Pokemon Essentials - Extraction

David Rodriguez Soares

August 12, 2020

Privacy policy

This is a confidential document and should not be distributed under any circumstance. Please click and read.

Contents

1 What this document is about

This document holds information about how to extract Pokemon Essentials game data, which is relevant in project PoGER's goal of proposing an implementation free from RPG Maker XP.

Please read this document's Privacy Policy.

As a result of the limited scope of PoGER and the limited time and information available to the author, the following documentation isn't complete and may not be accurate.

The information was obtained through the official RPG Maker XP built-in documentation, user content found on the internet (forum posts, videos) and the author's reverse-engineering work.

The following abbreviations may be present :

- **RMXP** - RPG Maker XP
- **PE** - Pokemon Essentials
- **RGSS** - Ruby Game Scripting System

Please note that the author is not a native English speaker.

1.1 Reminder - PoGER and motivations

PoGER is an academic research project, that started as an interest to understand of how Pokemon Essentials (and its derived games) work.

What I rapidly determined was that tools like RMXP and PE weren't adapted to creating a game in a *modern* way, and suffered from their age and design decision, to the point that they were basically abandoned and deprecated. It was clear to me that I should not try to build upon them, but rather bring them up somehow.

I started by identifying shortcomings of the studied software, then moved on to coming up with solutions.

I'm looking into implementing a similar engine that would be decoupled from RPG Maker XP and would run games as programs written in an interpreted domain-specific language I would create.

This document covers the complementary pieces of software that automate the port of Pokemon Essentials. Games based on PE should be able to be ported through the same process, with perhaps a few tweaks.

1.2 Reminder - technical findings

1.2.1 Structure

PE and its derivatives share a common project structure :

○ Root	Contains compiled game files and libraries, plus a few useful programs, some probably made by Pokemon Essentials developers.
↳ Audio	Contains the musical assets of the game, typically MIDI and OGG files.
↳ Data	Contains the logic of the game and most of its data.
↳ Fonts	Contain fonts used for displaying text.
↳ Graphics	Contains the graphical assets of the game, typically tile sets, in a well organized folder structure.
↳ (*PBS*)	(Facultative) Contains human-readable data to be compiled, like items, species, types, dialogue (and translations), etc.

1.2.2 Data representation

The data that constitutes a PE project is diverse :

Category	Description	Storage
Scripts	Logic of the game, classes, ect	`Scripts.rxdata` file
Objects	Files containing sets of class instances, grouped per file	`dat` and `rxdata` files
Maps	Contain map representation (incl. events), one per file	`MapXXX.rxdata`
Dialogue	Contain dialogue	TODO

2 Research

In order to decouple PE from its original code base and RGSS, it became obvious that all useful data had to be extracted from the project.

Since `Audio`, `Graphics` and `Fonts` directories contain already-accessible data, they didn't require any extraction efforts.

See section *Data representation* for categories of data that were identified to need extraction.

2.1 Scripts

What I found :

- `Scripts.rxdata` is the second largest compiled file of the project, and contains over 120k lines of code.
- The script is unique : it is divided in named sections (that appear in RMXP's editor on the left hand side) that allow to manage its colossal line count.
- Like other compiled files, its content are almost impossible to read outside of RMXP.

I spent quite a lot of time working on a python script able to decrypt `rxdata` files, and `Scripts.rxdata` in particular, but didn't have much success, so I tried looking elsewhere.

Trying something else : After some more digging, I found a functional utility that was created for the purpose of editing/extracting the scripts from this file : Gemini Editor.

Here are links to a forum post about it and a github repository.

With it, I was able to extract Pokemon Essentials scripts (see '`PE_scripts`' directory).

It proved invaluable to be able to explore/search the whole script base using a more powerful code editor, saving lots of time from my attempts to create my own scripts to export data.

Note that extracting scripts isn't really useful : the objective is to get away from the RMXP-tied original codebase. It's only useful for the purpose of understanding how RMXP/RGSS/PE work and how features are implemented.

2.2 Maps and Events

Maps are essential to games made with RMXP, and a huge portion of the interface is dedicated to them. They contain textures that build the world and events that add elements of behavior to them.

I documented all I could find on maps the events they contain and more in `RMPX_doc`.

What I found :

- Trying to extract everything from compiled files would require a lot of effort.
- Creating scripts in RMXP that extract data is way easier.
- A solid understanding of the Ruby language and classes involved is mandatory.
- Class declarations can be found in the code and/or in the official documentation with a bit of digging.

Here is a diagram for classes involved in Map/Event/Tileset data extraction.

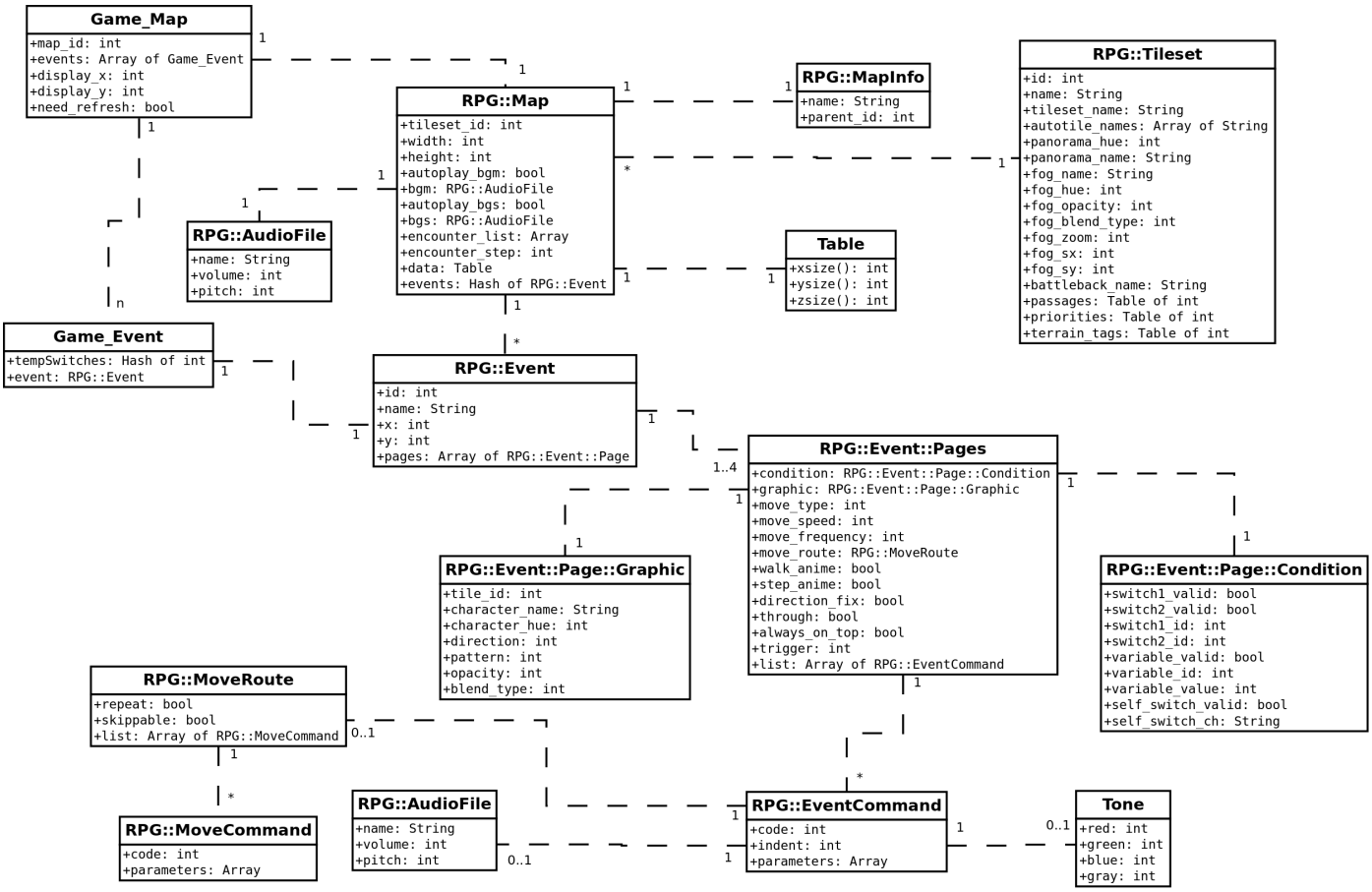

Figure 1: Simplified class map representation for Map/Event

Semantic/Syntax : Linked classes (with arity) display an **associative relationship**.

Note : There is no inheritance relationship between any two classes represented. Arities are logically deduced and may not be exact depending in proprietary implementation details. Class `RPG::AudioFile` was duplicated for ease of association routing.

2.3 Dialogue

English dialogue is hard-coded in events, therefore in order to propose alternative languages some trickery is needed.

More research is needed here.

3 Automated extraction process

Unfortunately, the first steps require the use of RPG Maker XP (version 1.02 tested, other may work also), and therefore Microsoft Windows (XP-10).

It may be possible to run RPG Maker XP on other operating systems using compatibility layers (such as Wine for GNU/Linux), but I didn't try it.

3.1 Opening a project with RPG Maker XP

First things first, a copy of the project to extract is needed. It should have the same structure as described in `section 1.2.1`. If so, it should be possible to open it and run it using RPG Maker XP.

Read more about obtaining a copy in `PoGER`.

The following information was obtained through a trial-and-error process when trying to open the Pokemon Essentials and a derivative game (Pokemon Uranium) in RPG Maker XP and run them in order to extract data from them. As such, they may contain erroneous statements or not apply/work for someone else trying to replicate the same process.

- You need the following files in the root directory :
  - `Game.exe` : The executable that launches the game.
  - `Game.ini` : A configuration file. If the `*.ini` in the project has an other name, *rename a copy* and **leave the original untouched**, because it seems RMXP needs a `Game.ini` file and game can specify an other name (hard coded in scripts)
  - `Game.rxproj` : Entry point for RMXP.
  - `RGSS102E.dll` : Contain RGSS dynamic libraries, probably necessary tu run the executable correctly.

  Renaming files may be needed for some project files.
- For projects with *compressed data* (no `Data` directory but there is one `*.rgssad` file at root), you need to use an extraction utility.

  I used RGSS Tool :

  ```
  python rgsstool.py -x -d "path\for\output\files" "path\of\rgssad\file"
  ```

  This worked with Python 3.x, google search it if you're not sure if you have a recent version of Python and how to launch it on your computer.

  Exemple for Pokemon Uranium when command line is at root :

  ```
  python rgsstool.py -x -d . Uranium.rgssad
  ```

- For projects missing the `*.rxproj` file, simply create a new empty file and write the following :

  ```
  RPGXP 1.02
  ```

  Save it with UTF-8 encoding and no "new line" character.
- Save files should be located in `C:\Users\<user_name>\Saved Games\<name_of_the_game>\`
- Tip : When running the project from RPG Maker XP, some files in the `Data` directory may be recompiled, which can cause errors. Keep a copy of the content of this directory in case you need to restore its content (you can exclude the `Scripts.rxdata` if you're editing the scripts, so you don't lose progress on this file).

3.2 RMXP script

The first step uses RPG Maker XP itself and a handmade script.

Steps :

1. Open the project in RPG Maker XP. Make sure the game can be launched (press `F12` or click on the arrow next to the note icon).
2. Open the script editor.
3. Open `Extraction.rb` and copy-paste its content to a new page in the script editor.
4. Next step is tricky : You need to call the script from somewhere.

   Recommended way : Run the game within RPG Maker XP and play until you can gain control of the character and save. Close the game. Then, create an event close to your character and add a script command with parameter `pbSaveAll()`.

   "Fast" (theoretical, untested) way : Find the initial event that runs when the game launches and add to it a script command with parameter `pbSaveAll()`.
5. Now, Launch the game again and trigger the event.

- Tip : Add "show text" commands before and after the script call to better visualize its execution (ex: "*Click to save data.*" and "*Done !*").

It works for Pokemon Essentials but may not work for its derivative works, especially if they changed data representations (unlikely) or if they contain unexpected data (likely).

After running the script, a few new folders should have appeared :

- `Maps` : Contain 2 files for each map (one for the map itself, the other for its events)
- `Tileset_data` : Contain one file per tileset.

There should also be a `Extraction.log` file, used to help debug issues with the script.

3.3 Event processing

Unfortunately, event porting is particularly complex for multiple reasons, including but not limited to :

- The extracted objects are representations of RPG Maker XP's event interface, therefore inherits its structure additionally to useful information.
- The bulk of the behavior is expressed as event commands, which are numerous and need individual attention.
- The goal is to extract semantics into a new domain-specific language.

For this reason, a Python program was written for the purpose of processing the JSON-formatted event files from the previous step into usable files.

4 Manual extraction

Unfortunately, I found some elements of RPG Maker XP to not be accessible using scripting or external utilities, thus making it impossible to automatically extract.

Fortunately, they are scarce and it's easy to manually extract them from the UI.

4.1 Switch and Variable names

There are around 34 switches and a dozen variables used by PE. Commands that use them use their *id* to reference them, not their name.

While it is not strictly necessary to extract their corresponding names, it makes resulting commands way easier to read.

The most straightforward id-to-name conversion strategy is to create a Python dictionary with ids as key and name as values. This allows the conversion with a simple dictionary call.

You can find such an implementation in the `PE_variables_switches.py` file.

Procedure to extract names :

1. Create/Select an event, to display the "Event" window.
2. Create a new event command. In the "Event command" window, select "Conditional Branch". *There are other commands that could do the trick, I just chose this one.*
3. On the "Conditional Branch" window, select either "Switch" or "Variable" and click on the switch/variable selection box immediately to its right.
4. You should see a new window lists of items. Just select them one by one and copy-paste their name in the dictionary you're building (at the corresponding index of course).

5 Remarks

5.1 Contact

Contact the author by email : David.Rodriguez.1@etu.unige.ch

5.2 Privacy Policy