AW School Modding Guide

The Basics

All mods for Accidental Woman make use of the JSON data format, and if you are using a code editor, you can save the mod as a .json file to make working with it easier (then rename it to `.awm` when you're done). Fortunately, the JSON format has plenty of tools that can make working with it easier! When you're done, try checking your data with a JSON validator like this one. It will catch any errors with the format or syntax, which will save you from trying to guess why a mod isn't working. There are JSON editors available to use online, but you can also use something as simple as notepad. Avoid using a word processor such as Microsoft Word or Google Docs, as they will make creating a valid JSON much more difficult.

If you want to use images for your mod, you'll also need to make an image AWM file. Note that if your image filenames should start with `modIMG` so keep that in mind when specifying their names in your school mod. Note that school mods only function in version 0.19.0 and later!

All values explained in these instructions are mandatory unless it explicitly states otherwise!

Finally, these instructions are intended to be used alongside the official Oppaido mod. The Oppaido mod is a completely new school for the game, and serves as an example of how to make your own school. These instructions won't make nearly as much sense without it! Go ahead and open it up in notepad or your favorite text editor. It may be helpful to turn off word-wrap to make the format more legible, depending on your editor.

Terminology

In these instructions and several others, very basic terminology is used to help describe the JSON format. Most of this is simply related to the type of data being used.

Property: You can think of this as a variable name that holds a value. Example: `{"me": "ThaumX"}` in that example, "me" is a property, and it contains the value "ThaumX". Sometimes property names ("me" is the name of the property) are called keys.

Number: Numbers are numeric values like 1 or 539. They are not surrounded by quotation marks.

String: Strings are basically alphanumeric values wrapped in quotation marks. `"this is a string!"` Property names (keys) are almost always strings.

Boolean: Booleans are basically a true/false value, as in Boolean Logic. They are always written as either `true` or `false` without quotation marks.

Object: An object consists of one or more properties inside curly braces `"ex": {"age": 374, "species": "Erolich"}` in the example, the "ex" property contains an object. The object itself contains two properties: "age" and "species", each which contain their own value. Objects tend to form a tree or branch-like structure of data.

Array: An array is like an object in that it contains values, but these values don't have their own property names. Instead they use an index, which is simply the number for the position of the value in the array. `"fruit": ["Apple", "Banana", "Cherry"]` That example array "fruit" contains three values. Index numbering starts with 0, so index 0 = "Apple" while index 2 = "Cherry".

Main Information

This section covers the basic information about the school, and also happens to be the first chunk of data in the Oppaido AWM file. The properties covered here set the basic information for the school, such as it's name and hours of operation.

"key": [string] This is the property name or key that the school will be saved as in AW's game data. It should be a uni que name unless you intend to overwrite other schools.

"name": [string] This is the proper name of the school as it will appear in the game.

"desc": [string] This is a description of the school. This property is typically a couple paragraphs in length, however there are no max or min length to this value. In the base game, this value describes the school in general, the types of things taught there, and the instructor.

"days": [array of Boolean values] This property determines what days the school is open. It must contain a total of 8 true/false values. A true value indicates that the school is open that day. False indicates it's closed. The very first value, index 0, is whether it is open on holidays and should probably be false. Index 1 corresponds to the first day of the AW week, Monday. Index 7 is the last day of the week, Sunday. The school MUST be open on days that a class is being held!

"hours": [array of numbers] This property states the opening and closing time for the school. The first value (index 0) is the first hour the school is open. The second value (index 1) is the hour that the school closes. The time is in 24-hour format. Classes should start during the school's open hours (or else players won't be able to attend).

"baseprice": [number] This is the basic price level of the course. A very cheap school would be 6, while a very expensive school could be 15 to 20.

"member": [Boolean] This indicates whether the player needs to sign up for a membership to take lessons at the school. Think of something like a gym membership, for example. Players will have to cancel their membership at a school to stop taking classes there. True = requires a membership

"hooks": [array of strings] You should probably leave this value set as `["none"]` but can enter hooks that would signal to other parts of the game that certain types of things are taking place. This system isn't finalized yet so there are no example hooks.

"instructor": [array of strings] This contains the name of the instructor/owner of the school. The first value is the first name, the second is the last name, and the third is a nickname or "none".

"loc": [array of strings] This is the location of the school. For new schools created in the mod, it will always be `["downtown", "northwest"]` You will only need to change this value when replacing a standard school.

"img": [string] This is the id of the school's title image. The title image should be about 450x300px, but smaller sizes can work.

"courses": [array of objects] This is where the different courses offered by the school are defined. See the courses section below to learn more about it.'

"classContent": [array of objects] This is where extra content for classes is stored. See the section below.

"events": [array of objects] This is where events for the school are stored. See the events section below.

Courses Information

This section describes how to define the different courses offered at a school. Having different courses is useful because it allows different times/days that a course can be held, and also allows for classes to focus on specific or more advanced skills. A school must have at least one course. While there is no limit to the maximum number of courses a school can have, limiting it to 12 or less is a good idea.

Each course object must contain the following properties:

"name": [string] proper name of the course ex: "Advanced Oppaido".

"days": [array of Booleans] Like the days property described above, this is an array of 8 true/false values. This value sets the day/s that the course is held. `[false, true, false, true, false, true, false, false]` for example, shows that the course is held on Monday, Wednesday, and Friday.

"time": [array of numbers] This is the time that the course starts and ends. This array includes minutes values, unlike the previous hours property that sets the open hours for the school. The format is [hours, minutes, hours, minutes] The first two values are the start time, and the last is the finishing time. Hours are in 24-hour format, while minutes should be a number from 0 to 59 (minutes).

"priceMod": [number] This is the relative price of the course compared to other courses at the school. Introductory courses, or courses with few hours per week will have a lower priceMod, while advanced courses and courses that have a lot of hours per week are higher.

The priceMod is currently balanced so that each "hour" of instruction is ranked between 1 and 5. (A 50-minute block is considered an hour.) 3 would be the average rate. 2 would be cheap. 1 would be ludicrously-low, 4 is more expensive, and 5 is very expensive for very advanced courses. You can "rate" a course to be something like 2 ½ per hour, however the final value entered for the priceMod must be a whole number. The prices should also relate to the "train" property below. When you have a per-hour value for the course, such as 3, multiply that by the number of hours in the course. Value: 3 * Hours: 3 = PriceMod: 9

"duration": [number] This is how long each session of the course lasts in minutes. Remember that there is usually break or changing time factored in, so a normal "one-hour" class is probably 50 minutes. Classes should not go beyond 180 minutes, and in general should probably last 2 hours or less for most classes.

"train": [object] The train property contains information on what skills are being trained by the player. It is an object that contains properties holding arrays. The property name is an abbreviation for the skill being trained. The array contains two values, the number of attempts, and the difficulty of the attempts. Example:

`"train": {"ath": [2, 15], "dance": [1, 12], "exercise": 2}` In this example, we are training athletics and dancing (set "exercise" aside for now). Athletics here gets 2 attempts to earn a skill point, while dance gets 1. The difficulty of athletics is 15, which is moderate and appropriate for an intermediate level course. Dance, on the other hand, has a difficulty of 12, which is the value for a secondary skill of an intermediate course. The exercise value is used to keep track of the player's exercise for fitness and weight purposes, and doesn't actually contribute to earning skill points.

As a reference, 1 point of exercise equals 15 minutes of intense exercise (sprinting), 30 minutes of moderate exercise (jogging), or 1 hour of light exercise (walking). Exercise performed in a class environment doesn't improve the player's mood. Don't include exercise in courses that don't have physical activity.

Training difficulty is based on the skill check system. The higher the value, the more likely a character is to earn a skill point, but only if they pass the skill check. Setting a class's training difficulty too high will cause the player to fail more checks, and thereby have fewer opportunities to earn skill points. Training values are usually separated into primary and secondary skills. Secondary skills are incidental skills gained that aren't the primary purpose of the class. In a dance aerobics class, for example, you might gain a little bit of skill in dancing, but that isn't the primary purpose of the class. This is represented by a lower difficulty value. A course should generally have no more than 3 attempts at training for a one-hour session, and not all those attempts should be in the primary skill.

Course Quality/Difficulty	Primary Skill Difficulty Range	Secondary Skill Difficulty Range
Beginner	11 – 13	8 – 9
Intermediate	14 – 16	10 – 11
Advanced	17 – 18	12 – 14
Master	19 – 20	14 – 16

Skill Abbreviations List

exhib: "exhibition", ho: "prostitute", sex: "sex", oral: "oral", sed: "seduction", com: "comm", org: "org", prob: "probSolving", fin: "finance", art: "art", ath: "athletic", dance: "dancing", clean: "clean", shop: "shop", cook: "cook"

"desc": [string] This is a brief description of the course that explains what it is about compared to other courses.

"req": [object] (optional – use "none" string instead of object if there are no requirements) This is a way to specify skill requirements to attend a course. Example: `"req": {"ath": 50}` This example requires an athletics skill of 50 or more to take the course.

"img": [string] (optional – use "none" if no image) This specifies the title image for the specific course.

Class Content Information

Class content is the content that appears when the player actually attends a course. Each content object can be displayed when a course is taken, and if there are multiple content objects valid for a course, it will randomize to choose the content to display each time the course is attended. While having any content objects is optional, It's good to have a class content object valid for every class to avoid a mostly-empty "attended class" popup. This could be a single content object valid for all courses in the school, or it could be numerous content objects, with multiple valid objects per course.

Each content object must have the following properties:

"course": [array of strings] This array of strings specifies the specific courses the content is valid for. It could have a single string value if the content is only valid for one course, like `"course": ["Course A"]` or it can have multiple course names if it is valid for multiple courses like `"course": ["Course A", "Course C", "Course D"]` Make sure the names here exactly match the string used in the name field of the course object.

"img": [string] (Optional – Use "none" if no image) This is where you can specify an image showing a scene from the class, such as participating in exercises or getting changed, etc. 😉

"content": [string] This is where you describe what occurred during the course. You can use twee markup for this, including twee macros like <<if>> <<else>> <</if>> or any other markup. Remember to add some html formatting such as <br><br> to break up large chunks of text into paragraphs (you can also use <p>blah</p>, of course). You can even include links or buttons, but don't count on the player always clicking them, they may close the dialog window instead.

Event Information

Events can be quite simple or rather advanced. Advanced events are going to require solid JavaScript coding knowledge to get working properly, but simple random events are quite easy to create. The "events" property is an array that contain zero or more event objects. You can exclude this field by simply entering `“events”: {}` to leave it empty.

Events contain a couple parts. The properties that determine when an event occurs, the content to show when it does, and a callback function to execute when the event is over. Unless you are familiar with JavaScript, you should set `"condition": "true"` and `"effect": "none"` as these are expecting JavaScript functions in string format.

Each event must contain the following properties:

"random": (Boolean) This determines whether the event is set to occur randomly or not. If you are setting "condition" to true, then "random" should also be true. (otherwise, the event will always occur)! You can set this to false to use the "condition" property to allow it to occur under specific circumstances.

"odds": (array of numbers) this is an array of two numbers that determines the chance of the event occurring. The first number (index 0) is the threshold, and the second (index 1) is the max value of the random number. The game uses the randomizer to generate a number from 0 to the max value. If the generated number is less than the threshold, the event will occur (assuming the "condition" is met, if there is one). Example: `“odds”: [1, 9]` In this case, the game will generate a number with a value from 0 to 9. If the value is less than 1 the event will occur. This is a 1 in 10 chance of the event occurring (remember that numbers start from 0, and 0 to 9 is 10 numbers). You could also use something like `[3, 3]` if you wanted a 75% chance of the event occurring, or `[1, 49]` if you wanted a 1 in 50 chance.

The first and second value must always be 1 or greater!

"condition": (string [JS code]) The condition argument is a string version of a function that will be evaluated and run to determine if conditions are met to play the event. Returning true will allow the event, returning false will prevent it. The function is supplied with two arguments: "school" (string), and "course" (string) these are the keys for the school and course being taken.

You can also use basic JavaScript logic in this property to check for a specific course, or other simple check. This example shows how to make sure the event only plays when taking "Course C":

```
“condition”: “course === ‘Course C’”
```

Take note of the apostrophes used instead of quotation marks inside the string

"image": [string] (Optional – use "none" for no image) This is a special image you can show when the event occurs.

"content": [string – twee] This is the content (text) that is displayed when the event occurs. It will appear inside an interact window. You can use SugarCube macros as well as html formatting in this content.

"effect": [string – JS function] This is a callback that is ran when the interact window containing the event is closed. There isn't much need to use this specifically for schools, but it is included for special circumstances where it may be handy. It's probably best to avoid using this unless you have a good reason to.

Modifying the Official Content

If you give your school mod the same "key" value as an official school, you will overwrite that school! That means that your mod needs to contain ALL the information for that school, even if you are only changing 1 thing.

If you would just like to add to an official school's content, a much better route to go would be to contact ThaumX via email or discord and submit the content you'd like to add to be included in the main game. This will keep you from needing to update your content manually if the official school gets more content in a new release.

To modify the official school using a mod, search for the information in the game's .html file and then enter it in your mod in JSON format. You can then change whichever values you'd like or add whatever you want in terms of content.

If people are interested enough in a simpler way of modifying official content, we may eventually add in the capability, but it's a significant amount of extra work to implement consistently so it isn't currently planned.

We won't support / troubleshoot bugs caused my changing official game content, as we do not support bugs caused by modding anyway… If you're having issues making a mod, or your mod is causing bugs, do feel free to drop by the Discord server and we can try to help you out. Basically, we'll try to help mod authors if they need it, but it'd be impossible to support problems that occur when people are *using* the mod. As a long time mod author, I want to be supportive, but there are limits to what's possible. Even big-budget games with dozens of people working on it rarely offer any support at all for modding.