**Fire Emblem XP (FEXP)**

**The Ultimate Guide**

# (V2, 5.0: 9/28/12)

Hi guys, I'm Anthony Carter, known as Klokinator online, and this is my guide for using Fire Emblem XP. FEXP is a cool new engine for the FE community that's only been out for about a year (As of this document's writing) and has nearly unlimited potential. Coding started on FEXP somewhere around 2007 to 2008 and the engine was coded by Bwdyeti.

Bwdyeti wanted to create a free fan-game engine using RPG Maker XP, because the hacking community was extremely well developed and was not going to progress much further as it had reached the end of its expandability, at least as far as basic tools were concerned. While hacking has a lot of great things about itself, the limitation on colors and creativity within a Rom are rather sizable, and he wanted to make his own game that included many, many things that are not implementable in a romhack. To that end, FEXP is his greatest accomplishment.

FEXP uses Rpg Maker XP as the base it runs on. It uses a somewhat modified version of RPG Maker's Scripting System, of which I don't really understand the details and Yeti would have to explain it himself. It uses external text documents for character conversations ingame, but also has the capability to use RM's default text editor if needed. FEXP has many different ways it can handle something which is great because the flexibility and power it offers is its greatest strength.

I hope this document advances your knowledge enough to make your own game someday soon, so learn as much as you can from me, then surpass me!

-Klokinator

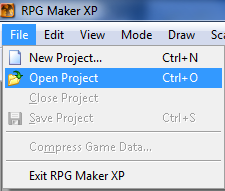
Table of Contents

(CTRL+Click to follow links to their respective chapters)

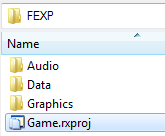
* [**Chapter 1: Introduction to FEXP's Functions**](#Chapter1)
* [**Chapter 2: Creating Characters**](#Chapter2)
  + [Chapter 2A: Formatting a mug for FEXP](#Chapter2A)
  + [Chapter 2B: Actor Units](#Chapter2B)
  + [Chapter 2C: Pregenerated and User-Defined Units](#Chapter2C)
* [**Chapter 3: Dialogue and Text Editing**](#Chapter3)
* [**Chapter 4: Weapons and Items**](#Chapter4)
* [**Chapter 5: Editing Classes in FEXP**](#Chapter5)
* **Chapter 6: Creating a Chapter**
* **Chapter 7: Eventing, the Monstrosity**

Chapter 1: Introduction to FEXP's Functions

Firstly, before you can use FEXP, you have to know how to open it and use it. Open up RPG Maker XP and go to Open Project.

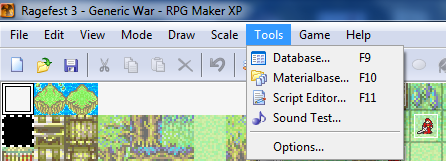


Next, navigate to the FEXP folder, and select the “game.rxdata” file inside [that folder](file:///C:\Users\Tony\Desktop\h).

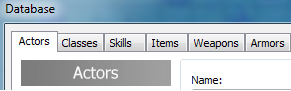


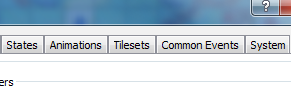
Now that your project is open, you'll see a whole bunch of things and you'll have no idea what you can do unless you know something about RMXP. A basic under- standing of RMXP is best, but a complete noob can learn how to use this engine quickly.

First, take a look at the hotbar, up top.



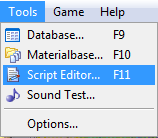
Inside the Tools drop-down menu is where over half of your actions inside FEXP will take place. Database is the first place I'll introduce you to. Here, you edit characters, classes, skills, weapons, animations, tilesets, and much more!



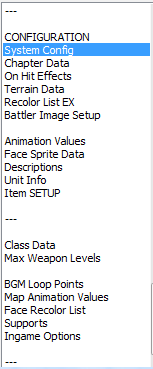


All these tabs at the top are what you use to navigate the Database tab. Click on some of them and take a look, they're quite interesting! We will go more into detail on Databasing in a later chapter, I just wanted you to see what it looks like for now.

Next, close the Database tab and take a look inside the Script Editor. This too is accessible from the Tools tab.



Even if you're familiar with RMXP, this is a very daunting prospect. To make RMXP a Fire Emblem emulation engine, Yeti had to write many many thousands of lines of code. I'm talking about tens of thousands of lines, really. Some of the code is borrowed from other programmers, but realistically 90% of the meat of it was written by Yeti. Let's see what we've got.



This isn’t even 10% of the code he added, and it’s already a lot to digest. There's around 100 or so custom scripts added by Yeti and you should familiarize yourself with those in the [top segment](file:///C:\Users\Tony\Desktop\h) at the least. That section is where 80% of my time in the scripts is spent, and 15% is spent in the Skills section, near the bottom, with the other 5% being spent editing various other sections as needed. I'll go more in-detail on Scripting in a later chapter of this guide, but for now you need to be aware of the edits it took to get FEXP working. If this is way over your head, you either need to learn more about RMXP or just give up now because you’ll be editing scripts a lot in FEXP.

For now, just understand that scripting will be a large mental obstacle to tackle, but you can easily figure it out just by reading the plain english syntax that Yeti uses. No weird complicated x=y^v132, just things like “pdef = physical defense” and things of that sort.

Okay, now let's take a look at something else, the eventing stuff. If you're familiar with RM, you already know something about eventing at the least, it's how you make your game. You place Events on the map, which correlate to units, tiles, actions, turn-based occurrences, spawns, and etc.

Let’s examine a few examples of eventing in FEXP. Right click on a tile on the map. Make sure your pointer is set to event mode, not tile mode.



Right click on one of the default units on the map. Assuming you're using Yeti's FEXP 1.0.1 release, just look for boxes that have a class icon, like so.

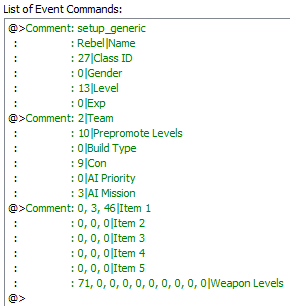


[*(NOTE)*](file:///C:\Users\Tony\Desktop\h)

[](file:///C:\Users\Tony\Desktop\h) [](file:///C:\Users\Tony\Desktop\j)

This will make the engine more professional and also add a bit of a quality tweak to the whole overall thing.

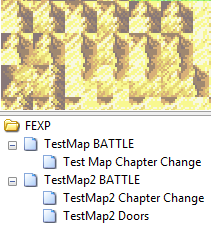
When you double click a unit event, or use the Open command from the right-click context menu, you'll get a window similar to this:



Where you see all the green text, in this example, is where you edit the unit.

Inside the event window you can find condition switches, triggers, ways of changing music, adding comments, script snippets, and many other things. I recommend reading a basic eventing tutorial rather than relying on this guide, I won't cover it in much detail. There are many good tutorials over in the RMXP section of [RPG Revolution](http://www.rpgrevolution.com), so I’d suggest going there and checking them out.

Now let's look at the final important sections of RMXP you'll be using a lot over your journey with FEXP, **mapping and map files**.



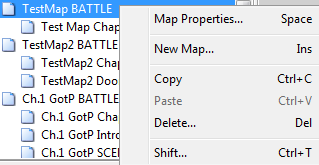
If you're used to mapping with Tiled or Mappy, using the FEXP mapping system will either be a lot easier due to functions like copy-paste and flood fill, or it will be more annoying because the way the tilesets are aligned is pretty different. I've personally gotten quite used to it and vastly prefer the [bigger tiles](file:///C:\Users\Tony\Desktop\j) and the [extra options](file:///C:\Users\Tony\Desktop\j) for laying tiles down. In order to place tiles down though, you need to be in mapping mode. Let’s swap to mapping mode now.

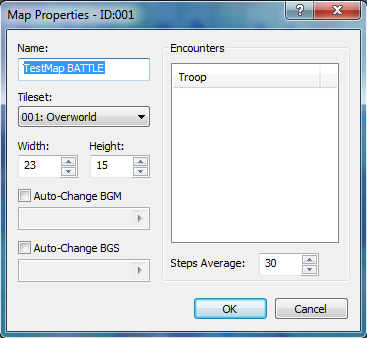
[](file:///C:\Users\Tony\Desktop\j)

Click on Layer 1 as seen above. To place tiles on the map, click on a tile from the left side of the screen, and simply click on the map to set the tile down. While making a basic map may be easy, making a good map is not as easy.

I will not be covering things like how to make good maps or pretty maps or even the theory of mapping. I leave that to my good friend Primefusion, who made an excellent tutorial on Serenes Forest ([Part 1](http://serenesforest.net/forums/index.php?act=findpost&pid=1595900) and [part 2](http://serenesforest.net/forums/index.php?showtopic=25963&view=findpost&p=1600898)) as well as my site, [Klokreations](http://klokreations.net/). Mapping is extremely simple when compared to GBA Hacking. There is no need to create maps in separate programs, memorize offsets, save to a specific file type, import to the Rom, change coordinates or any of that annoying stuff. You simply make the map in the editor, set the map name to BATTLE or SCENE and it works. Every time.

In addition, to change map properties, right click on a map file-name and go to Map Properties.





In this window is some information you may find useful and you can also edit if need be.

The 001 at the top is the map number. This is important to know for editing the title screen to go to a specific map. The map name can also be changed, and there are certain words that change how the map acts. Those words are BATTLE, SCENE, and PREP. I will go into more detail on these later on in the chapter creation section.

Finally, you can choose the tile-set you wish to use, and you can change the size of the map as well. Take note that 20x15 is the minimum map size in RMXP, and Yeti has made it possible to go a little smaller than that in FEXP, though not by much. As well, maps that are too large (I'd say 30x30 is the biggest I'd ever use) and have too many events (I'd say more than 150 events is overkill) will run very slow, so don't overdo it with map sizes.

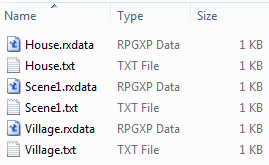
Now that we've covered everything in basic detail that can be done inside the engine, let's look at external sources outside the engine. Take note, any changes you make to external files will not be loaded in the engine unless you press F12 and playtest in the engine. This is because of some things Yeti altered that make it so game.exe is the “official” way of playing the game, while using the in-editor playtest option is the way to make changes and play as a debugger. Before releasing a FEXP game, always use playtest first, then use game.exe and make sure it works, otherwise someone could run into an error with game.exe that crashes the game, but playtest will work fine in-editor. It's complicated, but you'll learn it eventually.

Okay! First, go to your game's root folder. As an example, the route to get to my FEXP 1.0.1 folder is:

C:\Users\Klok\Documents\RPGXP\FEXP 1.0.1

Yours might be different, but hopefully you can find it. Go into the Data folder, then go to the Text folder, which should be there if you're using FEXP 1.0.1 or any of [my releases](file:///C:\Users\Tony\Desktop\j). Once in the Text folder, you'll see a folder called Test-map or something else, it doesn't really matter. You can make as many folders as you want and name them whatever you want and organize them to your needs, but I strongly recommend small names, like “Ch.1” as an example. Explanations of why will be left in the conversations and talk events chapter later.

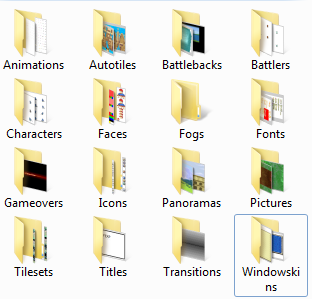
Inside this folder, you'll see a whole bunch of text documents that may resemble this:



First, you create a [text document](file:///C:\Users\Tony\Desktop\j) and name it what you want. After you have edited text, save the document, open up the editor, do the appropriate linking as described in a later chapter, then remember to playtest. Playtesting will add a file with an identical name as the one you made, but it will be an rxdata file. This tells the engine a bunch of things that aren't really necessary for you to know but it makes sure the text document works right.

After that, every time you edit that text document, remember to playtest in the editor so your changes go through. If you don't playtest, I guarantee you text will never save.

Now let's take a quick look at other things. Namely, the [Graphics folder](file:///C:\Users\Tony\Desktop\j). Go back to your root game directory and go to the Graphics folder. Here you will see a whole bunch of folders. This is normal. Some of these folders don't have anything in them, but don't delete them.



Let me tell you what each folder does.

**Animations** is where you put RMXP animation sheets so that they show inside the editor, in the Animation tab of the Database. I'll go more into this in a later chapter, but for now, just know that they are made up of rows of 5 squares across and infinite down, that must be 192x192 pixels, and the background should usually be transparent. If you don't know how to make backgrounds transparent, do a Google search.

**Autotiles** is used for tilesets. It allows things like moving water and basically tiles that are animated. I do not fully understand how they work and will have to question Yeti about this in the future.

**Battlebacks** are used for battle animations. When the characters fight, they are standing on platforms, and battlebacks is where those platforms are located.

**Battlers** is used for Dodging and standing frames for classes. Nothing else.

**Characters** is used for map sprites and basically anything involving units or tiles on the map not directly part of a tile-set.

**Faces** is where you put specially formatted FEXP mugs for your characters ingame, like where you would put Eliwood's face. I will go into details on this later in the guide.

**Fogs** is probably for fog of war. This was either never finished or I just don't understand it sufficiently.

**Fonts** is for fonts ingame.

**Gameovers** is for your gameover screen(s).

**Icons** is where all weapon and skill icons go. If you program icons to be used elsewhere, you might also use this folder.

**Panoramas** are used in conversations as backgrounds and are brought up with the \g[Filename] command. I'll add a bunch of details on these in another chapter.

**Pictures** is the main folder for pictures, obviously. This includes pictures used on the title screen (but not the main BG of the title screen) or other assorted pictures. Glance through the folder and you'll get the idea.

**Tilesets** is for tilesets used ingame and in the editor. I don't understand them fully yet, but I know that there has to be a 32x32 version (used in the editor) and a smaller 16x16 version (used in the game) in order for them to work properly. I can't provide much help on these for right now though, sorry.

**Titles** is for your main title screen. I have a code snippet that Yeti made for me that allows multiple zoom levels for the title screen, I'll share that at a later time.

**Transitions** I think is unused.

**Windowskins** is self explanatory.

Finally, there's the Audio folder which is extremely self explanatory.

**BGM** is for Background Music, used during any scene or battle. It’s background music… c’mon.

**BGS** is Background Sounds, used for things like clapping in the arena.

**SE** is Sound Effects, used primarily for battle animation sound effects like swords clashing and horses dashing.

**ME** is Musical Effects, used for placing a sound that stops the BGM temporarily until the sound has played.

This is it for chapter 1. While some of this may have gone over your head, my goal is for you to fully understand everything presented in this first chapter by the time you've finished reading this guide. It'll take time, and a lot of hard work, but you too can become a master FEXP game creator within a short period of time!



Chapter 2: Creating Characters

Unlike romhacking, there are no tutorials or documentation for FEXP aside from maybe a few scattered tutorials I made and notes in the scripts along with common sense. There is no real spot to start from, it's up to each individual person where they want to start. For me, when I first started, I wanted to know how to make a map. After that, I progressed to editing a character that was already on the map. Then, I wanted to get a map to function correctly, with music and other frilly things of that sort. Later I worried about learning how to make enemies, change their classes, and other things of that sort.

But you might be different. Maybe you want to start by learning how to script battle animations. Perhaps you're a great scripter who doesn't know where to start in the engine or what to modify or what's safe to modify, really the possibilities are endless. Unlike a Rom, there's no pre-created game to edit and tinker with which can be scary and daunting to some people. With the exception of the two test maps, there's little to edit and not many examples to learn from. Therefore, I will start you off with what I started with; making a character.

There are three different ways to create a character or unit in FEXP. There is the **Actor Unit** format, the **Pregenerated Unit** format, and also the **User-Defined Unit** format. Each has its own utility and specific occasion to be used, it’s recommended that you get used to all of them.

First though, let’s learn how to correctly format a mug to be used in FEXP. Otherwise, your character won’t have a face ingame.



This is Eliwood, a character in Blazing Sword, formatted to GBA Romhacking format. The mug goes on the top left, mouth frames have two sets that go underneath the mug, the chibi at top right, and the eye frames have one set which go on the middle right.



Eliwood again, this time formatted to FEXP’s template. Now, however, he has two sets of eye frames, a [Battle Face set](file:///C:\Users\Tony\Desktop\j), chibi at the bottom left, and the eyes and mouth frames are underneath the mug.

Let’s examine how to format a mug to the new formats.

Chapter 2A: Formatting a Mug for FEXP

First, we need a template that is ready to be used for FEXP. We will be using this template for this section of the guide:



This template comes with three emotion slots. The first slot is the “default” emotion, the one used in the Inventory screen ingame. It’s also the default expression used when loading a character for dialogue. The second slot can be loaded with one of the text commands detailed in a later chapter, and the third slot is also loadable with a text command.

If you are not already familiar with formatting a mug to GBA FE hacking standards, Seph1212 made a [handy tutorial](http://www.youtube.com/watch?v=_m6FQn7cycj) on youtube which will explain the basics of how to actually make frames for your mugs. Asking around on most Fire Emblem fan forums will also help if even his video tutorial doesn’t.

There are some significant differences between the way GBA FE handles mouth frames VS how FEXP would handle the same frames. As an example:



GBA FE goes from open to closed.



FEXP goes from closed to open.

This is a small, but important difference from GBA FE. If you just copy paste frames from a Romhacking formatted mug to a FEXP format sheet, the frames will be backwards. Always double check this.

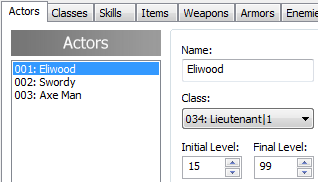
Also, mouth frames are positioned right below the eyes in FEXP, while in GBA FE there is only one eye emotion, and it's off to the side. This allows you different eye blinking emotions if you so choose to make them.

Chapter 2B: Creating an Actor Unit

I know I'm a little biased, but I have to say FEXP's format is clearly superior to GBA FE when it comes to flexibility and general utility. Also, there are no color limits so if your mug has 17 or more colors, it will still work fine in FEXP :)

All right, so your Eliwood is formatted and ready to go. Now, you save his face as a png inside the Faces folder, and remember what name you gave his file. (For example, Eliwood.png)

Now we want to make Eliwood our main character! This is the fun part, so pay attention. Open up the editor, and go to the Database. Open up the actors tab, and name the first actor slot Eliwood. If you're using FEXP 1.0.1, this should mean the 1st slot is named Armor Guy. Replace his name, he's lame anyway.



By naming the slot Eliwood, it searches the Faces folder for a file named Eliwood.png or any other file extension that's a readable picture format. If it finds it, Eliwood's face is assigned to that actor whenever he does something ingame, like on his inventory screen. You may also name multiple slots Eliwood if that's your thing and you love epic failure units, I won't judge you... for the most part. If Eliwood.png/whatever is not found, the face will come up blank ingame and you'll know you messed up.

You can also change his class and his level, but for now I'll just tell you how to change stats. The other stuff is… pretty self explanatory.



Here's what each box stands for:

Max HP = Base Actor HP

Max SP = Growth Rates for Character. More on that in a second.

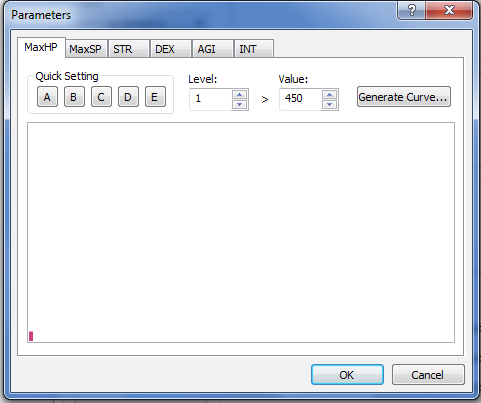
STR = Base starting POW, such as MAG or STR. There is currently no mag/str split in FEXP.

DEX = Base actor SKL.

AGI = Base actor SPD.

INT = Base actor LUK.

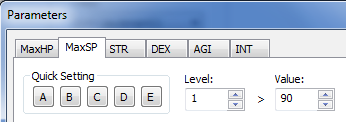
If you double click any of the above boxes, you’ll get the following window: (See next page)



The A, B, C, D, and E buttons are non-functional, as well as the “Generate Curve” button.

For the Level option, when it comes the HP, STR, DEX, AGI, and INT, the numbers 2-100 are unused. Level 1 is basically the base for those stats, set at x10 of the actual stat value you want to aim for, so 450 as displayed above = 45 starting HP for Eliwood. Once you've set your HP, you can close that window and choose the STR/other stat boxes. They function identically to the MaxHP box as far as the multiples of 10 thing goes.

Now for a quick explanation on MaxSP. This allows you to set your growth rates, no need for multiples of 10. Here's how it works.



Remember how I told you not to mess with the Level setting in these boxes? MaxSP is the only exception. Levels 1-10 have their own function as far as growth rates go. For example:

Level 1 - HP Growth. In this example, the level is 1 and the value is 90, so this means an HP growth of 90%!

Level 2 - STR Growth.

Level 3 – SKL Growth

Level 4 – SPD Growth

Level 5 – LUK Growth

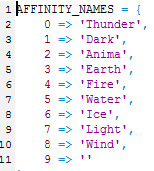
Level 6 – DEF Growth

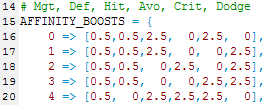
Level 7 – RES Growth

Level 8 – Affinity, more on that on the next page.

Level 9 – Gender/[Alternate battle sheets](file:///C:\Users\Tony\Desktop\j).

Affinities can be found and changed in the Scripts. In the configuration scripts, scroll down to Supports and there you will see a list of the affinities, their values, and what bonuses they give. You may also change their names and stuff here.

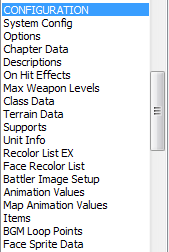




For example, 3 is the Earth affinity, so putting 3 into the Affinity value box would set that character to the Earth affinity.[(Note)](file:///C:\Users\Tony\Desktop\j)

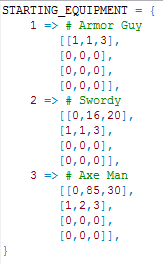
With that, you've learned how to make a character in the Database! Wow, don't you feel like a real professional? But wait! The character isn't showing up ingame that easily. No, we still have to edit them in the scripts, put their event on the map, and make their face appear in dialogue!

All right now let's open the Script Editor and go down to the configuration section. That is, this stuff:

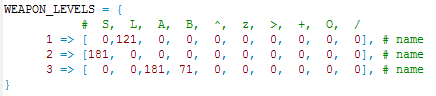


Of course, there's lots to cover here, but we'll stick just to what’s needed to make an actor unit. Now, let’s say you want the character to have more than one item when you get them, perhaps. You might also want them to have, say, weapon levels? I dunno, **I** could use Durandal with an E in swords but **I'm** just awesome like that.

Go to the Unit Info script first. Here you'll see things like this script snippet:



Also, you'll find things like generic unit stats and growths, generic unit CON, [death quotes](file:///C:\Users\Tony\Desktop\j), and the [class reel](file:///C:\Users\Tony\Desktop\j), which is a bit complicated so I'll put off that for a few chapters.

[](file:///C:\Users\Tony\Desktop\j)

Take a look at the above image, which displays weapon levels. The numeric values you may input go from 0-255.

0 = No weapon level

1 = E rank

31 = D rank

71 = C rank

121 = B rank

181 = A rank

255 = S rank

My friend Pwnagekirby successfully added in SS rank as well, so for you FE4 fanatics that might be extra useful to know about.

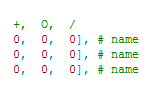
The weapons go from left to right. Swords, Lances, Axes, Bows, Fire, Thunder, Wind, Light, Dark, Staves. The numbers on the far left (1-3 in the diagram) are Actor ID's. Remember that window in the Database where you input Eliwood's name?

Take another look at that window.

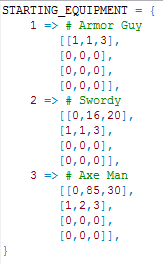


As you can see, there's a 001 on the left, indicating Eliwood's “Actor ID”. Actor ID is extremely important to remember, it's used constantly in the editor to bind characters to events, weapons, positions, and many other things of that sort. You'll be needing it to place Eliwood on the map in a couple minutes, as an example.

Back to the Weapon Levels script now.



As you can see in the above diagram, there's a ‘#’ sign and the words “name” three times off to the right. A '#' symbol makes anything after it a Comment and allows you to add... comments! These are ignored by the script and are great for making little notations. You'll want to replace actor 1's “# name” with “# Eliwood” so that you know who he is later on.

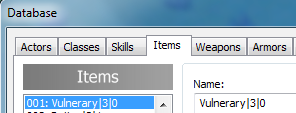


*Remember this snippet?*

This gives your #2-5 weapon/item slots ingame... things. You can give a character items and weapons here, though you probably won't understand what values to put here. Again, as you can see, there is a 1, 2, and 3 on the left of each chunk. These link to Actor ID's, don't forget which one links to which one, (Heck, I even forget occasionally.) Add a comment and the name of the character, always.

Armor Guy (Eliwood) has an item in the 1st series of digits (second item slot) and we will figure out what those digits mean. The first digit (1) tells the engine if it is a weapon or an item. **The only two values allowed for that digit are 0 and 1**. 0 is Weapon, and 1 is item. The next digit (1) is the ID of the item/weapon. Finally, the third digit tells the engine how many uses that item/weapon has. -1 indicates infinite uses, while any other digit is usable uses.[(Note)](file:///C:\Users\Tony\Desktop\j)

So let's use the first item as an example. Open up your Database. The first digit is a 1, so this means it is an item, not a weapon.



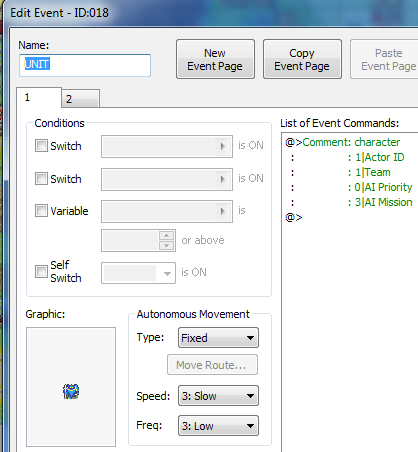
We open up the Items tab of the database, then go to the ID 001 item (A Vulnerary in this case), and the number of uses the item has will be three. We will go more in detail about items and weapons in a later chapter, so for now we'll just remember the three main digits as our starting point.

To change the Vulnerary to an Elixir on Eliwood, set the digits to [1,2,3]. That sets it to item, ID 2, uses 3.

Now that we've learned how to format a character, edit their stats and weapons, and their weapon levels, we need to place this actor on the game field! Now, you could try creating a new event for that character from scratch, but Yeti already made many example events that would be more efficiently copy-pasted, then edited. Let's open up the test-map in FEXP 1.0.1 and use it as an example.



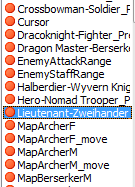
Double click the Lieutenant as seen above.



This is Eliwood’s actor. You could edit another Actor unit to ID 1 and Eliwood would be there as well, but we really only need one of any actor on the map. It causes glitches if there’s more than one.

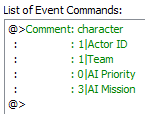
To be honest, if you did everything right up to this point, you'll be able to playtest and see Eliwood's smiling face where the Lieutenant is [ingame](file:///C:\Users\Tony\Desktop\j). That's great, but *how did he get there?* We’ll find out in just a few paragraphs but let’s not get distracted just yet.

As you can see in the above pictures, when you double click the Lieutenant's event you get an event window with only a few easily editable things in it. Over on the left you can see a box with the map sprite preview. This does not affect how the Lieutenant looks ingame, it just lets you change what you see in the editor. To change the preview sprite, double click the image and scroll through the options presented.



Lots of stuff to select, you get the idea. It's really self explanatory. Also, as I said before, my Super FEXP has revamped this entire section so that the classes are split into separate images and not merged together, two classes at a time. I also edited the alignment of every preview image manually so that they would be centered in the map preview image. Enough about me though ;D

Now, go back to the main event window and glance at all that green text stuff in the main window.



There are five lines, each one has it's own purpose.

**character**: this line is used to determine whether the event is an Actor unit, a generic unit, or a pre-generated generic unit. The three values that can be input here are...

[character](C:\\Users\\Tony\\Desktop\\j" \o "These are case-sensitive by the way, only use lower-case here.)

[setup\_generic](C:\\Users\\Tony\\Desktop\\j" \o "These are case-sensitive by the way, only use lower-case here.)

[setup\_from\_comment](C:\\Users\\Tony\\Desktop\\j" \o "These are case-sensitive by the way, only use lower-case here.)

**1|Actor ID**: This is rather self explanatory, it requires the Actor ID, since this is an actor event. Our Eliwood is Actor 1, so a 1 is required.

**1|Team**: this value determines the actor's alliance. 1 = Player. 2 = Enemy. 3 = NPC. Others can be added by you, the creator. More on that in a later chapter.

**0|AI Priority**: This determines the order the unit appears in the status screen ingame.

**3|AI Mission**: This is a complex value. It determines the unit's AI if not player controlled.

The values you may use for **AI Mission** include:

0 = Stand Still, attack if within melee/bow range.

1 = Attack in range

2 = Seek out and attack (any)

3 = Seek out and attack (weakest)

4 = Pillage (Requires Pillage AI on at least one village)

5 = Steal (Steals from units if possible, and chests if it has a chest key)

6 = Go to square (throne capture)

7 = Staff user

8 = Seek out unit to talk

9 = Avoid

10 = Do Nothing at all.

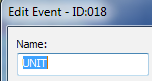
11 = Defend square

12 = Ballista/Long Range Attacker

500 = Boss AI. Stand still, attack if within range and heal self.

0-99 are for the [first AI bytes](file:///C:\Users\Tony\Desktop\j), while the hundreds place is used for movement priority. A higher number in the hundreds place will equate to that unit having a higher movement priority. (Bosses have a 5, as an example.)

Finally, let's look at the top of the event window, where you see two things that are important to know about.



The event ID is at the very top left of the window, and in this case it's 18. You won't need to use this too often, but it is important to know. I'll cover its uses later as they come up.

Next, look at the name of the event, “UNIT”.

UNIT, BOSS, and ITEM are the three values that make a unit event work and function properly. Forget to add them, and they will not display on the map and they may even cause your game to crash.

**UNIT** is always included by default.

**BOSS** is something you add after UNIT to make that unit a boss. Bosses have improved stats and if your chapter is a defeat boss chapter, the boss with the lowest event ID is automagically made the main boss. (More on chapters later.)

**ITEM** makes the last item droppable. ITEM is also added after UNIT.

Therefore, the combinations would be:

UNIT

UNIT BOSS

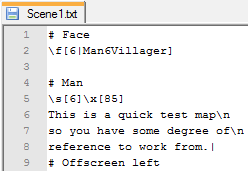
UNIT ITEM

UNIT BOSS ITEM / UNIT ITEM BOSS. (The order of BOSS and ITEM don't matter.)

Well, you've edited a bunch of things, your Eliwood is sexy and ready to get killed by all the enemies with no stats. Because he's Eliwood. And he sucks. Let's playtest! We press F12 while in-editor and our game starts up.



Success! Now my next goal is to get this creepy virgin into some dialogue! Go into the Text folder as I described in chapter 1 and glance in the documents there. Look for a document called Scene1.txt and open it up. You are greeted with this:



I will go more thoroughly into text commands in a later chapter, but for now let's make Eliwood's face appear in dialogue. Replace “Man6Villager” with “Eliwood”, save the changes, and playtest the game in the editor.

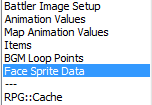
You should now see this:



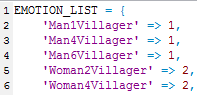
DEAR GOD WHY?!

Eliwood's face has fallen off-screen and he is clinging to the edge, hoping he won't fall into the endless abyss of [immosubuke error](file:///C:\Users\Tony\Desktop\j), it's up to you to save him! Yes, we did actually forget one very vital step.

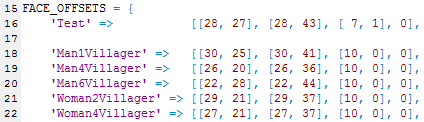
Open up the Script Editor, and look for the Face Sprite Data script, in the Configuration section as before.



Now we have two important things that must be edited before Eliwood will properly display ingame.



The emotion list script entry.



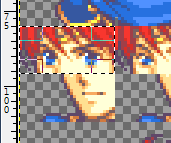
And the Face Offsets script entry.

Emotion List and Face Offsets are pretty important unless you plan to have a game with no legit characters and only generics. Emotion list is simple. Remember when you saw the Eliwood face earlier? There were two slots for facial emotions, by which I mean two spots that had eye and mouth frames. Therefore you'd create a new entry, name it Eliwood, and assign it a value of 2.

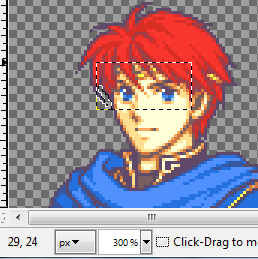


Like so.

Next, we need to add Face Offsets for Eliwood. This part is a bit complicated and it loses a lot of people, so I'll carefully explain how I obtain the offsets. I recommend downloading GIMP for this, because GIMP makes this step very simple.

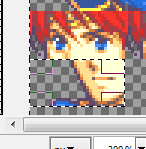


First, make a 16x32 box around the first eye frame, the frame that's identical to the one on the mug. Copy it, then paste it back on the sheet.

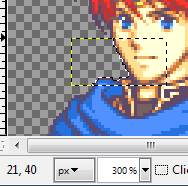


Move the whole frame until it matches the mug, then move your mouse to the top left of the frame. Don't click out, you need to see the outline of the 16x32 frame. When your mouse is aligned with the top left edge of the box, look down at the bottom left of the GIMP window (Diagram 2, bottom left) and you'll see the first coordinates you need. Those are 29, 24 (29X, 24Y).

Repeat this step but with the mouth frames. You should end up with 21, 40.



Copy the frame…



Paste it and move it back, your coordinates should match mine.

Now we need to use these numbers to make a format entry for Eliwood in FEXP.



‘Eliwood” binds this entry to any face with the name “Eliwood.png” in your faces folder. If any entry in the game uses the face Eliwood uses, it will bind these formatting changes to that face. Nifty huh?

The first of the 4 numbers is the eye frames, input the 29, 24 coordinates there. The next two digits are the mouth frames, input the 21, 40 coordinates there. Next we tackle the 10, 0 numbers, which are for Status Offsets.



You may notice Eliwood was off to the left a bit when we first played, because we hadn't added Status Offsets yet. Now that it's 10, 0, what will he look like?



He shifted right just a tad and is now aligned noticeably better.

The final digit, 0, assigns an emotion to the status screen. For example, if the second emotion slot in his formatted sheet was his “angry” face, setting this value to 1 would make his angry face the default status emotion.

Now that he's aligned properly, he should be perfect ingame. Let's see!



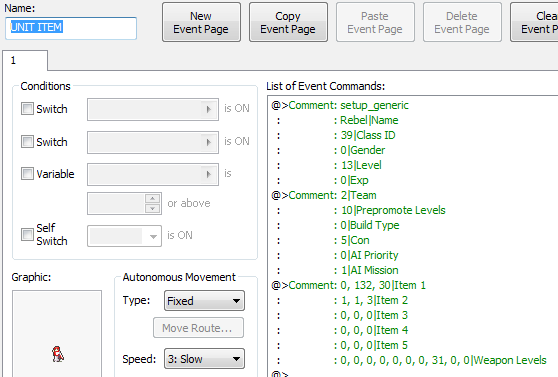
Chapter 2C: Pregenerated and User-Defined Units

Making Actor units can be complicated, and you'll eventually get used to it. For most people though, knowing how to make Generic units will be key to making good chapters. Without generics, the enemy has no cannon fodder and you have no useless allies, the staple of any Fire Emblem game!

Remember how I told you that after naming Eliwood.png, you could assign it to an Actor slot by naming that slot Eliwood? Well, you can do the same thing for generic units as well. In fact, generics aren't that different from Actor units, other than the fact that they don't carry through to the next chapter and they are created on the map instead of in the Database.

Glance around at the various enemies on your Testmap in FEXP 1.0.1 and you'll find that Yeti used only the pregenerated generic units format. Well, there is also the user defined generic unit type, and I'll teach you how to make those as well.

Loading up a random unit shows us this:



If you recall the Actor unit layout, you'll see there's a lot more to edit with a pregenerated generic unit. (setup\_generic is used for pregenerated units) Let's see what all there is to edit.

**Rebel|Name**: This sets the name for the unit. Never edit anything after the | mark, only what comes before it. If you have a face in your Faces folder that is called Rebel, that face will show up for this unit. If not, generic unit types will search for faces that match the class name. More on that... later.

**39|Class ID**: This sets the class ID. To find this, go into Database, find the class you want to set in the Class tab, and put its ID here. More info on Class creation in a later chapter.

**0|Gender**: As I said before, even numbers are male, and odd numbers are female and anything higher than 1 is for alternate battle animations.

**13|Level**: This sets the level for this unit. 20 is max by default but this is easily editable.

**0|Exp**: This sets the unit EXP. It is 0 by default and will be two dash marks ingame for non-player units, regardless of what you set it to.

**2|Team**: I explained teams earlier, if you recall. 1 = Player, 2 = Enemy, 3 = Ally. The rest you can add yourself.

**10|Prepromote Levels**: This will auto-level the unit's stats based on their average class growths. In this case, if the average HP growth was 50%, it would level that stat ten times and the stat would have a 50% chance to grow each time. This value can be set as high as you want, I used a 38 for the bosses in Generic War as an example. Also, class growths and base stats can be set in the Unit Info script.

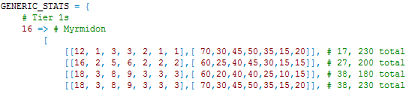
**0|Build Type**: This chooses one of the growth setups you defined in the Unit Info script.

In this case, the first line is for Build 0, the next for Build 1, the next for Build 2, and etc. You may have as many as you like. (See diagram below)

**5|Con**: This sets unit constitution, the only stat not defined for pregenerated generics in the Unit Info script.

**0|AI Priority**: This was explained in the previous sub-chapter.

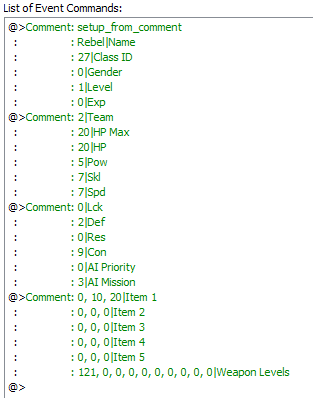
**1|AI Mission**: This was also explained in the previous sub-chapter.



(Information about editing class stats and growths will be featured in a couple more chapters, not this one. Apologies in advance.)

Items and growths are set in the final two options, they are set just like you would set them for the Starting Equipment part of the Unit Info script. There will be more elaboration on these in a later chapter.

This is really all you need to know to make a pre-generated Generic unit. However, there is one other unit type (I made sure to include it in my eventual Super FEXP release) you need to know, which is a User-Defined generic unit. It's not too different from a pre-generated unit, except you set all the stats manually. Check it out.



The only large differences are that you set the stats individually yourself, the unit has no natural growths and its stats can't be [autoleveled](file:///C:\Users\Tony\Desktop\j). This is largely useless for a major project unless you need a fine-tuned game, like Project Dondon or a Ragefest entry with a specific way of beating the chapter.

One other thing I should mention, weapon levels are easy to mix up. From left to right, this is the order of each level:

Swords

Lances

Axes

Bows

Fire

Thunder

Wind

Light

Dark

Staves.

If you should try and give your character a lot of weapon levels, keep in mind that the comment has to stay on only one line or the unit will not display ingame. Therefore, adding another comment and continuing the levels will not work. Generally more than 6 max weapon levels is all you can do for a generic unit. You can make all weapons max rank if it's an actor, but not for a generic.

In other news, you learned all this in this chapter:

* How to align a mug sheet to FEXP Format.
* How to assign the sheet to an actor slot.
* How to edit base stats and growths for actor units.
* How to edit starting weapons and items for actor units.
* How to change weapon levels for an actor.
* How to create bosses, units, and droppable items.
* How to edit the different facets of an actor Event.
* How to find the coordinates so a mug displays correctly in FEXP.
* How to create Actor, Pregenerated, and User-Defined units.

This concludes the chapter on basic unit creation! I did not elaborate on classes, items, weapons, text editing, battle animations, and many other things, but those will be featured in upcoming chapters.

With that, we move on to the next order of business... the basic text editing chapter!

Chapter 3: Dialogue and Text Editing

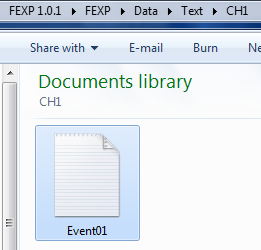
Text editing is an integral part of any game project. If you want your characters to seem realistic, have good, interesting cutscenes, and have good grammar and spelling so that they don’t look retarded on screen, well, I can’t help you with that. I can show you how to make text documents, arrange them in an organized way, and even give you a couple tips, but otherwise this part of the guide will not make you an amazing dialogue writer.

More than a few of the people reading this guide will have used Feditor to write text and will be at least somewhat familiar with how to write text for FEXP. However, romhacking and FEXP may use similar styled text scripting, but FEXP has a few things that set it apart.

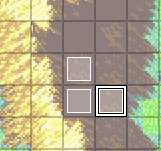
Some of my FEXP using friends like to edit their text in-editor, “because it's easier”. Well, it may be easier, but the text comes riddled with errors, and editing large chunks is a lot harder inside the editor. You also lose all the functions of a word-document, like Find and Replace, shortcut keys, and all the extra space an external file gives you. In addition, editing inside the editor is sloppy and can slow down your maps a bit due to the fact that large events run more slowly.

Therefore, I will be covering mainly external editing, but will also add an addendum for internal editing later as well.

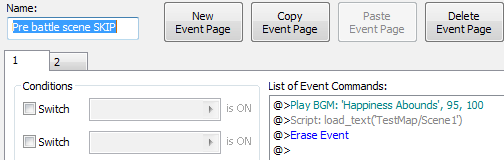
First, go to your project's folder. Assuming you're using FEXP 1.0.1, go into the Data<Text folder, and make a new folder called “CH1”. (Minus the quotes of course.) Inside that folder, make a new text file called “Event01”. It should look like this when you're done.



Great! Now before we edit anything, let's go into the editor and make a few small changes. Load up your project, and go to the first Testmap. There, look for this event and double click it:



Open that event and you'll see this: (Next page)

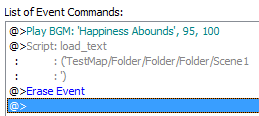


This event is an autorun event, which means that on a [BATTLE map,](file:///C:\Users\Tony\Desktop\j) it plays before the first turn starts. The event title is mostly unimportant for this event, except for the “SKIP” in the title. This tells the event that this is a CG-style event, and that pressing the “start” button would skip this event. Well, it should *in practice* anyway, but I’ve never had luck getting this to work.

Looking at the event commands box, you'll see the script command that loads text. **load\_text('TestMap/Scene1')** tells the editor to [load a text file](file:///C:\Users\Tony\Desktop\j) and play it as a dialogue scene. [(Note)](file:///C:\Users\Tony\Desktop\j)

**TestMap** tells it to search the Text folder in your System folder and go to the first folder it finds, TestMap.

“Scene1” tells the editor to open a text file named Scene1, since it's the last option in this case. If Scene1 were not there, the editor would assume TestMap was a text document in the root folder and would attempt to load it as so. You could also add another layer or two of folders, but be careful not to scroll the text like this or it'll break:

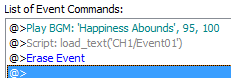


By doing this, the editor cannot process it correctly, so this is why short names for your folders and files are essential. However, Yeti gave me a workaround after the first version of my tutorial that explains a way to incorporate longer names for your folders.

[Load\_text(‘TestMap/Folder/Folder/’ +](C:\\Users\\Tony\\Desktop\\j" \o "Using a plus mark (+) allows a script line to be extended to the next line, and the next again, affording you more room to write out your complex script snippets.)

[‘Folder/Scene1’)](C:\\Users\\Tony\\Desktop\\j" \o "Using a plus mark (+) allows a script line to be extended to the next line, and the next again, affording you more room to write out your complex script snippets.)

Anyway, change the load\_text command to look like this:



Replace the **TestMap/Scene1** with this command:

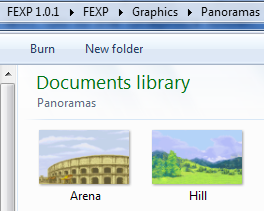
**CH1/Event01**

This redirects the old text command to your new folder and allows you to now start editing that text document. For shits and giggles, let's playtest the map now

Success! The game is now permanently hanging! Redirecting to an empty text document makes the game hang. Don't do it.

Open up your Event01.txt now and let's get to writing.

First, we need to add a [Panorama](file:///C:\Users\Tony\Desktop\j) to the text. Go to the Graphics folder, and look for the Panorama's folder as I detailed in the first chapter.



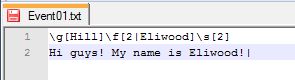
W-wow! There are only two. 😐 (Seriously yeti you only added two panoramas wtf lol)

Right well, I'll be adding more in Super FEXP, but for now let's use Hill as our background. Go back to your text document and type in...

**\g[Hill]**

Basically, “\g” [loads images](file:///C:\Users\Tony\Desktop\j) from the Panorama folder into dialogue. The images must be 320x192 and preferably .png format for the smallest possible filesize with highest quality. Ignoring the file extension, you must load them by their name, case sensitive (I think case sensitive, not sure). In this case, it's \g[Hill].

Okay, now let's add Eliwood to the text. Type the following into the document:



(I really recommend using [Notepad++](http://www.notepad-plus-plus.org) by the way, Prime showed it to me and it is sooo much better than the default .txt windows editor.) [(Further note)](file:///C:\Users\Tony\Desktop\j)

Now for an explanation of what you typed.

\f[2|Eliwood] loads Eliwood's face from the Faces folder into the second character position slot. That is, the equivalent of [LoadMidLeft] in romhacking, or rather, the middle left portion of the screen. The loadable positions include:

0 = Offscreen left.

1 = Far left of screen.

2 = Mid left of screen.

3 = Left of screen.

4 = Right of screen

5 = Mid right of screen.

6 = Far right of screen.

7 = Offscreen right.

-1 = Ghostly voice, middle of screen.

-2 = Closes textbox.

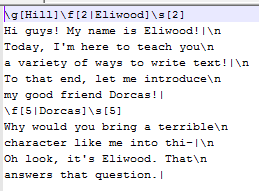
Next, you'll notice the \s[2] command I used. Using \f loads a face into the dialogue (And if a face already exists in the slot you load to, it replaces that face), but \s makes the specified slot open to speak. Therefore, \s[2] makes slot 2 start speaking when you add in dialogue afterwards.

Speaking of dialogue, Eliwood now speaks and says:

**Hi guys! My name is Eliwood!|**

The | command is used to indicate a text stop, and is identical in function to the [[A] command](file:///C:\Users\Tony\Desktop\j) in GBA FE romhacking.

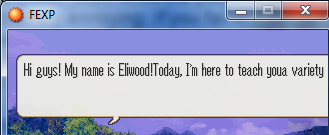
In order to advance in our knowledge of text editing, I want to introduce you to a new character, he is also from Fire Emblem 7; Blazing Sword. His name is Dorcas and he is of the Fighter class.



You may notice the constant usage of **\n** at the end of every line. \n starts a new line of text, which is one thing a romhacker will have to get used to with FEXP. In GBA FE, you would simply press Enter and continue typing text, but in FEXP you need to use \n or your text will scroll offscreen and become invisible.

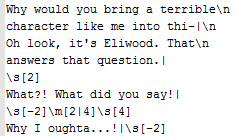
Also, you do not need to use | except at the end of a sentence, or every 3 lines. If you use it at the end of every line, you'll force the player to press the “A” button on every single line which is extremely annoying. If you fail to use it every 3 lines, you'll have what's known as a Text Skip, and [EVERYONE](file:///C:\Users\Tony\Desktop\j) hates text skips.

In addition, you want to ensure this doesn't happen to you:



By forgetting to include the \n command, or just by trying to fit too much text on a single line, that line will flow offscreen and in some cases it will break text and the game itself. I've had instances where including too much text actually made all the text invisible. Weird stuff there.

Now, let's learn some more text commands!



When you use the \s[#] command, you have the option of putting text after it, or under it and it'll display the same, like this:

\s[2]What?! What did you say!|

or

\s[2]

What?! What did you say!|

For organization's sake, [I prefer](file:///C:\Users\Tony\Desktop\j) putting the text underneath the commands. It makes them easier to tell apart and distinguish from one another, and lets you spot errors more easily. I don't really ever make errors though which is why I almost never [check my text](file:///C:\Users\Tony\Desktop\j).

Anyway, you'll notice the usage of a command called \m[2|4]. This is the Move Character command. It moves slot 2 to slot 4, in this example. The first number is the slot chosen to be moved, and the second number is the slot to move to. The unit will retain its facing direction, and if the slot to move to is occupied, the mover will swap with the occupant of the slot he takes over. Like so:

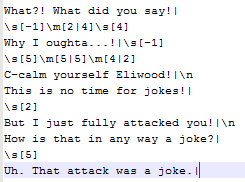




(Okay so this clearly isn’t Dorcas he’s talking to but whatever man)

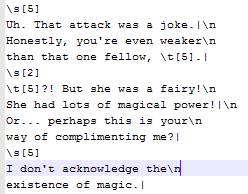
This is helpful to know in case you want your characters to charge forward angrily in text or something similar. Now, one other thing I added is \s[-2] just before the move command. \s[-2] erases the textbox from the currently speaking character, which looks better stylistically than the textbox hanging over where they used to be as they charge forwards. Yes, this does matter if you want your game to look good.

Now, let's look at more text commands you might decide to use.

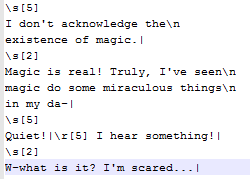


Haha my guide is so [funny](file:///C:\Users\Tony\Desktop\j)!

Here, I've used the \m command again, but this time with a change. When you use the \m command to move a character to the same spot they're already in, it makes their mug [“hop” a pixel or two](file:///C:\Users\Tony\Desktop\j), as if they're jumping in fright or surprise or something. I use it in this case to make Dorcas hop, then push Eliwood away.

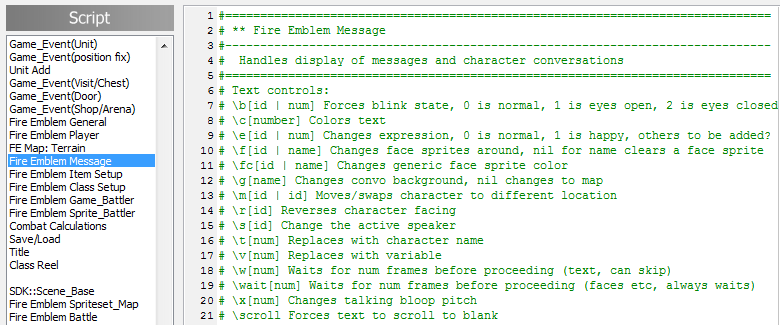


Next, we have the \t[#] command. This command is the equivalent of the [Tact] command in GBA FE, except it links to any actor slot you want. For example, if your #5 actor ID is named Hector, it would replace \t[5] with [Hector](file:///C:\Users\Tony\Desktop\j) ingame.



Next is the \r[#] command. This [reverses the facing](file:///C:\Users\Tony\Desktop\j) of the currently speaking character. This is a unique function that was not in GBA FE, thanks to Yeti for this great addition. This can also be set before a character is loaded to make them load facing backwards.

If you would like to know more of the text commands (Some appear to be non-functional), you can always look in the top of the Fire Emblem Message script, seen below.

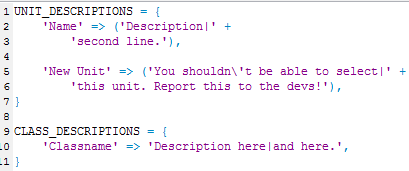


All the useable text commands will appear here, and you may add more here as well. On another note, my scripter friend Laureola from Serenes Forest added an additional function to FEXP that will allow BGM and SE to be edited inside the text document and won't make the screen flash like it originally did. This will be a part of Super FEXP when I eventually release it.

Now that we've covered the basics of writing dialogue, you should be able to write your own dialogue from scratch. However, there are some places inside the editor that use all the rules of the Text editor (As far as commands like \s and etc go) so I'll show those to you as well.

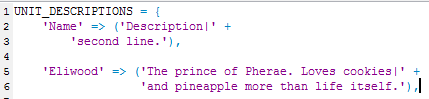
Go inside the Script Editor, and click the script titled “Descriptions”. In FEXP 1.0.1, you'll see something like this:

(Continued next page)



To be honest, Yeti didn't really leave much in here, but it's plenty for now. Unit descriptions are assigned by name, along with the description afterwards.

As an example, here's a description I wrote up just for our Eliwood fellow.



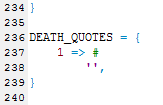
If you're a proficient GBA romhacker, you'll recall that there's a limit of two lines for character and class descriptions. There's no such limit in GBA FE, aside from practicality uses. I could even give Eliwood a detailed biography here if I wanted to, though I prefer sticking with the GBA FE style whenever possible. I do occasionally break that rule and use three lines but it's rare, and at least it's always a possibility.



**AUGH MY EYES**

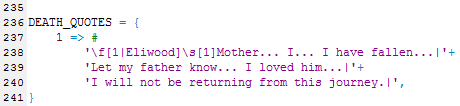
This is an example of the lack of limitations I'm talking about. I took that from Terra Vistia/Kishoa's FEXP game, Saga of the Vista. Aside from being a horrible derpy game, it saved me the trouble of writing out a bio myself and made a good example. Let us never mention that abomination again.

Another place that you'll want to add later, for aesthetic purposes, is Death Quotes.



Scroll down in the Unit Info script and you'll find this insanely tiny footnote.

Woo, that hardly even counts as an example, thanks Yeti! I'll give you a better one.

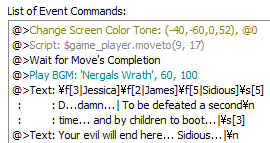


You must load faces manually for death quotes. If there is no entry here or the entry is blank, there will not be any death quote at all.

Text formatting is identical to conversational text and may be as long as you wish, though realistically, death quotes are supposed to be short unless it’s your main character

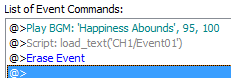
Now that we've learned how to edit text in external text documents, and we've also learned about a few scripts that have important text to edit, there's only one thing left in this chapter to cover: [Editing text internally](file:///C:\Users\Tony\Desktop\j)! The good thing is that it's more time spent in the editor and less switching to an external document. The bad news is how clunky and ugly your text is going to end up, and what a pain it is to find and correct mistakes.

Here's an example of me using internal text editing for conversations:

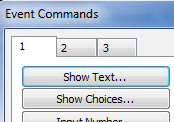
[](file:///C:\Users\Tony\Desktop\j)

This is a screencap taken from one of the oldest versions of my Phoenix Saga game, made way back in May 2011. Before Yeti added the capability to edit text documents externally, I was already being frustrated with how limited the internal editor is.

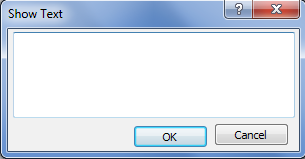
To create your own internal text document, let's open up that event on Testmap in FEXP 1.0.1 that we used in the beginning of this chapter.



We're going to delete the load text command and then we're going to double click on Erase Event, the blue words seen in the above. This will open up a box of things we can insert directly above the Erase Event command.



There are tons of options to click and I will definitely cover those in a later chapter. Until then, we'll only be clicking the Show Text button as seen above. When you click it, you'll see a text box appear which looks like this:

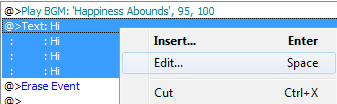


First off, anything you type gets automatically wrapped to the next line when it hits the end of the box. You can download an RMXP program called [ExtendTest](file:///C:\Users\Tony\Desktop\j) which increases the box's horizontal size, but you cannot increase the vertical size.

In addition, though there's a handy feature you can use called “Batch Text”, the default release of RMXP is broken and makes it so every 4th line of text becomes uneditable and disappears randomly. Basically, this is a horrible way to write text. If you're still desperate to use it and you hate awesome external documents, then I'll just have to teach you how to use this.

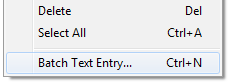
When you're actually writing dialogue, you'll have to remember to use the \n and | commands, because the word wrapping does not affect how the text displays ingame. if you fail to notice this part, you'll end up with text flying offscreen and immosubuke errors all over the place. After you've written your 4 lines of text, press OK, then open another Show Text box and keep typing.

Once you've written a few things, you might notice you goofed on a line or two, and you need to edit that text box. Right click on it...

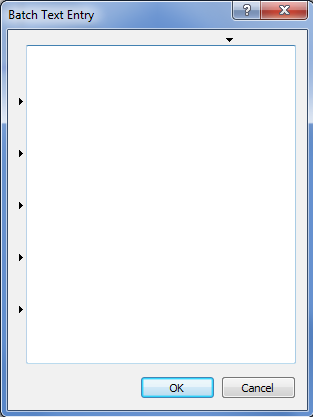


And click edit. Or, while the box is selected with a left click, press the space bar. This allows you to fix goofs and errors.

Uh what else. Right, to use the Batch Text function, awful as it is, right click in the event window, and select Batch Text.



Like so. After selecting Batch Text, you're greeted with this window:



If you do not have the patch that was created by some nice soul on the internet, get it [[HERE](http://www.hbgames.org/forums/viewtopic.php?f=179&t=66366)] and apply it to your RMXP. If you're too lazy to do that, press the Enter button to skip every 4th line or you'll have some broken text and it'll be ugly.

Of course, smart people use external documents but you can suit yourself however you like. I'm here to teach, not judge.

This concludes chapter 3. The next chapter will be all about Weapons and Items. It should be a relatively short chapter, and I hope it teaches you another 1/5th of the basics you need to know to make a decent game.

**Onward to victory!**

Chapter 4: Weapons and Items!

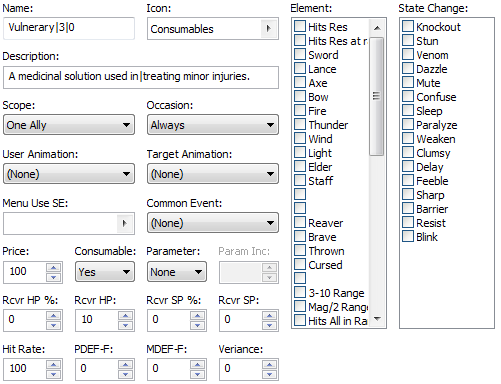
Weapons and Items are handled quite interestingly in FEXP. They took me quite a long time to learn how to edit them quickly and accurately, hopefully this will save you months of time and pain.

There were even some simple things I didn't know how to edit until a few months ago, when I started work on Generic War and PwnageKirby and I shared a lot of tips with each other. While I won't cover Skills in this chapter other than a few references, most of what I know about Skills should be credited to her.

Now then, let's start with Items. Open up the Database screen and flip to the Items tab.



Here, you'll see a lot of things to edit. Even if you're familiar with RMXP, I'll warn you now that everything is different.



There is a lot of stuff to cover in this window so if I miss something, just post about it in the guide topic and I'll try and include it in the future revisions.



This is the first thing to know about. You set the name, number of uses, and the icon number of the sheet it belongs to, along with any skills you want assigned.

**Vulnerary**: The name of the item. If the name is too long it'll scroll offscreen ingame and look generally awful.

**3**: This is the number of [max uses](file:///C:\Users\Tony\Desktop\j). If you set this to three, but place a unit on the map and give them a vulnerary and assign it one use in the event, it'll display as 1/3 uses ingame. You can also set it to 6 or 20 or whatever in the event and it'll display as 6/3 or 20/3 or whatever. What I'm saying is that this is for aesthetic purposes.

**0**: The icon number on the sheet this item is assigned to. Wait, what sheet? Look at the top right of the item editor and you'll see this:

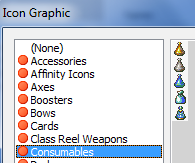


**Consumables**. This is a selectable option you can choose a sheet from. The sheets can be found in your FEXP 1.0.1<Graphics<Icons folder. Looking for Consumables.png in that folder gives us this sheet:



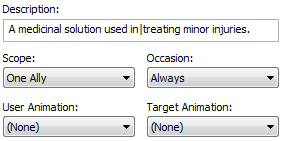
The vulnerary is the 1st number, which equates to 0 in FEXP. Then comes the second spot, which is a value of 1, and etc. PERSONALLY, I like to just make 16x16 icons and give them their own .png file by themselves and set the value to zero in the item editor, but this can really add a lot of files to your Icon folder if you're overzealous with icons.

If you want to change which icon sheet to link to in the editor, click that Consumables button and you get this:

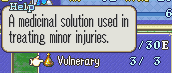


You can choose which sheet you like here and after choosing, edit the 0 in the name of the item to whichever slot you want. You can also edit the graphics for the Vulnerary icon by using your preferred image editor, just remember to save it as a [transparent PNG](file:///C:\Users\Tony\Desktop\j) so it shows up correctly ingame.

Now that we've covered the first tiny section of Items, let's edit some other stuff.



You can edit the item's description ingame here. Take note, this is only for the “R” description on the inventory menu. That is to say, this:

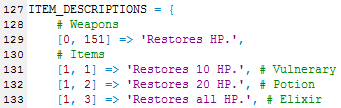


If you want to edit the Use Item description...

(Continued next page)



You'll have to go to the scripts to edit this. Go to the Script Editor and look at the Items script in the Configuration section. Scroll down near the bottom of that script and you'll see a giant wall of words.



This is where you edit those descriptions. Go wild and whatever. However, for the “R” menu descriptions, you do have these three restrictions:

-Only two lines of text allowed. (Obviously since this is FEXP this can be changed but whatever)

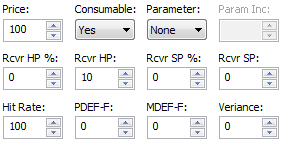
-The only command allowed is the | command, which drops the text to the second line.

-No apostrophes allowed. The lone exception is if you have a backslash before the apostrophe, like this: \'

Pretty easy rules to follow, in my opinion.

The other things I showed you, **Scope**, **Occasion**, **User Animation**, **Target Animation**, **Menu Use SE**, and **Common Event** seem to be unused in FEXP. Scope seems to be set to One Ally for the three healing items, but otherwise it's set to None for everything else. I have no idea what they do, but since they didn't do anything when I edited them over the last few times I tried, I just assumed they do nothing.

Next, let's look at these little thingamajigs.



Not gonna lie, I don't know what everything here does. Heck, I don't even know what they did in normal RMXP, let alone FEXP.

**Price**: This is the cost per use of an item to buy that item in a shop. In the case of Vulnerary, it's 100 per use, and costs 300 to buy in a shop because Vulneraries have 3 uses by default.

**Consumable**: Unused in FEXP. By default it's always set to Yes.

**Parameter**: Unused.

**Param Inc**: Also unused.

**Rcvr HP%**: This, in the case of potions, affects how much HP% you heal to yourself. (Items are self-use only in default GBA FE, but this can be altered in FEXP, naturally.) Elixir would use 100% here while Vulnerary uses a set number of HP to heal so it uses the next box.

**Rcvr HP**: This heals a set number of Hit Points. In the case of Vulnerary, that amount is 10. Potion is set to 20 by default.

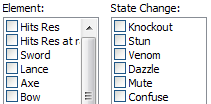
**Rcvr SP and %**: These are both unused in FEXP, since there is no SP in Fire Emblem.

**Hit Rate**: This affects the hit rate of an item. Lower hitrates below 100% mean the item will be less likely to hit. Useful for a ragefest or some other purpose.

**PDEF/MDEF**: I don't know if this does anything, never bothered to test it. Never really cared.

**Veriance**: (Misspelling of variance? Why, Enterbrain, why?) I don't think this does anything.

That covers the third main area of item editing. Next we take a glance at Elements and State changes.



(Continued next page)

Element is used to add special effects to an item. [(Note)](file:///C:\Users\Tony\Desktop\j) One thing you can do with Element is set an item to unsellable, at the least.

As for State Change, I think that once States are actually programmed into FEXP, you could make troll items that put the unit to sleep, or put the unit in a Trance state which gives Defense and Resistance and Heals HP, at the cost of being frozen for a turn, just things like that. Right now though, State Change is pretty much useless for items since States aren't coded yet. (Poison is though, yay!)

To edit the names of Elements, go to the tab at the top of your Database that says System, and on the left you'll see an editable window where you can change their names. Editing anything other than their names has to be done in the scripts though, so sorry.

As for States, you can edit those in the appropriate tab of the Database.



I never really touch these, since for the most part they're useless. If you're a scripter though, there's lots of room to add features for these in the scripts and I recommend it too.

All right, this about covers items. There's only one other thing I should mention about them though.



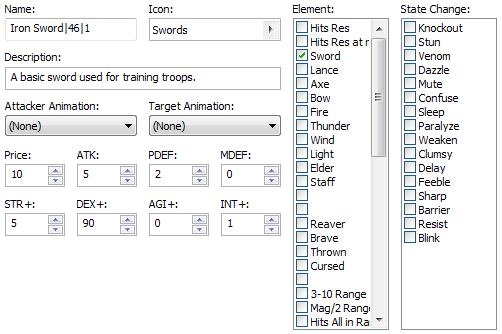
Do you see the 36 at the end of these two items? No? Well you're blind. The 36 assigns a Skill to an item, generally an [equip item](file:///C:\Users\Tony\Desktop\j) will have a skill assigned to it. To find which skill is equipped to an item, open the Skills tab in the database and scroll to the appropriate Skill ID #.



Nullify, in this case. To assign multiple skills, use dashes as a separator. 36-72-112 would assign skills 36, 72, and 112, as an example.

All right, enough about items. Other than assigning item effects in the scripts and the more complicated things of that sort, you now know basically how to use the Items tab of the Database. Now we move on to the Weapons tab as well. We won't be covering the advanced scripting parts of items and weapons yet, but in a later chapter I promise to get to those.

Looking at the weapons tab, we see this for the Iron Sword.



All right, less stuff to edit than on the Items tab, but more of it actually does stuff. Let's start with the basics again.



Most of this is similar to what you already know from Items, but there are a few differences.

**Iron Sword**: The weapon name, you get the idea.

**46**: Max number of uses, same as with items.

**1**: AHA! This is different, it represents the weapon's level. The possible entries are as follows:

**0** = PRF Weapon. You set this in the scripts, more on that in just a few seconds.

**1** = E rank.

**2** = D rank.

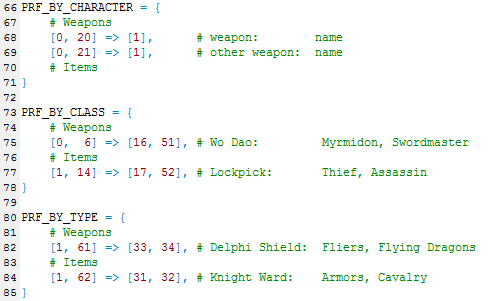
**3** = C rank.

**4** = B rank.

**5** = A rank.

**6** = S rank.

To edit PRF weapons, we have to go to the Script Editor once more. The methods for making PRF weapons are far more diverse and useful than in romhacking, as you'll soon see. Navigate to the Items script to find the section for editing PRF weapons and items.



You can assign PRF to characters, classes, AND class types. This is a huge step up from romhacking, which only allowed assignment to a specific class or character. You can make a sword be assigned only to light swordsman, or a tome only to heavy magi, anything is really possible. Heck, you can even [mix and match](file:///C:\Users\Tony\Desktop\j). Best of all, you're not limited to only (I think) 6 weapon locks total, like in romhacking, you can have unlimited locks! Now let's see how to format them.

**Prf By Character**:

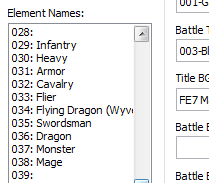
The first two digits select the weapon/item. [0, 20] in the first example chooses a weapon with the 0, and the weapon ID #20 with the second digit. As stated in a previous chapter, the 0 can be changed to a 1 to make it select from the Items list. After the =>, you get the digit [1]. This selects the character to assign it to. In this case, it's our Eliwood in Actor slot 1.

**Prf By Class**:

Mostly the same as the last one, except for the second digits, you can select as many class ID's to assign the weapon/item to as you like. In the first example, the Wo Dao is assigned to only the Myrmidon and Swordmaster classes. Simple, right?

**Prf By Type**:

To find the class types, you open up the Database again and flip to the system tab. All class types are listed here...



29-38 are the official types, but to be honest you can assign a PRF to anything in the Element list, even unsellable items and weapons or other complicated things of that sort. The possibilities are only limited by your imagination.

All right, enough about PRF locks. With this, you know what to do and we will get back to weapon editing. Open up your Database tab again and flip back to weapons.

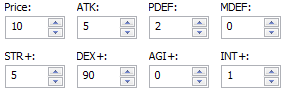


Weapons do not have a Use Item description, except in special circumstances. (Like if you recreated the Light Sword from FE5, but that's a bit advanced and you probably won't need my help by then.) Descriptions, I do believe, are limited to a single line of text, but they might extend another line with the | command. This is because weapons already display 2 lines of text for the weapon's stats, so adding another line makes the box pretty big. Adding two more looks [rather ugly](file:///C:\Users\Tony\Desktop\j).

Uh let's see, what to bring up next...



Haha these do absolutely nothing! Never touch these or you’re wasting your time!



These all do something and make up the meat of your weapon editing time. Let's learn!

**Price**: The amount each weapon use costs. To buy this iron sword, you need 46 uses times 10 gold, which equals 460 gold. Of course, the sell price is half price, 230 gold.

**STR**+: This is the Weapon WGT stat.

**ATK**: This is basically the MGT stat.

**Dex**+: This is weapon Hitrate.

**AGI**+: This is the Critrate for a weapon.

**INT**+: This is how much exp the weapon gives to your WEXP stat after a battle.

Alright, now for an explanation of PDEF and MDEF, because they aren't so simple.

**PDEF** is the icon # of the weapon icon for the selected weapon. Basically it's the same thing as the 0 on this sheet:

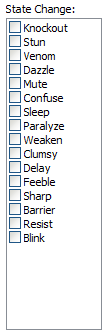
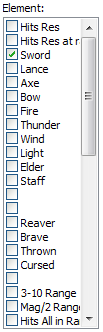


I detailed how this functions on page 35.

MDEF is for staves only. Because some staves do the exact same effect every time they perform an action (Like a Torch staff for example), they need a set amount of EXP. The MDEF box sets how much EXP (not WEXP) a stave gives the player.

For example, the Warp staff, though nonfunctional in FEXP currently, would give 85 EXP after every use.

Next we have States and Elements to explain. These are pretty similar to how they function for items, but a bit different because weapons are more awesome.

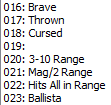


Anything in the Swords to Staff section changes what weapon type the weapon is. As an added bonus, setting a weapon to more than one type means only a unit with the correct [level](file:///C:\Users\Tony\Desktop\j) in both weapon types can wield that weapon.

Reaver, Brave, Thrown, and Cursed are all types that you can assign to a weapon. You can assign more than one and all their effects stack cumulatively.

Ranges are a bit more complex. To this day, I have not [found a way](file:///C:\Users\Tony\Desktop\j) to edit a weapon’s range in FEXP or how to add my own. The closest I could get was assigning a skill that increases a weapons range to that weapon, though swapping weapons would cause the game to crash in some instances.

Therefore, using the default ranges is your only option for altering a weapon’s range. Here is a list of what each of the 5 range modifiers alter:



**017**: This sets a weapon at 1-2 rnge and uses a throwing animation. Primarily used for javelins and handaxes, this will use no animation in battle if the class doesn’t have a throwing animation.

**020**: Used Long-Range magic tomes, this automatically sets the tome to [uncounterable](file:///C:\Users\Tony\Desktop\j).

**021**: This bases the range of the weapon by a character’s POW/2. If they have 10 POW, their range is 5. The weapon also becomes uncounterable.

**022**: Currently nonfunctional in FEXP. This would act similarly to Latona and the Fortify staff, I’m guessing.

**023**: Used by ballista, gives an item ballista range and makes the weapon uncounterable.

All the other element boxes are pretty self-explanatory.

For states, you can either set a + or a – in their boxes. A plus sign means that the weapon inflicts that state when it connects, therefore a poison sword inflicts poison status when it connects. A minus sign means it **removes** an inflicted state. This effect is typically used by restore staves so that they can heal status ailments, but you could make funny weapons with it if you wanted to.

I hope this chapter was informative. The next chapter will be about editing classes and will be quite detailed on the subject.

Chapter 5: Editing Classes in FEXP

This chapter will focus on editing and creating classes in FEXP. It will not explain battle animations or how to assign battle animations, those will get their own chapter later on. I will teach you the following things:

* How classes are edited inside the Database
* How to assign a class in the database to its animation and map sprites
* How to assign skills to a class
* How to change the base stats for generic classes, as well as growths
* How to set vision ranges in FOW for all classes and individual classes
* How to change promotion options for classes (Multi-Promo not supported yet)
* And other assorted things to edit on classes!

Well, life is short, so let's get started! Open up the Classes tab of the Database. A lot of this is really simple stuff so I'll just go over it briefly.



While “Position” does nothing at all, the class name does do something.The name of the class and the tier number of the class, extremely simple.

**0** would be like the trainees in Sacred Stones; Ross, Amelia, and Ewan.

**1** is like regular unpromoted classes; Soldier, Thief, and Cavalier as an example.

**2** is a promoted unit, like Halberdier, Assassin, and Paladin.

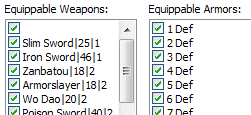
**3** Is basically Athos and Nergal from FE7.

On a side note, Class names, item names, weapon names, and character names can all use an underscore \_\_ to ignore everything else in the name after said underscore ingame. As an example...

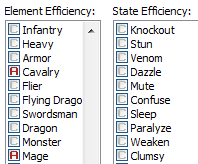
Troubadour\_Priscilla

This would make a class name of Troubadour appear ingame, and you could make multiple troubadour classes, differentiating each one's map sprite from each other.

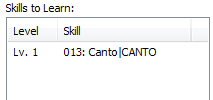
You can also use this for faces in the same way Pwnage Kirby did with Xiao Emblem to give all of her enemy units different faces. Her system is “Name\_1”, “Name\_2” and etc, then she assigns a Name\_# to each unit on the field, keeping them all the same name, but giving each their own face. Pretty cool huh?



Equippable weapons and armors should have check marks going all the way down. (Except the last 10 CON points, no idea why) I don't honestly know what these do, so I just copy-paste an existing class everytime I need a new one. Otherwise, I'd have to check every box all the way down and what a pain that'd be I\_I.



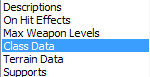
This is the interesting stuff. The efficiency ratings affect how much something affects a class. An “A” rank denotes they do extra damage to the class. A “C” rank indicates normal damage. An “F” rank denotes they do less damage against the class. With this, you could make it so any weapon with, say, a Poison State, would do extra damage against a class with lower Efficiency ratings. This is also where you assign class weaknesses to a class, as [seen above](file:///C:\Users\Tony\Desktop\j).

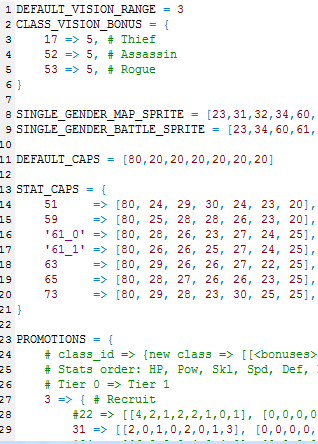


Down at the bottom right is where you assign Skills to a [class](file:///C:\Users\Tony\Desktop\j). Double click in the skills section to assign a skill of your choice. It's pretty self explanatory, just know that the Levels don't really do anything. You could code them to do something, but that's up to you. Also, only 4 skills are visible on a unit ingame at any time. Item skills will not show up in the skills, but weapon skills will.

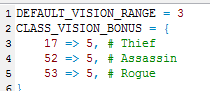
Personally, I recommend not going overboard with Skills. They can get hella annoying if you add too many to a unit. I will provide more information on skills in a large skills chapter later on.

Now that we've covered all the basics of editing classes in the Database, let's glance inside the scripts to see what other things are essential to get a class working ingame. Scroll down to the Class Data section of the Configuration scripts.



There's a lot of things to edit in this script, including vision range, default caps, stat caps for each class, promotion options, [terrain ignore](file:///C:\Users\Tony\Desktop\j) settings, movement values, making immobile classes (Like the Tent in FE7), and shortening class names ingame.

So yeah, there's a lot to cover. Let's start from the top and work our way to the bottom.



The default range for all units when it comes to FOW is 3 tiles. Thieves, Assassins, and Rogues also get 5 bonus range, (In addition to the 3 default) and you can easily [set your own](file:///C:\Users\Tony\Desktop\j) values here.

To make FOW visible, put this in your INFO event on the map:

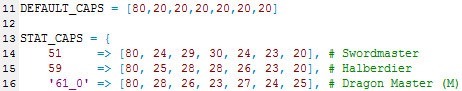
[$game\_map.fow = true](file:///C:\Users\Tony\Desktop\j)

I will provide more extensive information on FOW in a future chapter. I don’t actually know a lot about it yet but since one of the games I plan to work on in the near future will have FOW in the first chapter, this is something I need to tackle anyway.

Next, scroll down just a bit and we find this:

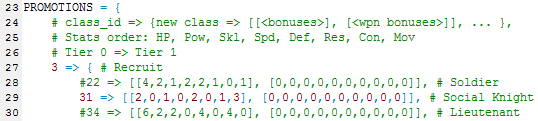


This tells the engine which classes have only one map sprite, regardless of their gender. For example, fliers and cavalry and armors usually only have one unisex map sprite. Helpful, in case you don’t want to make two sets of map sprites for each class.



The default caps for all classes, regardless of tier, is 80 HP and 20 for all other stats. You may modify this if you wish. The order of stats are as follows: HP, STR, SKL, SPD, DEF, RES, CON. You may notice in the above screenshot that the Dragon Master has a class ID of 61\_0, and in this case the 0 refers to the gender. The only valid genders are 0 and 1, anything else will be reduced to it’s male/female equivalent.

Next in this script we have...



The image is a bit blurry because I had to shrink it. Sorry.

This is the Promotions section of the script. Now, should I or Yeti or another scripter manage to get multiple promotions working, some parts of this will actually work. Right now you can only promote to one class which makes people who love the FE8 system feel a bit left out. If you're a decent/good scripter and want to help out with FEXP, multiple promotions are a solid starting point. For everyone else, lemme explain how this works.

Where you see the Class ID, 3, this is the promoting class. That is, the Recruit. The recruit will promote into one of the three classes listed in the indent below, and gain bonuses. The bonuses look confusing but they're actually easy.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID# | HP | POW | SKL | SPD | DEF | RES | CON | MOV | S  W  R  D | A  X  E  S | L  A  N  C | B  O  W  S | F  I  R  E | T  H  U  N | W  I  N  D | L  I  G  T | D  A  R  K | S  T  A  V |
| 31 | 2 | 0 | 1 | 0 | 2 | 0 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

These numbers are the bonuses the unit will gain upon promotion. A '2' in the HP slot means +2 to HP upon promotion. A '31' in the lances slot means getting a D in lances upon promotion. As a quick reminder about weapon levels...

0 = No weapon level

1 = E rank

31 = D rank

71 = C rank

121 = B rank

181 = A rank

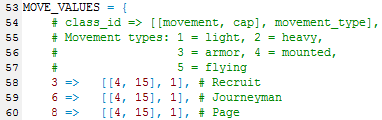
255 = S rank

When it comes to multiple promotions, should someone ever code it, the goal would be to integrate it into the already built-in and nearly ready system Yeti made. You would just put any promotional options in and they would add together and be visible ingame. Basically if you removed the #Comment signs above, you'd be able to promote into a Soldier, Social Knight, and Lieutenant from the Recruit.

Next, let's look at this little snippet that Yeti handily added to the engine.



Yeti had his own default balances implemented for the FE7x game he was making before switching to XNA. This would make mounted units ignore terrain avoid and fliers would also ignore terrain DEF boosts on top of it. The numbers seen above do not refer to class ID's, but movement types. To find the movement types, look just a smidge lower in this script.



Here you can edit movement types each class belongs to. As you can see, 4 and 5 refer to mounted and flying types, respectively. Movement types do not affect the number of tiles a unit moves, but they affect Terrain Costs, which I'll cover in this guide at some point. Though, they're pretty complicated and not fun at all :<

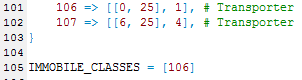
First number as always is the Class ID. After that comes the edits to the class.

3 => [[4, 15], 1], # Recruit

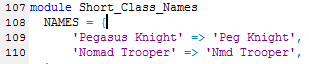
**4** is how many tiles the unit can move, which is 4 tiles in this case.

**15** is how high the movement can be capped at while playing the game.

**1** sets the movement type.



Scroll down to the last of the Movement Types and you find the storage classes, like the tent and transporter. These are fairly straightforward, but in case you're wondering what the “Immobile Classes” does, it makes a class unselectable on the map. If you played FE7, you might recall putting your cursor on Merlinus' tent and pressing the “A” button, only to discover you couldn't move him or even select him at all. FEXP handily [duplicates](file:///C:\Users\Tony\Desktop\j) this function.



A quick note about this section, which shortens class names ingame. In the editor, Pegasus Knight is what's displayed, but thanks to this script insert, you can shorten the name of your classes ingame so they don't scroll offscreen.

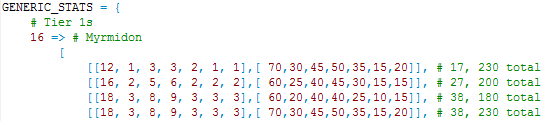


Like so. As you can see, the extended name crams into the battle sprite and looks rather ugly. ~~The extended name also seems to have given her an HP boost!~~

This completes the first half of the Classes section. The Class Data script is where most of your edits to the classes in FEXP will take place, and is vital to understand what everything does so you're not totally confused.

Now, we need to go to the second most important script when it comes to editing classes: the Unit Info script. Yes, we covered this in the second chapter, but we need to cover the parts we missed back then as well.

Open up that script and scroll down to this section:



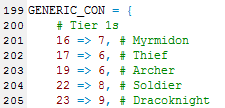
As I stated in the second chapter, these are build types. The first one correlates to build 0, the second to build 1, and etc. Add, remove, and edit build types for the pregenerated unit type here. But what do all the stats do?

BASE STATS BASE GROWTHS

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HP | POW | SKL | SPD | LUK | DEF | RES | HP | POW | SKL | SPD | LUK | DEF | RES |
| 12 | 1 | 3 | 3 | 2 | 1 | 1 | 70 | 30 | 45 | 50 | 35 | 15 | 20 |

The first section is for base stats, and the next is for base growths for each build. The strategic possibilities for multiple builds, unlimited generic units, 999 character slots and etc are quite mindblowing. Naturally, the speed of FEXP is a huge issue but if you don't care about that then the lack of limitations on your game's size can be pretty useful.

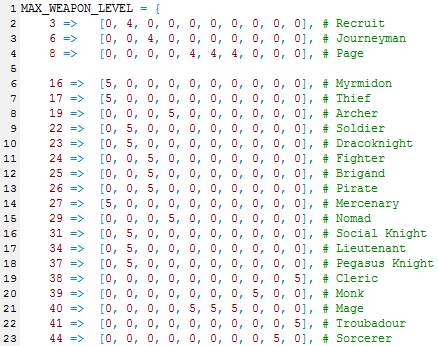
Now scroll down a bit lower.



This defines generic CON for each class. If using a pregenerated unit, set their con to 0 and they will use the generic con instead.

Finally, let's scroll to the last main script in the Configuration

section we'll be needing to edit; Max Weapon Levels.



If you've paid any attention at all to the times when I listed the order weapon levels always go in, you'll know it's very simple. But heck, I'll list it again, for shits and giggles.

Sword, Axe, Lance, Bow, Fire, Thunder, Wind, Light, Dark, Staves.

The numbers you can input here correlate to the highest weapon level that unit can reach. For example, all T1's generally only reach “A” level, or the number 5. All T2 units generally reach “S” level, or 6, though [only one](file:///C:\Users\Tony\Desktop\j) of their weapons can “S Rank”.

As a reminder, these are the numbers for the weapon levels:

0 = No weapon level

1 = E rank

31 = D rank

71 = C rank

121 = B rank

181 = A rank

255 = S rank

Take note that Yeti had also programmed the Hero to get an additional maximum of “B” rank in Lances and Bows, though this is not edited in the Max Weapon Levels section, but in the Skills section instead.

Now, let's say you want to change which map sprites a class links to, or maybe you just want to rename a class, period. Well, it's time to again open up your game's folder and go to the graphics section. This time, open the Characters folder.

(Continued next page)



We'll use the Archer as our example. Classes need certain things to display ingame properly. They need map sprites, and they need database information, and they need some script information.

MapArcherM and MapArcherM\_move are what assigns this sheet to the database entry “Archer” and the generder entry “Anything even numbered”. When the game loads up the Database names, it searches the Characters folder for anything with a prefix of Map and if something has that prefix, it searches the rest of the title for a class name that's in the database, such as Archer. Then, it finds the gender prefix and loads that specific map sprite for that class.

If the map sprite is [misaligned](file:///C:\Users\Tony\Desktop\j) it will display weirdly ingame and look awful. One of my goals for Super FEXP was at least adding all the map sprites for all the classes, but I could never figure them out. If anyone else knows and wants to help me out, hit me up with a PM at either [Serenes Forest](http://www.serenesforest.net/forums) or my site, [Klokreations](http://www.klokreations.net), and I'll edit this whole section here.

So anyway, to change the name of the Archer ingame, edit the ArcherM to something else, like BowyerM. Next, go into the Database and change the name of the Archer class to Bowyer.

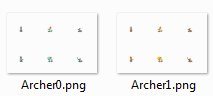


Success! To assign a battle animation to your class, keep in mind that Battle Animations are also assigned to classes by their name.

Go into the animation folder and find the Archer battle sheet.



Edit the part of the sheet that says “Archer” to Bowyer, then go to the Battlers folder and change the name of the Archer sheet to Bowyer as well.

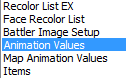


This should make you golden!

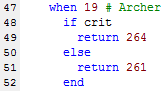
Remember to edit the Descriptions script too if you want your new class to have a description. Base stats and other class information are purely assigned to class ID so unless this is a new class and you need to reassign them or assign new ones to the new class, you should be good to go.

While there will be a section of this guide dedicated purely to Battle Animations in the future, I thought I’d show you some of the basics of editing Battle Animations now. These are really just the easiest easiest parts though, it won’t all be this easy.

Open up the script editor and go to the Animation Values script.

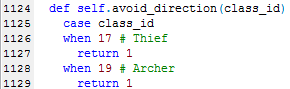


Here you’ll see a long, long, looooong list of values and numbers and syntax and dkjguyit GODDAMN THERE IS A LOT HERE. Point being, this is pretty overwhelming the first time you see it. But really, it’s not too terribly hard.



C:\Users\Tony\Desktop\FEXP Tutorial Image.png

C:\Users\Tony\Desktop\FEXP Tutorial Image.png

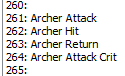


Where you see the number 19, this is the class number. In the case of magic tome animations, this would also refer to the ID of that tome.

The other numbers that are from 261-264 are the animation values. If you open up the database and go to the Animations tab:

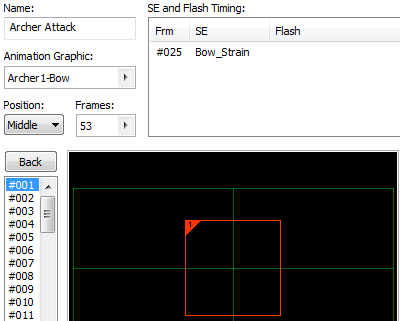
C:\Users\Tony\Desktop\FEXP Tutorial Image.png

You can find the values for the archer in the 261-264 ID ranges.



The names for each entry tell you what it’s function is in the scripts. I used to think that each word was a keyword that activated an effect, but I don’t think it does now. Yeti might clarify this for me.

Clicking on an entry and looking to the right shows us a window like this:



This window tells the engine what the animation looks like.

You can choose the graphic for the animation from the [animation graphic](file:///C:\Users\Tony\Desktop\j) button, you can paste animation frames from the sheet into the animation window, rotate them around, define transparency, set sound effects, all sorts of cool stuff.

For now, that’s all I’m going into. More detail in a later chapter, as I always say! It’s much too soon for your virgin mind to be exposed to how this complicated stuff works so I’ll spare you the agony.

This concludes chapter 5. The next chapter will be about Basic Chapter Creation in FEXP, and will outline the steps to making your own chapter from start to finish.