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 job 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 Bullseye job mod. The Bullseye mod is a completely new job for the game, and serves as an example of how to make your own. 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".

Down to Brass Tacks

The Job must have a simple code, preferably 2 capital letters. This will be the property name for the job information. Your JSON format should look like `{ "jobs": { "CO": { -actual mod data-} } }` As usual, you can include multiple jobs into one mod, by listing more than one code & data. Similarly, you can also include your images in the same mod file. It is recommended that you add your images to the file after testing your mod because editing large text files can be a pain. If you want to set up your mod with images, the JSON format will look something like this: `{ "jobs": { "BS": { job data } }, "images" { "modIMG-BSwallpaper": base64 data, etc. } }` This goes for all mods in the game, any combination of data can be combined into a single file! While there is no order required for the mod to work, it is courteous to put any images at the end of the file, and smaller mod items toward the top.

Basic Data

employer: [string] the proper name of the business/location employing the player.

code: [string] the same exact code used as the property name for this data, such as "CO" or "BE"

skills: [string] text description of the main skills needed for this job.

desc: [string] text describing the job itself, and what kind of work it involves

img: [string] the image name for the job listing image. The dimensions for the image 400 by 250 pixels. To the right is an example of this image, which gives an idea of the type of work involved, the job title, and the employer logo. You don't have to strictly follow this format, but your mod will look more integrated with AW if you do.

title: [string] the basic job title for the job series. "What do you do?" "Oh, I'm a [job title] at the Institute."

apply: [array: [string, number, string, number]] these are two skill checks used to determine the starting rand of the player at rank 0, 1, or 2. The skills used should be the primary skills for the job, at a difficulty that is similar to the difficulty of standard job tasks. The format is [skill code, DC, skill code, DC] skill codes are listed in other documentation, but the main job skills are CM = Communication, OG = Organization, PS = Problem Solving, FI = Finance, CL = Cleaning, PS = Prostitution (SX, OR, EX, SD, AS, EX, DA, SP, CO)

pcJob: [string] lowercase word that identifies the job. Ex: "services", "maid", "sperm", "bullseye"

wallpaper: [string] the image name of the wallpaper background for the job screen. The Institute uses a concrete wall, while the Bullseye job in the example mod uses the Bullseye employee breakroom. (The Bullseye mod wallpaper happens to include an employee in uniform...) The image resolution is 1920x1080 pixels. Higher resolutions are okay, but lower ones may appear pixelated when they are stretched to fit the screen.

jobPercept: [number 1 to 5] how socially respectable the job is. Most jobs should not exceed 3 to start, which is a generally favorable impression about the players employment. (2 is average, not good or bad. 4 is prestigious like a doctor or upper management position, and 5 is essentially being the CEO of an international corporation...)

Working Time!

schedWorkTime: [array[]] this is the weekly schedule of work times, including lunch and other breaks. The format is `[ hrsPerWeek, timesArray, timesArray, timesArray, timesArray, timesArray, timesArray, timesArray]` hours per week is a simple number, such as 40 for 40 hours per week. The "timesArray" listed above use this format: `[ startHour, startMinute, endHour, endMinute]` Remember to include break times into the duration of time 'at work'. Minutes can be values 0 to 59, but please stick to reasonable values such as 0, 15, 30, or 45. The hour is in 24 hour format. A day off is recorded as `[0, 0, 0, 0]` The order of the timeArrays indicates the day of the week, starting with Monday and ending with Sunday. Note: all shifts must complete during the calendar day. You can't start a shift at say 8pm and end it at 5am. The value of the end time must be larger than the start time!

schedWorkDays: [array] this is a quick set of Boolean values that show which day the player has work. The format is `[workHrsPerShift, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean]` the workHrsPerShift is a simple number that indicates how many work hours occurs per day, not including breaks. The Booleans are either true or false, true meaning that the player has work that day. The Boolean values start with Monday and end with Sunday.

rulesWorktime: [array of numbers] the format for this is [weeklyHrs, hrs, hrs, hrs, hrs, hrs, hrs, hrs] where weeklyHrs is the total number of hours per week, and hrs is the number of hours worked on that specific day. As usual, the first 'hrs' refers to Monday, and the last is Sunday.

rulesBreaktime: [number] the number of minutes per shift allotted to breaks land lunch.

Job Tasks

Job tasks are the 'work' performed by the player during a normal shift at work. They consist of several task objects containing information about them, as well as some basic settings/information. The tasks themselves have property names like "rulesTaskX" where X is a capital letter from A to K. You should not skip a letter, such as using A, B, C, E, F for tasks, but they do not need to be in a particular order in the mod file. The arrangement of data is a little awkward, so you can blame Besty.

rulesTaskRatio: [array] this is the ratio that job tasks should occur. There should be a number for each "rulesTask" object (covered below) in the job. These values determine the relative frequency of each task when tasks are performed at work. These values do not have to add up to a specific amount. Example: `[16, 8, 4, 2]` in this array, task A is 8 times more likely to occur than task D, and two times more likely to occur than task B. You can also interpret it as a probability for each task. Because the sum of all the values is 30, task D has a 2 in 30 (1 in 15) chance of being selected when a task is chosen. Generally, harder or more unique tasks should occur less frequently than simple ones.

rulesTasks: [number] this is the number of tasks in the job, and MUST match the total number of tasks present. Each job must have a minimum of 4 tasks. (A, B, C, and D)

rulesStrict: [Boolean] This should normally be true, but indicates how strict the job is about completing tasks and following the rules. Using false will result in the employer being slightly more forgiving of tardiness, and less upset about occasional poor performance.

rulesTaskX: [object] Each task object contains 7 properties, type, DC, retry, effect, stress, hap, and desc. These are detailed immediately below.

type: [string] this is the type of skill check the task requires. (see the skill check codes above) ex: "CL"

DC: [number] this is the difficulty of completing the task, and is the skill check DC. 10 is quite easy, 13 is moderate, 16 is difficult, and 19 is very difficult. For job tasks, we want some more common tasks to be easier, with less common tasks to have more difficult skill checks.

retry: [Boolean] this determines if the task can be retried if the initial attempt (skill check) fails. Some tasks naturally can only be attempted once, you either succeed or not. Others should be fine to make a second attempt. The second attempt allows the player to salvage their performance score.

effect: [number 1 to 5] how important the task is, or how much of an effect it has when considering the player's performance. If you create a certain critical task that would have major consequences for failure, it should have a higher effect value. This value affects the overall performance score, as higher values weight the average performance value to favor the task with higher effect. It also increases the penalties for failure to a certain degree.

stress: [number 3 to 20] this is how much stress performing the task adds to the player. The usual range should be from 5 to 10. Extremely unstressful tasks could be lower, while high stress tasks can be 15 or more. Please note that high stress tasks should be uncommon, or they can cause too much difficulty for the player. It's also worth noting that higher paying jobs should have more stressful tasks, and a higher frequency of high-stress tasks.

hap: [number -3 to 1] this is the effect of the task on the player's happiness. Most tasks can be 0, but unpleasant tasks can be -1. Lower values should be used with caution for the most unpleasant tasks, as they can add up and cause difficulty for the player to manage their mental health. A positive value 1 should only occur for a task that somehow magically makes the player more happy when performing it. So far there are none in the game.

desc: [array of strings] this is an array of strings labeling the "task". When a task is selected, a random entry from this list is chosen as the specific task attempted. Common tasks should have more entries here to avoid repetitiveness. This is merely for flavor, so having more than 1 description isn't mandatory, but it's highly encouraged. Task names here don't have to fit in the same "group", they just need to fit the skill involved (washing a car, and cleaning toilets could be in the same task, for example).

Events

Events are things that can occur while at work. It might be a simple event of only a few paragraphs, or could be a multi-event chain that spans over several work days. Your imagination is the main limit when it comes to events!' You aren't required to have any events at all for a job mod, though they do spice things up. You don't need to know JavaScript to create an event, everything can be done with twee... but it does help if you want to create more complicated conditions for the event to occur (this is still pretty easy).

Note that events are checked in index order to see if they occur, so you should place more important/rare events at the beginning of the array.

events: [array of event objects] the "events" array holds several comma-separated objects {} with properties that define the event. Those properties are explained below:

rank: [array of numbers] the rank property contains numbers associated with the job's ranks. It must contain at least one number. This determines what rank/s are eligible for the event to occur. If the player's current rank is not in this array, they will never see the event. Example: `[1, 3, 4]` in the example, the event would only occur for players of ranks 1, 3, and 4. Note that ranks 0 and 2 are excluded, meaning that

the event will not occur while the player is those ranks. Be cautious when limiting an event to only rank 0 or 1, because players may start in the job at rank 2.

image: [string] this is an optional image that will appear in a separate draggable box. This is more useful for showing portraits, or some static/environmental image, rather than a scene-based image. You can use "none" to not display an image.

random: [Boolean] This is whether or not the event should use randomization to create a probability of the event occurring. If this is true, the odds property is used to determine the odds of the event occurring. Using this, you can set the event to only have a (for example) 10% chance of occurring per day. Note that the condition property listed below is still checked to see if the event is eligible, so you can still restrict the event to certain conditions AND have only a certain chance of occuring.

odds: [array of 2 numbers] The odds property contains two numbers. The first is the threshold, and the second is the max. The randomizer works by generating a random number from 0 to the max. If the random number is less than the threshold, the event occurs. Example: [1, 9] in this example, the generated random number would be a value from 0 to 9. If the number is 0, it will be lower than the threshold, and the event will occur. Because there are 10 possible numbers for the RNG to generate (9 + 1 for the zero), and only 1 value that will trigger, the example would have a 10% chance of occurring. If the example was [9, 9] instead, the event would have a 90% chance of occurring, because all values except nine are less than 9. If you want to have an event occur 100% of the time, please just use false for the random property value above. *Note: the threshold value must always be 1 or greater, or the event can never occur.*

title: [string] This is the title of the event. It will appear at the top of the window. If part of an event chain, it may be useful to include something like "[1 of 3]" or "[2 of ?]" in the title so that the player is aware, but this is up to you. The title itself can be anything that fits the situation of the event. You may also want to include your author name in the title as well, if you'd like credit for your work :D

condition: [string – JS function] the condition property is a JavaScript function in the form of a string. While the string format is required to fit JSON syntax, you can certainly write and test it normally using a standard code editor before turning it into a string. You cannot include line breaks in a JSON string, so this is also handy to write it in a more readable fashion before removing the linebreaks. The function will be supplied the current rank of the player, but all of the player and job variables will be in scope to use as well. If a value of true is returned by the function when checking for events each work day, the event can occur. A returned value of null or false will mean that the value can not occur. If you don't wish to get involved in the JS for your events, simply use the random feature, and copy and paste the "always true" function string "`(function(rank){return true;})`"

content: [string – twee] the content property contains—obviously—the content. This is in twee format for SugarCube. There are tons of guides on how to use twee available online, so I won't cover that here. You can find detailed information on the standard SugarCube macros at motoslave.net here. There are also many AW macros that *should* at least be mentioned in another document. A particularly useful one for job events is `<<intreplace>>new content to show<</intreplace>>` it clears the event window of its current content, and replaces it with the content inside the macro tags. When substitute passage adding is enabled, you will also be able to use a more convenient `<<intgo "passageTitle">>` Finally, you can use image markup to display scene-based images as well in the content, just as you would in normal twee markup. `[img[imageID]]`

I recommend writing your content in a more normal editor, either a code editor like Visual Studio Code, or something like MS Word or Google Docs. (if using a word processing program, be careful of smart quotes!) When you're done writing and proofreading your content, you can remove the line breaks so that it can be included in a JSON string. There will be an AW tool available for this, or you can do it manually. Also be cautious of using " quotation marks inside your content without an escape character, as this will cause an error in the JSON formatting (the AW tool will also check for quotation marks ;).

Job Content

"jobContent" is an array containing content objects. Unlike events, content objects are the more everyday things that happen at work, and are generally non-interactive (though minor interaction is possible). The example Bullseye mod has several example content items so that you can get an idea of what I'm talking about. I recommend having several content items so that the player doesn't see the same one or two over and over.

jobContent: [array of content objects] as with events, jobContent is an array containing objects {}.

rank: [array of numbers] just like rank described above for events, this is a list of rank numbers for which the content is eligible. You can use this to change which events show as the player advances in their career.

image: [string] the id for an image displayed above the content text. You can use an image tag in the content property if you'd rather have the image display below, and leave this as "none"

content: [string – twee] the text describing the event, in normal twee markup (please see the events content property above for more info). This twee can use normal macros, however the <<intreplace>> macro mentioned above for events won't work here because it is not in an interact window.

Job Ranks

This is your 'rank' at your job. This isn't like the military or police where there are literal ranks; rather it is just a term to describe the job progression. The primary jobs in the game will have 10 ranks, ranging from entry level at the bottom, to some form of middle management at the top. For now job mods should limit themselves to 5 ranks, because there will actually be a transition point for rank 5 that hasn't been coded yet. Additionally, you can't leave a rank number empty if there is a rank above it (ranks can't go 0, 1, 3, 4, etc.). An empty rank will break the game when the player is promoted to that rank.

ranks: [number] the number of the highest rank available in the job NOT the total number of ranks. (blame Besty) so if you have ranks 0, 1, 2, and 3, your ranks property should be 3.

rankX: [rank object] Each rank has a set of properties which are explained below. The property names for the rank objects are rankX where X is the number of the rank. (example: rank0, rank1, rank2, rank3, etc. Also blame Besty for this.)

rulesCutoffs: [array of numbers] this array contains a set of 5 numbers that determine the expected performance from the player. These numbers are not connected to individual tasks, rather they are a set of cutoffs for how the final performance value at work rates. Basically, several tasks are attempted, resulting in a final performance score of 0 to 100. These cutoffs determine which category the player falls in with a given performance, and thus what affect it has on the player. All the values are "less than" checks.

Example: `[20, 35, 50, 65, 80]` In this example, less than 20 is horrible, less than 35 is bad, less than 50 is poor, less than 65 is acceptable, less than 80 is above average, and 80 or more is good.

vacationRate: [number] The number of hours of unpaid vacation earned each month.

vacationRatePaid: [number] the number of hours of paid vacation earned each month.

sickRate: [number] the number of hours of paid sick leave the player earns each month..

pay: [array of 1 number] this is how much the player is paid per hour. Decimal values are allowed. Because 1 credit is the lowest denomination in the game, any fractional remainder for the week's pay is rounded. A pay of 7.5 would amount to 225 for 30 hours, not 40. Important: this number must be inside an array! Ex: `[8.5]`

statsRank: [number] Blame Besty some more, because this is the rank number of the current rank object. If you are defining "rank2" then this value should be 2.

name: [string] This is the job title of this rank. This should be capitalized, and should form a logical progression with the other ranks. Example titles: "Trainee Assistant Janitor", "Probationary Assistant Janitor", "Assistant Janitor", "Junior Janitor", "Janitor", "Senior Janitor" etc. etc.

rulesBoss: [string] this is the name of your boss at this rank.

promotionBonus: [number] Nobody knows what this does... actually, it doesn't do anything yet. Eventually it will represent a bonus check the player receives as an award for being promoted, obviously at higher ranks only, because janitors and corporate drones don't get bonuses.

Other

This is an optional element, but can be useful.

acceptance: [array of 0 to 3 strings] these are custom messages shown to the player after they apply to your job. The order of the array corresponds to the rank the player was hired at, with the first item being rank 0, and the last being rank 2. They can be written with twee markup, but that is probably overkill.

Don't forget to add images to your mod! Images are fun. Here are some of the images included with the Bullseye job mod: