

# Introduction

This tutorial will focus on giving you the knowledge that you will need to create your own in-game cutscenes (called “events”). Events can do a multitude of things, such as animating character sprites, giving the player items, changing a character's sprite/palette, and more! Any time a dialogue box displays or an NPC moves along a fixed path, there are events at work. This tutorial will focus only on events that occur outside of battle.

You don't need to be a programmer to understand how to make your own events. More or less, you'll just be giving the game a sequence of commands, which the game will follow when the event happens. Once you understand the process, all you'll need to do is string together the correct event commands to get the desired effect. By the end of this tutorial, you will have learned:

1. What tools are available to help you
2. How to make events “happen”
3. How to modify events in the game ROM
4. How to learn new event commands

## Tools

### Event Editors

While there are comprehensive editors for other parts of the game, such as sprites and monster stats, there is no silver bullet for event editing. One existing event editor is called “ZoneDoctor”, which is a modified version of FF6LE (the map editor for FF6). However, ZoneDoctor is considered to be fairly unstable, so it is risky to use. It is also somewhat limiting compared to editing events manually. If you are aware of ZoneDoctor's quirks and don't need the extra power, it might be worth using, as it can help to visualize the events that you are creating. Otherwise, it is probably better to avoid it. If you do want to download it, it is available [here](#).

There is another editor that can be used to create events, packaged with Everything's [FF6Tools](#). This editor is only compatible with macOS, so you will only be able to use it if you have a Mac computer or install a virtual machine. For the moment, this guide will focus on editing events manually. More information about FF6Tools may be added at a later date. That being said, this tutorial will teach you some concepts that should be useful regardless of whether the events are created manually or through FF6Tools.

### Manual Editing

When manually editing events, several tools will be of use to you:

- [FF3usME](#): You've probably seen this one around the wiki a few times already, and with good reason! This editor can do a lot. Of particular note is its ability to edit dialogue, but it can also provide useful data for various IDs. For example, you may look up a monster formation ID if you want to force an enemy encounter upon the party, or you may need the ID of an Esper to give the party a specific Magicite.
- [FF6LE Rogue CE](#): The main purpose of this tool is to allow you to edit maps. That's outside the

scope of event editing, but this tool is helpful for other reasons. Like FF3usME, the LE is a great source of information on various IDs. It also allows you to place NPCs and event tiles in addition to setting entrance events (to be discussed below).

- [Event Commands Document](#): This document is a list of all the event commands and how to use them. Examples of how to use the commands are provided, which should hopefully make it easy to use unfamiliar commands once you have learned the basics.
- [Event Bits Document](#): A list of event bits. Once you know how to read/use it, it will be fairly straightforward.
- [Event Script Dump](#): The holy grail of event hacking. This document provides a way to peruse through any of the game's events in a human readable format. Don't leave home without it!
- [HxD](#): This is a hex editor. It will allow you to modify the game directly. HxD is recommended over WindHex because it offers a sane way to copy-paste chunks of game data.

## Headers

Before we begin anything else, let's talk about the number one thing that trips up the unaware. A header is a chunk of \$200 bytes that appears at the beginning of some SNES ROMs. The keyword here is *some*; if you download a random FF6 ROM (whether it's version 1.0 or 1.1), it may or may not have a header. You need to determine if the ROM is headered or not before you begin. Open the ROM in HxD. The ROM should look similar to one of

these images

.

As you can see, the first \$200 bytes of the ROM on the right are all zeros; this unnecessary data is the header. Functionally, the two ROMs are identical. The only reason why the header matters (for our purposes) is that it pushes all the data forward by \$200 bytes. This means that the game's data won't be where you expect it to be. This problem could technically be remedied with some simple math, but it's easier to just use a ROM without a header. You should either continue searching for ROMs until you find one without a header or remove the header by <doing X, I forget the best way to do this>. As an aside, I also recommend using version 1.0 for hacking.

## Addresses

If you open up the Event Script, you'll probably notice some values at the left side, such as CA/0000. These are addresses. Addresses are a way to uniquely identify any particular byte in the ROM. Every byte in the ROM has its own address. If we have looked in the event dump and found an event that we want to edit, we can use the address to find that event in the ROM data as well. Take a look at CA/0010. We see that the byte at CA/0010 is 4B. 4B happens to be the command that displays some dialogue. The event dump also tells us that the dialogue that would be displayed is "Found <N> GP!". So this event is called by the game whenever you receive GP from a chest! To be specific, this part of the event only controls the dialogue box, so it doesn't actually give the player any GP.

Let's say that we want to modify this event. That would be strange, but we'll pretend that's our goal for the sake of the example. The first step would be to find the event in the game data. Unfortunately, CA/0010 is not the address we need to search for in HxD. <I don't know much about Hi-ROM offsets,

so it's hard to give an explanation here> Therefore, we need to search for the address A0010 instead. You can go to an address in HxD by clicking on Search > Goto, or using the Ctrl-G keyboard shortcut. Once we have reached A0010, we can see that the byte here is indeed 4B, so we are in the right place:



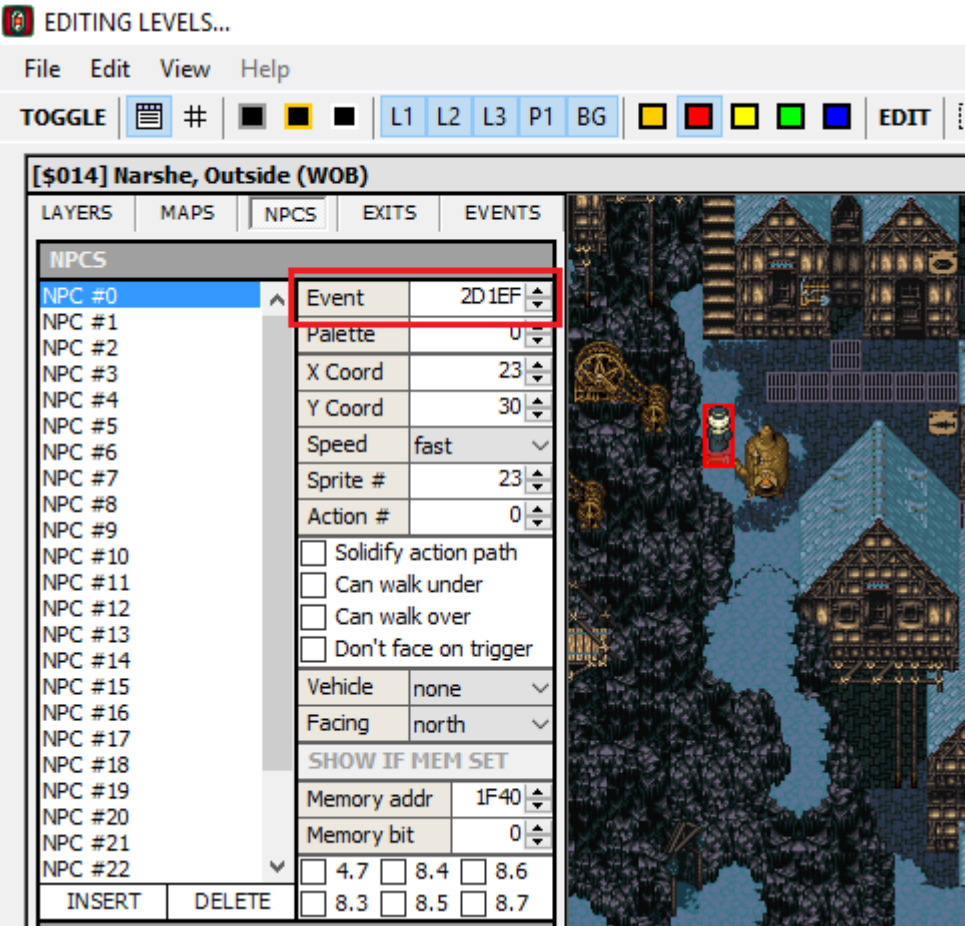
Therefore, if we wanted to modify the game so that something other than a dialogue box occurs when the party receives GP, this is the part of the ROM that we would have to modify.

## Starting an event

So how do events begin? There are several different ways that an event can begin:

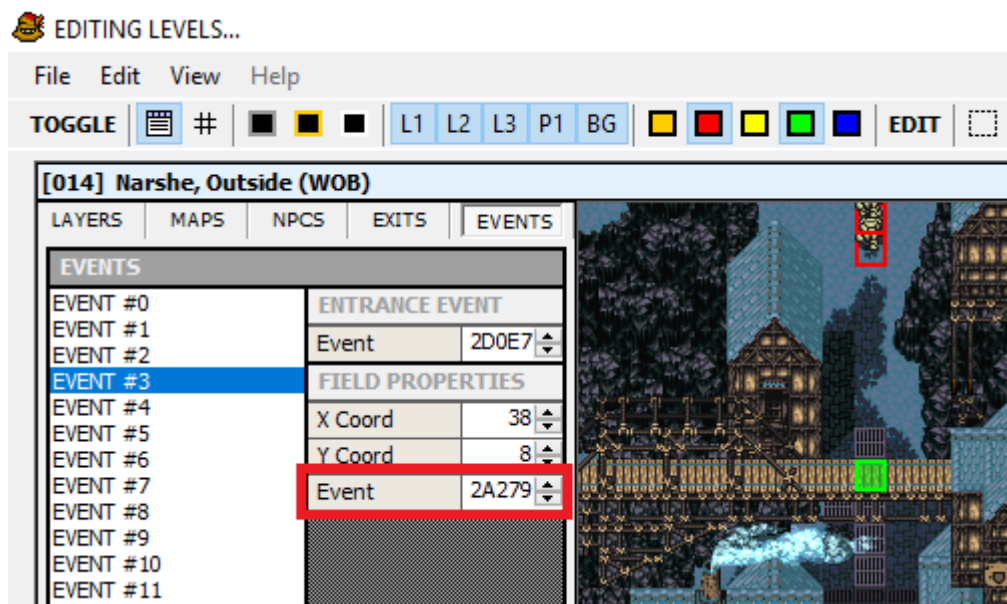
At the start of the game, the opening cutscene of the game will occur. This cutscene seems to start at CA/0003, but it immediately jumps to CA/5E33.

An event will begin when you talk to an NPC. You can find out which event will occur when you talk to an NPC by looking it up in FF6LE. Open up the LE and select a map with some NPCs on it (for example, "Narshe, Outside (WOB)"). You will also need to click the red square at the top of the screen to make NPCs visible. Now you can select an NPC and view its properties. In particular, we can see the address of the event that the NPC calls. Clicking on NPC 0 (the old man to the left of the weapon shop), we can see that the event he calls is at address 2D1EF.

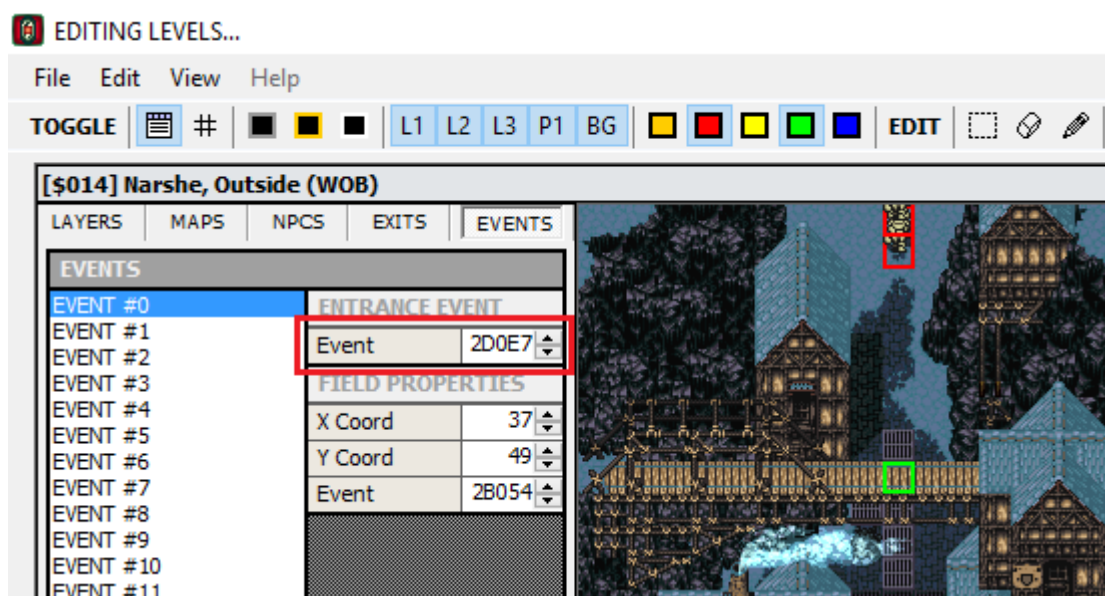


Unfortunately, this address is not in the format that we want it to be in. We need to add \$CA0000 to convert the address from the way it appears in the LE to the way it appears in the Event Script. In case you're wondering about the meaning of the \$ sign, it indicates that a number is in hexadecimal (base-16). You may find the Windows Calculator app (or your favorite alternative) to be handy for calculations like this if you set it to "Programmer" mode. \$2D1EF + \$CA0000 = \$CCD1EF, so we can find the old man's event at CC/D1EF in the Event Script. As you may have guessed, this event just displays another dialogue box.

Another way to begin an event is by using an event tile. In the LE, click the green square at the top to show event tiles. When the party steps on one of these invisible tiles, an event will begin. For example, when Terra steps on the green tile that you see on the bridge, it will trigger the cutscene where four guards look at her and shout "She's up there!" If we click on the event tile, we can see that it calls the event at 2A279 (or CC/A279 in the Event Script).



Finally, there's the entrance event. Whenever you enter a map, an event is called to initialize the map. This is necessary to initialize NPCs that walk on a fixed path, among other things. Narshe's entrance event is shown to be at 2D0E7, or CC/D0E7. If you're clever, you might notice that this entrance event affects 9 NPCs and opens the secret cave if the party has discovered it.



## Event Commands

Each event command causes “something” to happen in the event. That “something” can be rather diverse. You've already seen the event command that causes a dialogue box to be displayed, but event commands can also make the screen flash, give a character status effects, etc. Every command consists of a single byte followed by some parameters. The first byte indicates what the command does, and the parameters give the game extra information that the command needs to function. For example, take our command to display a dialogue box. Earlier, we saw in the Event Script that this command is \$4B. We can use this information to look up the command in the Event Commands Document and learn more about it.

In the Event Command Document, we see that command \$4B has two parameters. Each parameter is



also one byte in size. The document tells us to refer to command \$48 to find out what the parameters do. Command \$48 also displays a text box, but it does not force the player to confirm the text box before continuing the event. Both commands need two parameters in order to tell the game *which* dialogue to display and *where* to display it. The document shows a couple of examples of how this works.

If you wanted to display text box \$0123, you would flip the two bytes around, as shown in the example. Therefore, our two parameter bytes are \$23 and \$01. If you see 4B 23 01 in the hex editor, it might mean "Display text box \$0123 at the top of the screen inside a text box, and wait for the player to press A before continuing the event". The second example is meant to demonstrate how to move the text box elsewhere on the screen. The document shows a list of values that we can add to the last byte, depending on where we want the textbox to be. For example, we need to add \$C0 to the last byte to display text at the bottom of the screen and without the blue background. Therefore, 4B 23 C1 might mean "Display text box \$0123 at the bottom of the screen and outside a text box, and wait for the player to press A before continuing the event".

That's a lot of info to process, and if you're unfamiliar with this process, you may need to reread the above example several times to digest it. The good news is that a large number of event commands are structured in a similar fashion. If you can understand each piece of information above, you can reuse that knowledge when figuring out the other event commands.

In general, the Event Command Document is built around the examples. Often the first example is the most straightforward, and the later ones introduce complications or make a specific point. Each parameter's purpose is explained alongside the example. If you understand what the first example is doing, you should be able to build off of it afterwards. It's fairly common for bytes to be reversed from what you'd initially expect, as in the example above. We were trying to display text box \$0123, and we had to reverse it to 23 01 when we were figuring out the parameters. Keep this in mind when interpreting the examples.

Now you have all the nuts and bolts out of the way, so you can move on to [creating an event](#).

From:  
<https://www.ff6hacking.com/wiki/> - **ff6hacking.com wiki**

Permanent link:  
<https://www.ff6hacking.com/wiki/doku.php?id=ff3:ff3us:tutorial:events:background&rev=1525560763>

Last update: **2019/02/12 10:24**

