fields documented below simply list known values; there may be additional significant data in this section:

RS		E		FRLG		
Offset	**Size**	**Offset**	**Size**	**Offset**	**Size**	**Contents**
0x0000	7	0x0000	7	0x0000	7	Player name
0x0008	1	0x0008	1	0x0008	1	Player gender
0x0009	1	0x0009	1	0x0009	1	Unused/Unknown
0x000A	4	0x000A	4	0x000A	4	Trainer ID
0x000E	5	0x000E	5	0x000E	5	Time played
0x0013	3	0x0013	3	0x0013	3	Options
0x00AC	4	*N/A*		0x00AC	4	Game Code
N/A		0x00AC	4	0x0AF8	4	Security key

Player name

The name of the player. Represented as a string that can be from 1 to 7 characters in length.

Player gender

Specifies the gender of the player character:

- For boy characters, this is set to `0x00`
- For girl characters, this is set to `0x01`

Note that changing this value will not be reflected immediately on the character sprite when first loading the game. (This is because the game saves the state of the current map and the sprites, scripts, and tiles on it separately.) To update the sprite, the player must load into a new map first, such as entering or exiting a building.

Trainer ID

The player's internal Trainer ID.

- The lower 16 bits represent the visible, public ID.
- The upper 16 bits represent the hidden, Secret ID.

Time played

Specifies how much time has elapsed during gameplay.

This value is actually 4 values representing, in this order: the hours, minutes, seconds and "frames" that have elapsed. A frame is 1/60th of a second.

The *hours* field is 16-bit. The other three fields are 8-bit.

Options

These are the options that can be changed in-game via the Options menu entry.

The first byte (0x13) contains the Button Mode settings. The different settings correspond to these values:

- *Normal*RSE/*Help*FRLG: `0x01`
- *LR*: `0x02`
- *L=A*: `0x03`

The second byte (0x14) contains the settings for Text Speed and Frame. The lowest three bits designate the Text Speed and the remaining bits the index of the chosen Frame style. The options for Text speed result in these values in the save file:

- Slow: `0b00000000`
- Medium: `0b00000001`
- Fast: `0b000000010`

Since there are only three different Text Speeds selectable in-game but eight possible values in three bits, the effect of values outside this range is unclear.

The index of the chosen Frame can be found in the upper five bits. A gallery of all the frames can be found here. The indexes in the save file start at 0, so a Frame value of 15 results in Frame 16.

The third byte (0x15) contains the settings for Sound, Battle Style and Battle Scene. Each setting is represented by one bit, basically switching the setting from one value to the other.

- Sound: Bit 1. `0b00000000` for *mono* (the default) and `0b00000001` for *stereo*.
- Battle Style: Bit 2. `0b00000000` for *switch* (the default) and `0b00000010` for *set*.
- Battle Scene: Bit 3. `0b00000000` for *on* (the default) and `0b00000100` for *off*.

Game Code

Identifies which game this save data applies to.

- For Ruby and Sapphire, this value will be `0x00000000`.
- For FireRed and LeafGreen, this value will be `0x00000001`.
- For Emerald, this is the location of the *Security Key* field, so any value other than 0 or 1 can be used to identify Emerald.

Security key

Used to encrypt sensitive data, such as money or item quantities. Applied to data via a simple exclusive or operation.

Important Note: *The ultimate origin of this value is not well understood. It may be derived from other data, or it may just be a random number. The offsets listed in the table above are not known to be 100% guaranteed, but a copy of the value appears at offset 0x01F4 in Emerald and 0x0F20 in Fire Red/Leaf Green.*

Ruby and Sapphire either do not utilize this masking operation, or the mask is always zero. *Editor's Note: This could use some further investigation. If the mask does exist, it would be located at the same spots as in Emerald. In all of my testing, Ruby/Sapphire data has not been protected by a security key.*

Section 1 - Team / Items

This section contains information regarding the player's team of Pokémon as well as their item inventory.

Fields documented below simply list known values; there may be additional significant data in this section:

RS		E		FRLG		
Offset	**Size**	**Offset**	**Size**	**Offset**	**Size**	**Contents**
0x0234	4	0x0234	4	0x0034	4	Team size
0x0238	600	0x0238	600	0x0038	600	Team Pokémon list
0x0490	4	0x0490	4	0x0290	4	Money
0x0494	2	0x0494	2	0x0294	2	Coins
0x0498	200	0x0498	200	0x0298	120	PC items
0x0560	80	0x0560	120	0x0310	168	Item pocket
0x05B0	80	0x05D8	120	0x03B8	120	Key item pocket
0x0600	64	0x0650	64	0x0430	52	Ball item pocket
0x0640	256	0x0690	256	0x0464	232	TM Case
0x0740	184	0x0790	184	0x054C	172	Berry pocket

Team size

The number of Pokémon currently on the team.

Team Pokémon list

Data representing information for up to 6 Pokémon, as an array. For the format, please refer to: Pokémon data structure (Generation III).

All 100 bytes of the Pokémon record are used.

Data representing Pokémon beyond the team size are dummied out with byte value `0x00`.

Money

The amount of money held by the player. Must be XORed with the security key to yield the true value.

Coins

The number of coins in the Coin case. Must be XORed with the lower two bytes of the security key to yield the true value.

Item pockets

Lists of item entries representing the player's item inventory for the various Bag pockets.

The number of item entries available per pocket in each game can be calculated by taking the number of bytes from the table above, and dividing by 4, which is the size of the item entry structure. The inventory limits are as follows:

	PC	Pocket	Key Item	Ball	TM/HM	Berry
RS	50	20	20	16	64	46
E	50	30	30	16	64	46
FRLG	30	42	30	13	58	43

Item entry

Item entries are defined using the following structure:

Offset	Size	Contents
0x00	2	Item index
0x02	2	Item quantity

Item index

The item's index.

Item quantity

The number of this item available in the Bag. Must be XORed with the lower 16 bits of the security key to yield the true value.

The security key does NOT apply to items stored in the PC.

Section 2 - Game State

This section contains many flags and variables associated with the state of the game, from daily step counts, berry tree, and secret base information, to plot advancement flags and the state of scripting events across the game.

RS		E		
Offset	**Size**	**Offset**	**Size**	**Contents**
0x0408(?)	4	0x0464	4	Mirage Island value

Note that only the low two bytes of the Mirage Island value is used. (Little endian)

Section 3-4 - Game Specific Data

This section contains more information regarding game state. The data in this region varies by game. In RSE, this section has lots of data used with record mixing, secret bases, interviewer data, Trainer Hill, mail, etc.

In FRLG, this section has data regarding the rival character.

Fields documented below simply list known values; there may be additional significant data in this section:

RSE		FRLG		
Offset	**Size**	**Offset**	**Size**	**Contents**
N/A		0x0BCC	8	Rival name

Rival name

The name of the rival. Represented as a string that can be from 1 to 7 characters in length.

Sections 5-13 - PC buffer

These sections contain information regarding the contents of the Pokémon Storage System.

Although the PC buffer occupies 9 sections within the game save, it still functions like a single section in and of itself: it's like one big section that was split apart across multiple sections.

The bytes within the PC buffer's component sections are contiguous within it. That is to say, the 3968 bytes occupied by section 5 represent bytes 0 to 3967 in the PC buffer, the 3968 bytes in section 6 represent bytes 3968 to 7935 in the PC buffer, and so-on.

Altogether, the PC buffer contains 3968 bytes from each of 8 sections and 2000 bytes from 1 section, for a total of 33744 bytes.

RSEFRLG		
Offset	**Size**	**Contents**
0x0000	4	Current PC Box
0x0004	33600	PC Boxes Pokémon list
0x8344	126	Box names
0x83C2	14	Box wallpapers

Current PC Box

The most recently viewed PC box, minus 1. That is to say, 0 represents Box 1 and 13 represents Box 14.

PC Boxes Pokémon list

A list of 420 Pokémon records. For the format, please refer to: Pokémon data structure (Generation III).

Only 80 bytes of the Pokémon record are used, meaning everything after the substructures is not included. Instead, those values are regenerated upon withdrawing a Pokémon from the PC. This is the basis of the Box trick.

Records are ordered as left-to right, top-to-bottom, per box. That is to say, the first 6 records represent the top row of Box 1 (boxes are 5 rows by 6 columns), and the first 30 records represent Box 1.

Empty cells within boxes simply dummy records containing nothing but byte value `0x00`.

Box names

The 14 box names, 9 bytes apart. Represent strings that can be from 1 to 8 characters in length.

Box wallpapers

The 14 box wallpapers, 1 byte each.

Hall of Fame

Hall of Fame Record

The Hall of Fame consists of an array of 50 Pokémon teams that have beaten the champion, oldest record first. This data is kept separate from the two save game blocks mentioned above. Each team is an array of 6 Pokémon. The Hall of Fame does not use the normal Pokémon data structure, and instead uses its own 20-byte structure, described below:

RSEFRLG		
Offset	**Size**	**Contents**
0x0000	4	Trainer ID
0x0004	4	Personality
0x0008	2	Species (bits 0-8) / Level (bits 9-15)
0x000A	10	Nickname

One Hall of Fame record is 120 bytes (6 Pokémon * 20 bytes). All 50 Hall of Fame records take 6000 bytes.

Hall of Fame Sections

The Hall of Fame data consists of two sections. Each section holds only 3968 bytes. The first section holds the first 33 records and the first 8 bytes off the 34th, or the first 3968 bytes of the array. The second section holds the rest of the 34th record and the remaining records, or the next 2032 bytes of the array. The remaining 1936 bytes of the second section are unused.

Data structure in the Pokémon games

Generation I	Pokémon species • Pokémon • Poké Mart • Character encoding • Save
Generation II	Pokémon species • Pokémon • Trainer • Character encoding • Save
Generation III	Pokémon species (Pokémon evolution • Pokédex • Type chart) Pokémon (substructures) • Move • Contest • Contest move • Item Trainer Tower • Battle Frontier • Character encoding • Save
Generation IV	Pokémon • Save
TCG GB and GB2	Character encoding

This data structure article is part of **Project Games**, a Bulbapedia project that aims to write comprehensive articles on the Pokémon games.