

Ability System Script Tutorial

This tutorial explains how to implement and use the ability system script made by Eugene222 on Tsukihime's Shop Manager Script.

This script is extremely versatile and as such it requires an user of at least medium level experience, who knows how to handle the RM database and eventing. It can't be used as "plug'n play".



Overview of script capabilities

- Actors can gain ability points (AP) in several ways, to train (purchase) different abilities at different trainers (shops)
- An ability can be anything that can be defined in a common event, including skill learning, parameter bonuses and activation of game mechanics but also (in combination with other scripts) dynamic features, map changes or whatever else can be initiated by a common event.
- Abilities can be conditioned to other abilities and to classes, allowing the creation of ability trees where different actors learn different abilities.
- Abilities can be blocked by other abilities, forcing the player to choose between options.
- Trainers can be created with the shop processing on maps, but there is also a general training scene for basic abilities that can be learned always and anywhere.
- Trainers can be conditioned to have some abilities only available for training as the story progresses or as quests are solved.

1) Script installation

- a) Go to Tsukihime's blog, get the shop manager script (<http://www.himeworks.com/2013/02/22/shop-manager/>) and install it below Materials but above Main in the script editor.
- b) Get the ability system script from (http://www.avarion.de/download/skripte/ability/ability_engl.txt, English version) or (http://www.avarion.de/download/skripte/ability/ability_deut.txt, German version) and install it below the shop manager script in the script editor.
- c) Configure and reserve the variables for the ability system
 - i) Open a new event on the map and doubleclick in the content area
 - ii) Select "control variables", enter the variable area and choose two currently unused variables by naming them "Actor learning" and "item used". Confirm the naming but cancel the event.
 - iii) Search the lines ACTOR_VARIABLE = 1 and ITEM_VARIABLE = 2 in the ability system script and change the numbers 1 and 2 to the numbers of the variables you've reserved in the previous step.

2) Script configuration

- a) How do the actors gain AP?
You now have to choose how the actors should gain APs. There are three basic possibilities here: By leveling (gaining EXP), by battling (killing enemies) and by eventing (story progress). Eventing is always possible in addition to all other options, but you have to decide between leveling or battling or none of both, because that needs to be set in the script and can't be changed in game.

If the Actors should gain AP by killing enemies, search the line GET_MODE = in the script and set it to

GET_MODE = :kill

In this case, you have an additional choice on whether the AP gained by killing should depend on the actor or on the enemy. A few lines below the GET_MODE-Line is another line with KILL_MODE =.

If you set this line to :actor, then once per battle the actor will gain the AP-Number set for the actor (or the default). It does not matter how many or which enemies were killed in that battle – and different actors can gain different amounts of AP depending on their notetags.

If you set this line to :enemy, then after the battle all actors get the same amount of AP, and that is calculated by adding the AP-Numbers from all enemies killed in this battle. This number can vary by notetag in the enemy, or it will use the default value.

If the actor should gain AP by leveling, then you need to set the GET_MODE to
GET_MODE = :level

In this case the notetags of the actor will determine how many and how often the actor gains AP – this will be configured later as it's outside the script.

If the actor should get AP only by eventing (neither leveling nor battling), also set the GET_MODE to :level – the notetags will take care of the rest.

- b) How many AP's should everything cost?
That is a balancing question and the effective costs can be changed later, but the line DEFAULT_POINTS = determines how many points are gained if no notetags are used. You should set this to a number that is what you believe to be the average of the costs.
- c) Should it be possible to unlearn abilities?
If you are in doubt, say NO...
The ability system allows for unlearning of abilities, but in this case each ability needs two common events (instead of one) and the setup is much more complex.
It will be discussed later – for now only remember that to allow for unlearning requires a lot more configuration in the script itself.
- d) Do you wish to change the messages?
If you want to translate the vocabs or change them to better reflect your game world, this is possible in the Vocab area of the script. If this is your first use of the script, keep the default Vocab until you've configured the rest.

Now that the script is installed and the basic configuration is ready, what to do next?

Answer: Shutdown the computer, take a notebook and a pen and start planning...

No, really – if you're using the ability system script instead of a regular skill shop or common event shop script, then most probably you want to create advanced skill/ability trees and give the player a lot of choices.

That means we're talking about several dozen abilities at minimum, and each ability requires an item and a common event to store all needed data. To get this right without later problems requires a bit of planning ahead, and we still have to make several assumptions to reduce the amount of work to check for errors.

The first assumption is that all ability items will only be handled through the training scenes provided. Theoretically it is possible to make the ability items real ingame-items, but that would require the common events to manually check all conditions. A later chapter of the tutorial will show how that is done, but for most parts I'll assume that the script handles the conditions inside the training scenes.

The second assumption is that an ability cannot be unlearned after it has been learned, and that an actor who joined the party will stay in the party and never be removed (which means that party abilities won't be removed either). Both forms of ability removal are possible, but they require advanced handling and I do not want those descriptions cluttered into this basic tutorial.

The third assumption is easy to achieve with a bit of planning, and it will also make your life as a game developer much easier:

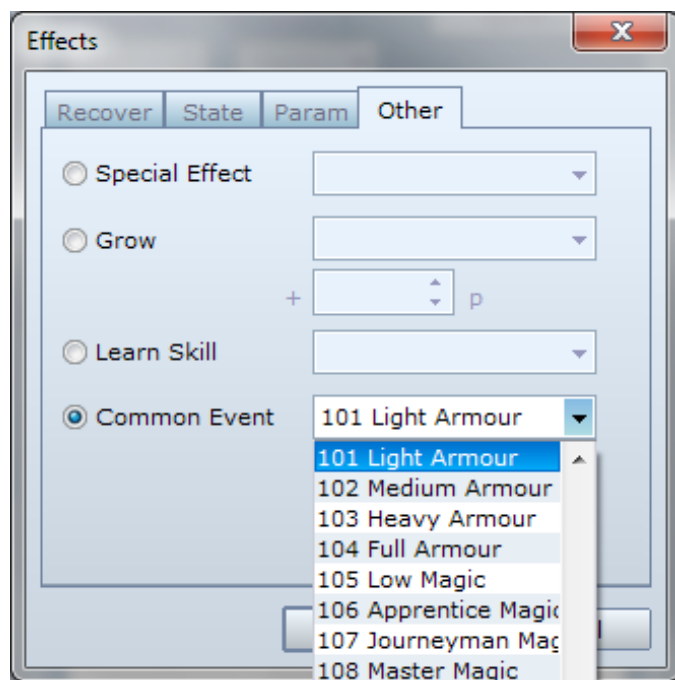
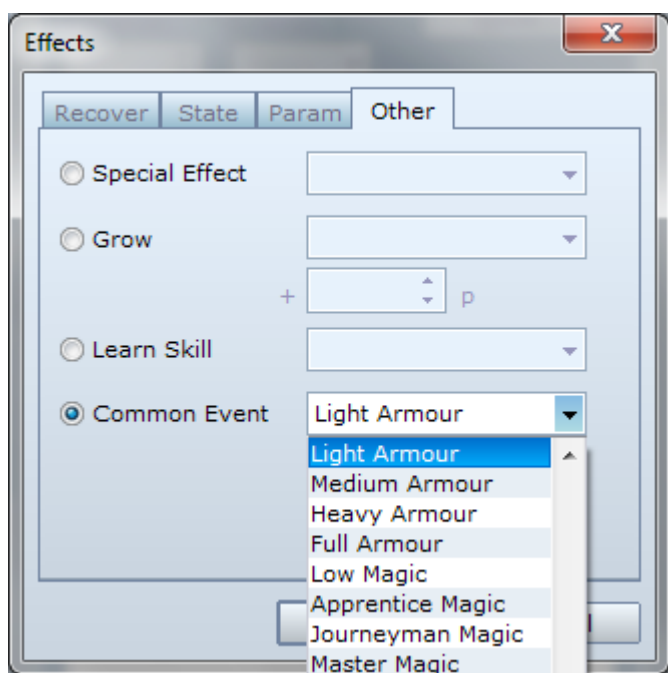
Each Ability needs both an item in the database and a common event to handle all data, and in all cases we'll assume that the number of the common event and the ID of the item are identical, and use that number as the ID of the ability.

This assumption can easily be achieved by starting the abilities with higher numbers like 100 – just increase the maximum on both items and common events to 500 or so, and start putting the data for the first ability into item 100 and common event 100. If you think you need more than 100 general items (or more than 100 common events for other functions), start with 200 instead.

If all planning fails and you later need to add abilities when you no longer have the reserved slot numbers, then make sure that the real item ID is placed inside the common event name, as the script focuses on the item-ID, and only the item name (not the common event name) is displayed ingame.

This (placing the item/event-ID in the CE name) might be a good idea anyway, because the effect for calling the common event doesn't list the ID, only the name.

The picture to the left shows the regular naming of the common events, the picture to the right shows how it looks if you add the ID to the name manually



As an example, let's start with two simple ability trees for armor and for magic – and that the ability for using stronger armors prevents the learning of higher magic (and vice-versa).

You'll need a dynamic feature script in addition to allow your actors to gain features like armor types later in the game, but let's assume that you already have that and continue to put the conditional data for the ability trees into the notetags only...

Let's say there are the abilities

100	No Armor	105	Low Magic
101	Light Armor	106	Apprentice Magic
102	Medium Armor	107	Journeyman Magic
103	Heavy Armor	108	Master Magic
104	Full Armor	109	High Magic

High Magic could only be cast without real armor, Master Magic only with light armor, journeyman magic only with medium armor, apprentice magic would tolerate heavy armor and low magic would be possible even in full armor.

Each of the Abilities would require the lower one to be able to be learned.

This is the minimum you should have as a result of our planning: a complete list of abilities with their IDs (and hopefully a summary of what they do), before you enter them into the project.

I'll stop the example here – the list for a real game would be longer, but this is enough to see how the abilities are set up.

Now let's check what one of the ability items contains as data, and how this is entered in the database:

Database

Actors | Classes | Skills | **Items** | Weapons | Armors | Enemies | Troops | States | Animations | Tilesets | Common Events | System | Terms

Items

992: | 993: | 994: | 995: | 996: | 997: | 998: | 999: | 100: No Armour | 101: Light Armour | 102: Medium Armour | **103: Heavy Armour** | 104: Full Armour | 105: Low Magic | 106: Apprentice Magic | 107: Journeyman Magic | 108: Master Magic | 109: High Magic | 110: Low spells | 111: Fire Apprentice | 112: Fire Journeyman | 113: Fire Master | 114: High Fire | 115: Ice Apprentice | 116: Ice Journeyman | 117: Ice Master | 118: High Ice | 119: Fire&Ice | 120: Rogue | 121: Lockpicking | 122: Lifeforce | 123: Improved Lifeforce | 124: | 125: | 126: |

General Settings

Name: Full Armour | Icon: [Icon] | Description: Full Plate is the best protection, but blocks everything other than low magics | Item Type: Normal | Price: 3 | Consume: Yes | Scope: One Ally | Occasion: Always

Invocation

Speed: 0 | Success %: 100 | Repeats: 1 | TP Gain: 0 | Hit Type: Certain Hit | Animation: None

Damage

Type: None | Element: | Formula: | Variance: | Critical: | Quick...

Effects

Kind: | Content: Common Event [104 Full Armour]

Note

<abil_req: 103>
<abil_blk: 106>
<class_req: 1>

Annotations:

- Name, Icon and Description will be displayed on the training and viewing scenes for abilities. The description should hint at requirements or blocking abilities.
- The price is the number of Ability Points (AP) removed from the actor when learning this ability in a training shop.
- The item ID is the key that is used to identify an ability in the notetags and script commands
- This common event will be executed to give the chosen actor this ability, like adding parameters or skills
- Notetags are used to list required or forbidden (blocking) abilities, and can set a few other things as well.

OK | Cancel | Apply

Most of these points should be clear, so let's have a look at the notetags. Three of the four possible notetags are used in this ability.

<abil_req: 103> is the notetag you'll probably use most of the time. It tells the script that an actor has to have learned the Ability 103 (Heavy Armour) before this ability can be learned.

Alternatively, I could have used a notetag like <abil_req: 100, 101, 102, 103>. That would have told the script that all forms of armour need to be learned before this ability could be learned, but there are three reasons why I made only one requirement:

- 1) In the selected structure, all armour abilities require the next lower ability to be trained – placing them in a higher ability again would be redundant.
- 2) The training scene can only display four requirements due to space limitations. Listing redundant requirements might result in the player unable to see the later requirementst.
- 3) If only the direct requirements are set, some actors might be set to bypass this by script calls (see the third actor in the demo for an example)

If several requirement abilities are listed, all have to be learned to be allowed to learn this ability.

The second notetag is <abil_blk: 106>. This notetag tells the script that any actor who has already learned the ability 106 (Apprentice Magic) will not be able to learn this ability (Full Armor).

If several abilities are listed as blocking, then any of those abilities block the training if they are learned.

The last notetag is <class_req: 1> and this results in only actors with that class being able to train this ability (single class only, the script won't handle multiple class requirements).

You can see other examples for ability items in the demo, some of them are explained later in this tutorial.

There is one more item notetag for the script: <general_ability>. In the demo this notetag is used only in the three abilities "No Armour" (100), "Lifeforce" (122) and "Improved Lifeforce" (123).

This notetag has nothing to do with ability requirements. Instead, it flags the ability to be available in the menu training scene. This is a training scene in addition to the regular evented training scenes, and it will offer only the abilities with the notetag <general_ability> for training.

You as the game developer can decide if you want to use evented trainers on the maps for training, or if you want to train only on the menu, or combine both. Evented trainers have more options, but don't fit into every game or story.

Actor Notetags

As said in the script configuration part, you can set everything up with defaults and there would be no notetags on the actors needed. However, if you want your actors to gain AP in different amounts, then you can set that up by notetags.

Let's start with the notetags used when the script configuration is set to GET_MODE = :level, that means the actors will gain ability points by leveling.

Your first decision as the developer has to be when (at which level) do the actors gain AP? It could be that the game is based on many abilities and they gain the points for several abilities each level, or the abilities could be something rare and they would gain only one or two ability point every other level.

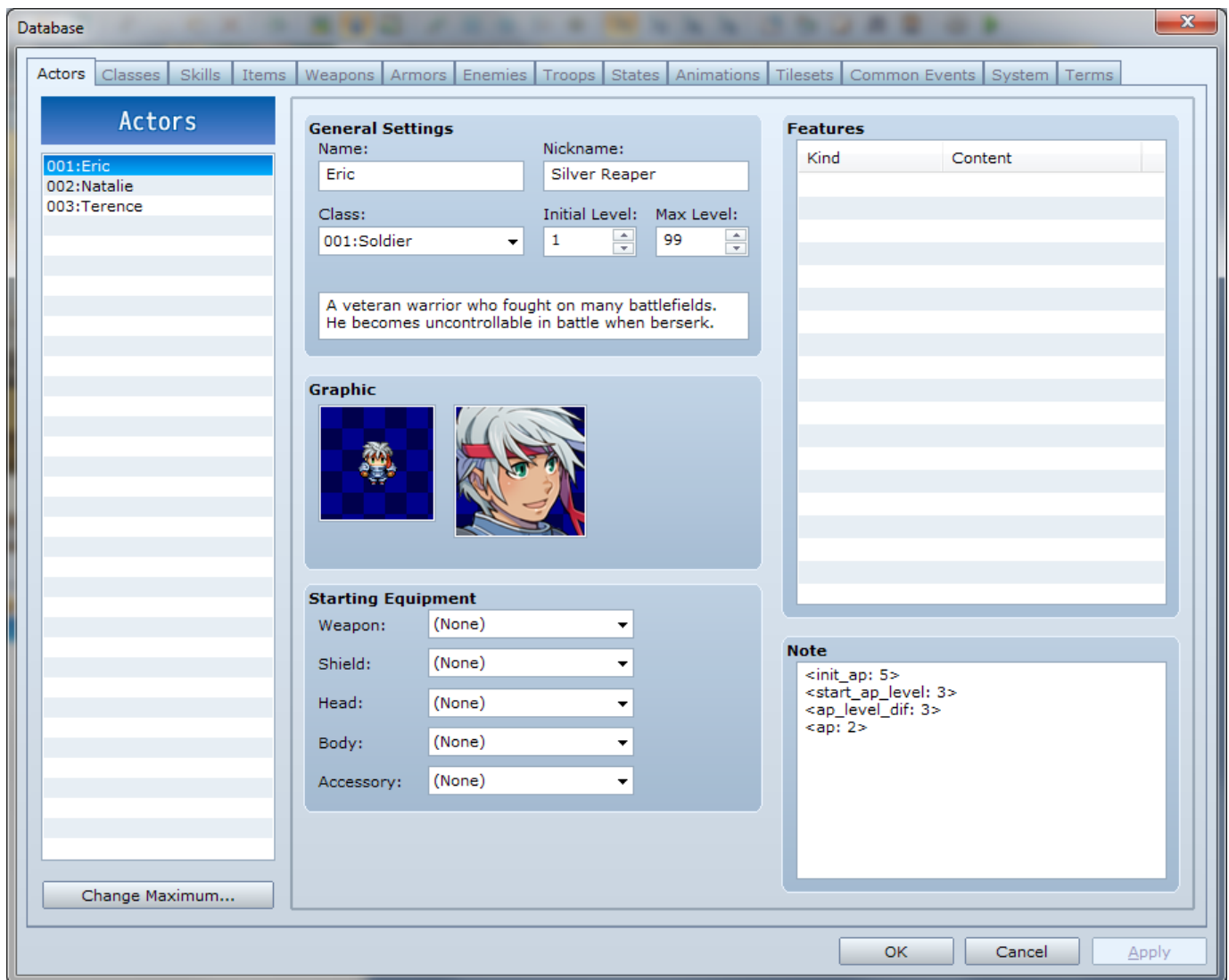
There are two notetags controlling on which levels an actor would gain AP:

<start_ap_level: level> will set the first level when the actor will gain AP. The basic default is 2, which means that the actor will start gaining AP when he hits his first level advancement to level 2.

Note: if you want to disable automatic AP-Gains, set the Starting level for APs to a number higher than the maximum level in the database, for example 100.

<ap_level_dif: number> will tell the script, how many levels should pass between gaining AP. The basic default is 1, which means that the actor gains AP at every level.

If you increase those two numbers, ability points become rarer for the player. Let's see an example from the demo:



With these settings, Eric will gain AP at levels 3, 6, 9, 12, 15 and so on – but what are the other two notetags in this example?

<init_ap: number> will tell the program that Eric will start with some AP (in the demo it's 5 AP) when the actor is added to the party. This is a good way to provide the player with a few AP to start learning abilities.

<ap: number> will tell the program how many AP the actor should get whenever AP's are distributed – in the demo Eric will gain 2 AP every three levels.

Now for another example: What will happen when the script configuration says GET_MODE = :kill?

In this case, the first two notetags for leveling become useless and will be ignored.

<init_ap: ##> will still give the actor some initial points to spend, but all future points will be gained only by winning battles.

However, the :kill – mode has two options on how the AP are gained after battles. This is also set in the script by the KILL_MODE , that can either be :enemy or :actor.

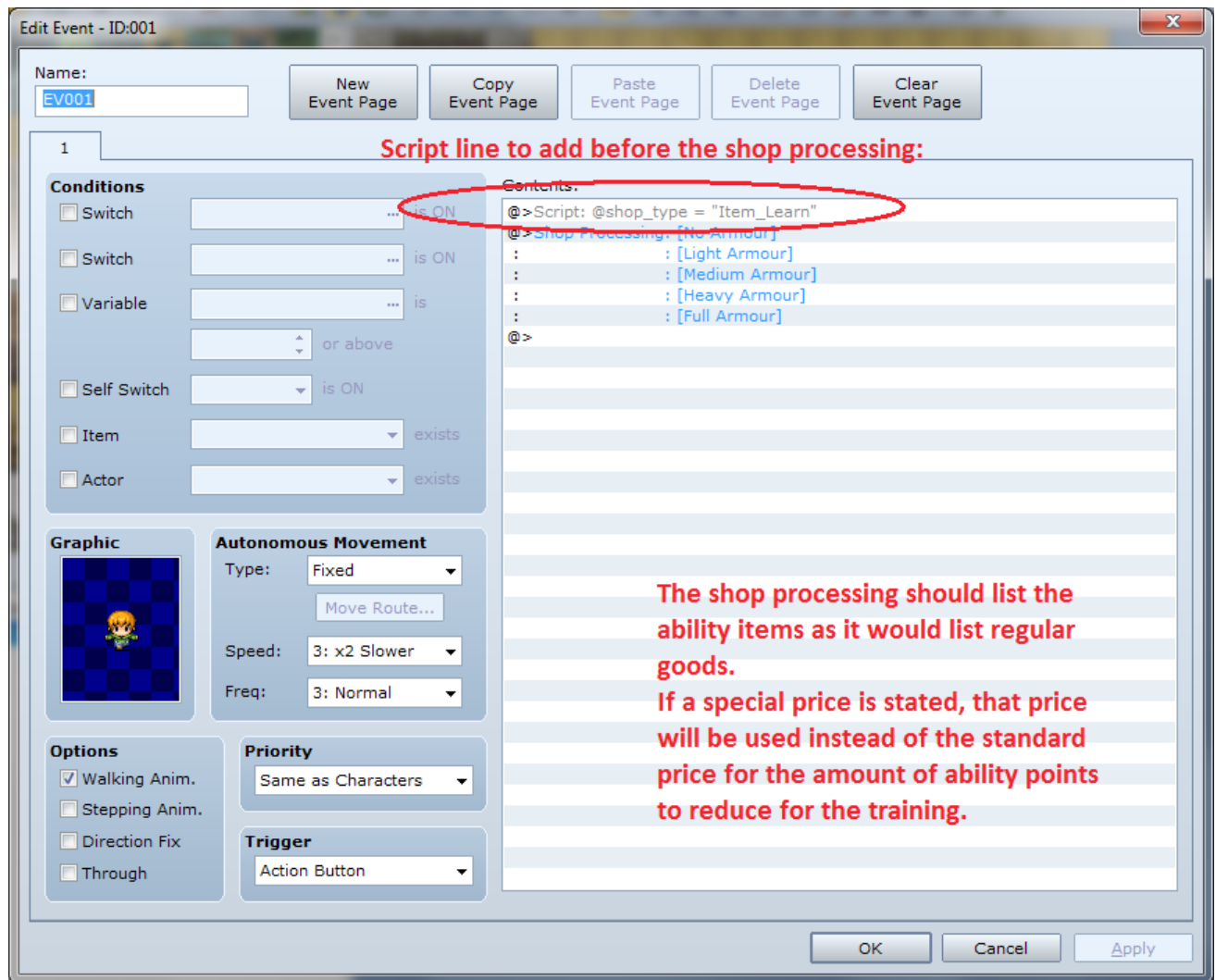
In the case of :enemy, the <ap: ##> tag needs to be placed on the enemy notetags. After each battle, all actors from the battle will gain as many AP as the sum of the AP-Tags of all enemies. This is similar to the distribution of EXP after battle – and you can decide if only boss enemies should give a few AP, or if every enemy gives dozens of AP.

In the case of :actor, the <ap: ##> tag needs to be placed on the actor notetags (similar to the leveling case), and each actor gets its own amount of AP once after each battle, no matter how many enemies were in that battle.

In both cases, a missing notetag causes the default setting to be used – the script's setting DEFAULT_POINTS = 10 is directly below the setting for the KILL_MODE.

How to create an ability shop (Trainer Event)

The basic ability trainer is a very simple event:



As you can see, the event only contains a script line

@shop_type = "Item_Learn"

Followed by a regular shop processing command with some Ability Items as the goods.

That is the power of Tsukihime's Shop Manager, but that script can do a lot more – so let's look at a more complex ability trainer:

Edit Event - ID:008

Name:

1

Conditions

☐ Switch is ON

☐ Switch is ON


☐ Variable is or above

☐ Self Switch is ON

☐ Item exists

☐ Actor exists

Graphic



Autonomous Movement

Type:

Speed:

Freq:

Options

☒ Walking Anim.

☐ Stepping Anim.

☐ Direction Fix

☐ Through

Priority

Trigger

Contents:

```

@>Text: ~, -, Normal, Bottom
:      : Look what happens with my offers when you
:      : change the lever...
@>Script: hide_good(3, "!s[4]") Hide Ability #3 when switch 4 is OFF
:      : hide_good(5, "!s[4]") Hide Ability #5 when switch 4 is OFF
:      : hide_good(6, "!s[4]") Hide Ability #6 when switch 4 is OFF
:      : @shop_type = "Item_Learn"
@>Shop Processing: [Low spells] Ability #1
:      : [Fire Apprentice] Ability #2
:      : [Fire Journeyman] Ability #3
:      : [Ice Apprentice] Ability #4
:      : [Ice Journeyman] Ability #5
:      : [Fire&Ice] Ability #6
@>

```

OK Cancel Apply

You'll see that there are three more lines in the script command before the shop processing, all following the same structure:

Hide_good(#, "!s[ID]")

Where # stands for the number of the item/ability in the following shop processing, and ID for the ID of a switch that activates or deactivates the ability in the training menu.

In the demo, the switch is turned by the lever mentioned – in a real game that switch is usually controlled by a completed quest, where the option to train new abilities is the reward for completing the quest.

However, there is one thing you have to keep in mind when using Tsukihime's hide-command: The command works as "Hide if true", but the game switches are OFF/false by default. And it's usually easier to remember something like "OFF => Ability can't be trained" and "ON => Ability can be trained, especially since most switches are used with quest solved = ON.

Because of this, the negation !() is added around the switch sequence s[ID] – if your trainers ever work in reverse of the intended switches, that means that you either forgot the negation in the hide-command or that you setup your quests in the way of solved = OFF.

I will not go into every other option available in Tsukihime's shop manager – look at her script description to learn the other possible commands or her formula-addons to the shop manager.

But by correct use of the formulas for hiding or disabling shop goods, you can have the ability trainers very versatile.

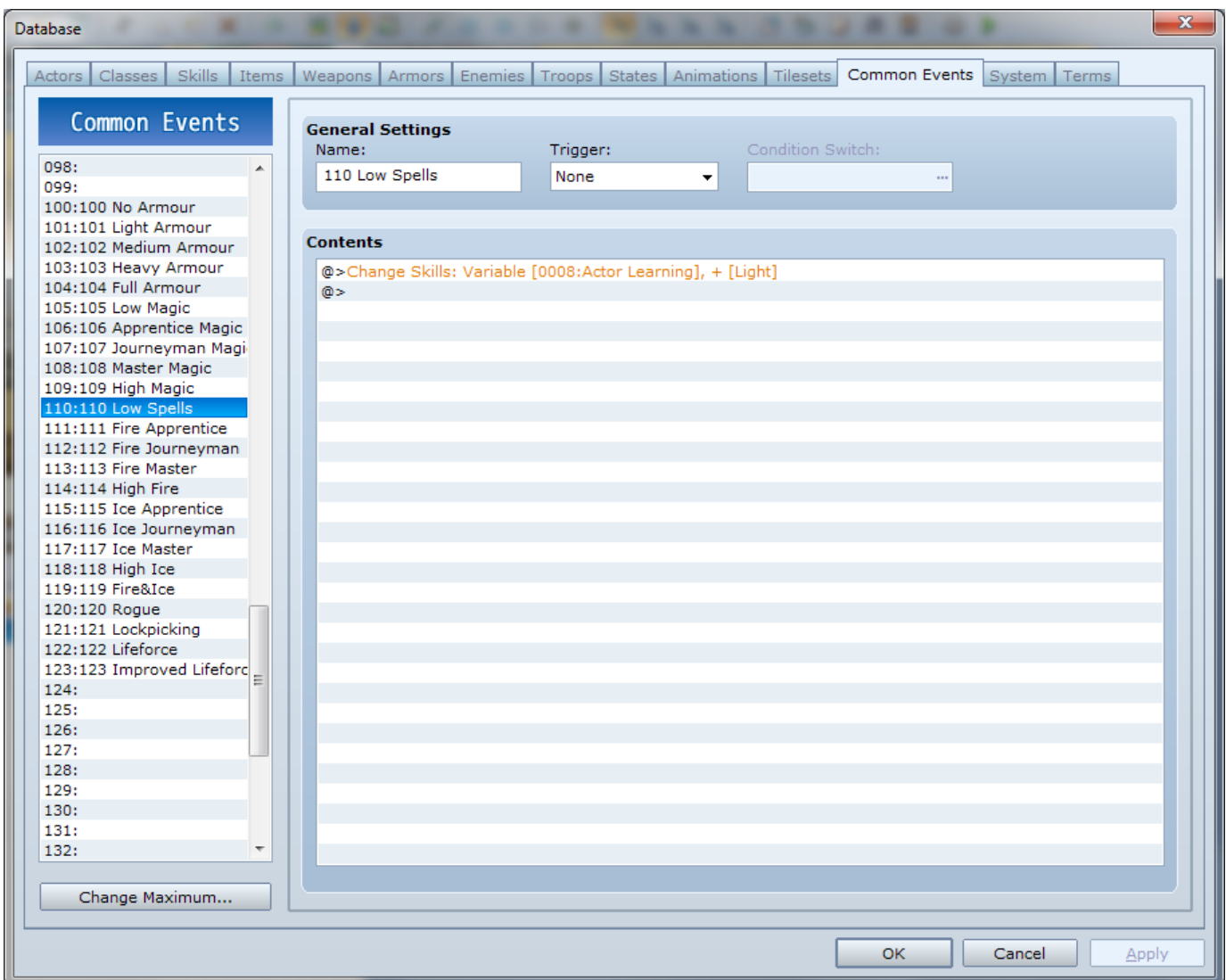
How to setup an ability: the common events

Now it comes to the interesting part of the tutorial: what can an ability be and how do you set the common event for any actor selected to learn this?

An ability can be a lot of different things:

- A skill can be learned
- A Parameter can be increased
- A script can be called to implement other features
- Pseudo-classes and Multi-classes can be implemented
- An event mechanic can be activated
- Anything that can be caused by a common event...

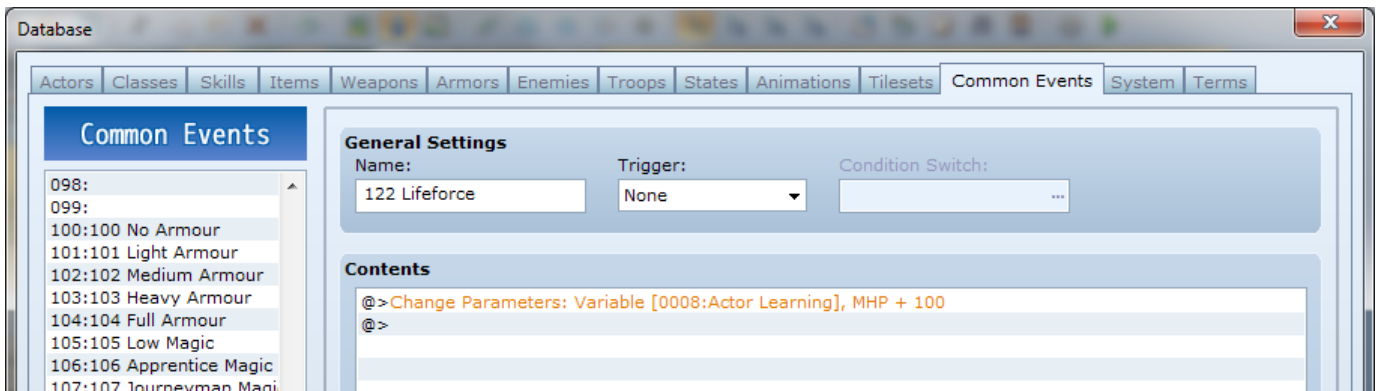
Let's see how this is done one after another, beginning with the very easy skill learning:



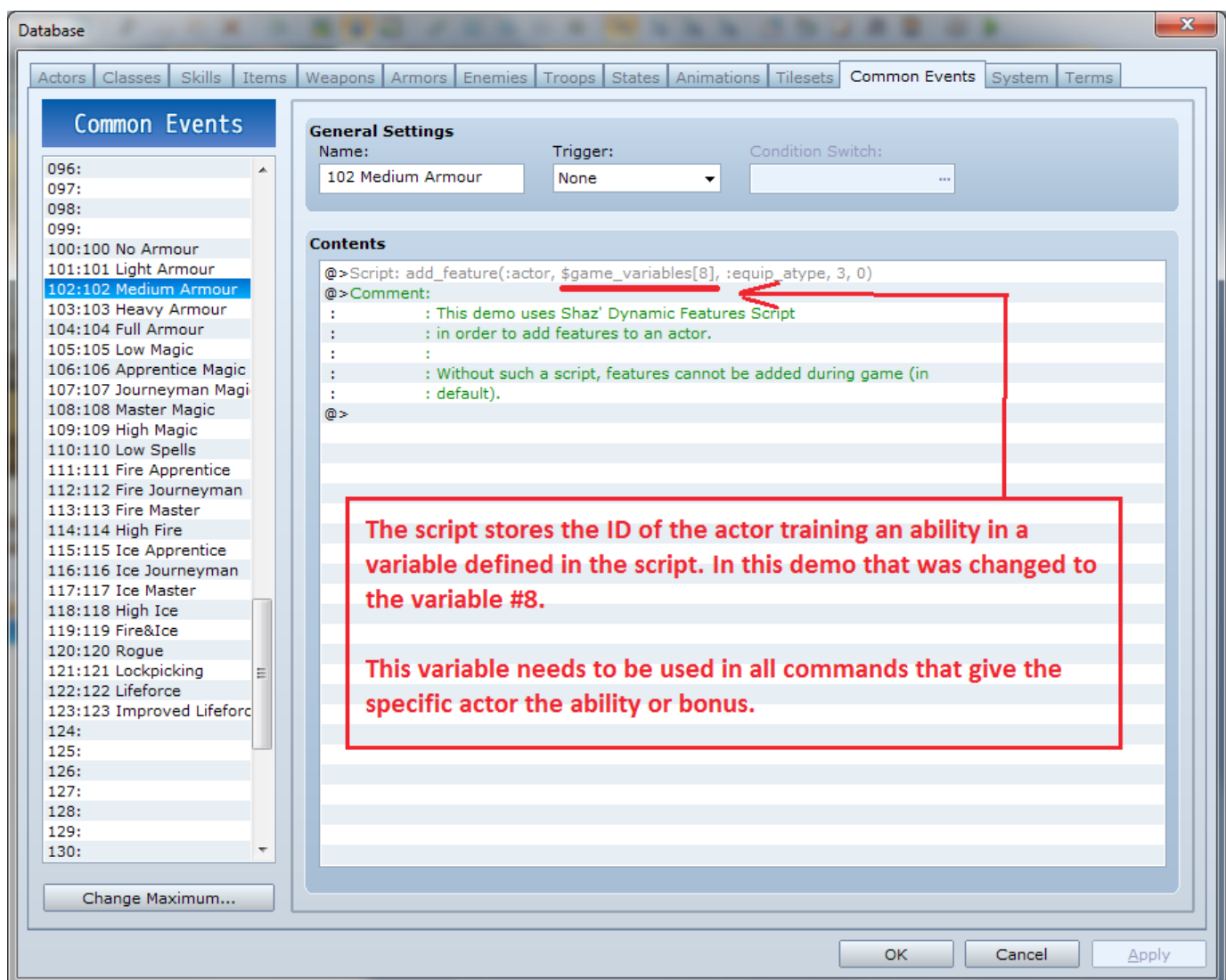
As you can see, this is a simple event command “Change Skill” set to a variable instead of a fixed actor learning that skill. And the variable used is the variable we reserved for the actor ID in the script’s settings. That is enough to make the event operate on the actor selected in the training scene, and give that actor the purchased skill.

That of course means that any other event command can also be used in the same way for training – including the often asked parameter changes on leveling:

As you can see, this ability simply adds 100 to the actor's maximum HP, and any player can now select if the ability points gained should first be used on learning skills or increasing points.



The next example shows how to use a script to implement other options. In this case, it's Shaz' "Dynamic Features" Script, and I use it to allow the actor to gain Features whenever he wants to pay them with Ability points:



Of course, other scripts might be used as well to change the settings for an actor in similar ways – most scripts should be compatible if they provide scriptcalls to add or

activate their functions. And somehow I suspect that a lot of users will start experimenting which other scripts can be used together with this options ;-)

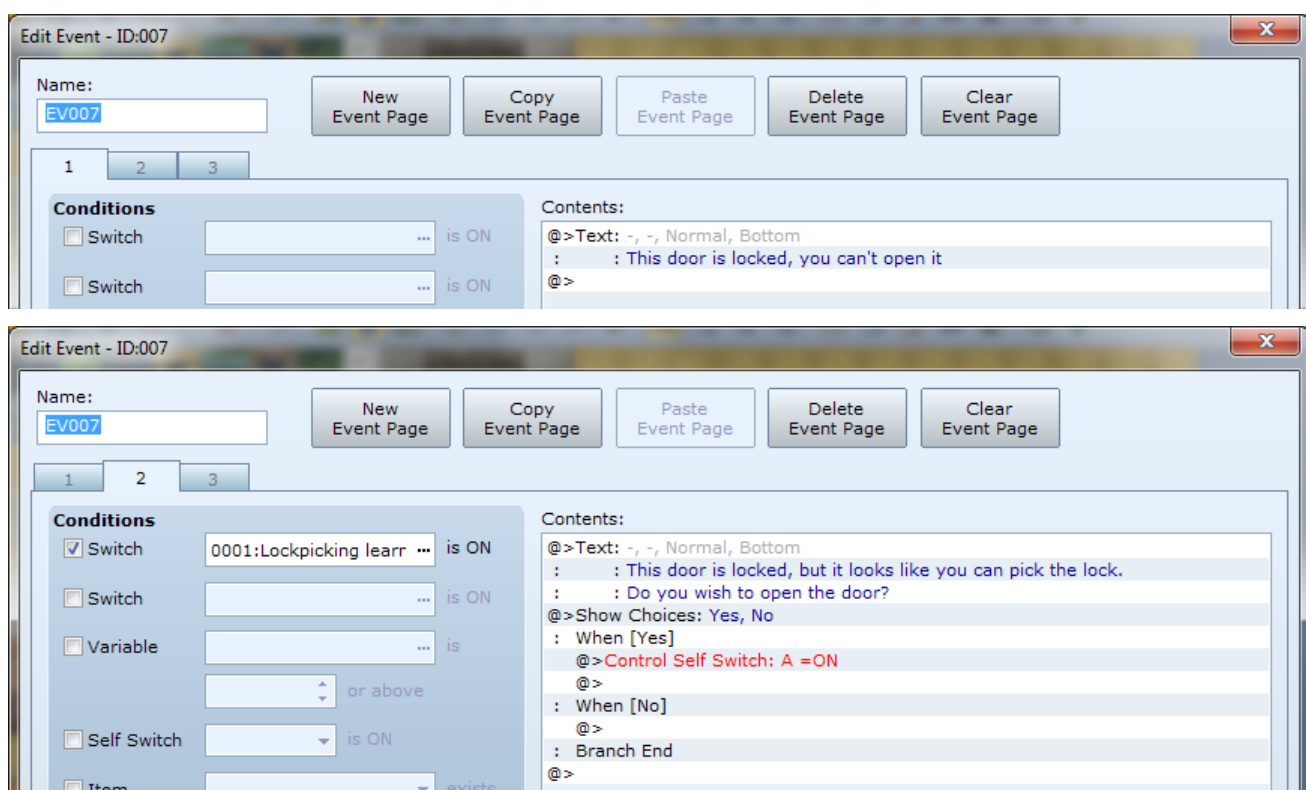
The next example is somewhat different. At the very beginning I said that all abilities should have an item and a common event of the same ID to make the system easier to handle. There is one exception to this: An ability that does not provide anything to the actor does not need a common event. Such an ability can only be used as a requirement, as a condition to allow other abilities to be learned or blocked. But Why should someone use such empty abilities?

Take a look at the autorun event in the demo bottom left – it not only gives the introductory text, but also adds one such “empty” ability named “Rogue” to the lead actor. In my planned main game, I will use the regular class engine to implement races (elf, dwarf, human, and more) into the game – and the classes (including the option to multiple classes) will be implemented by this ability system.

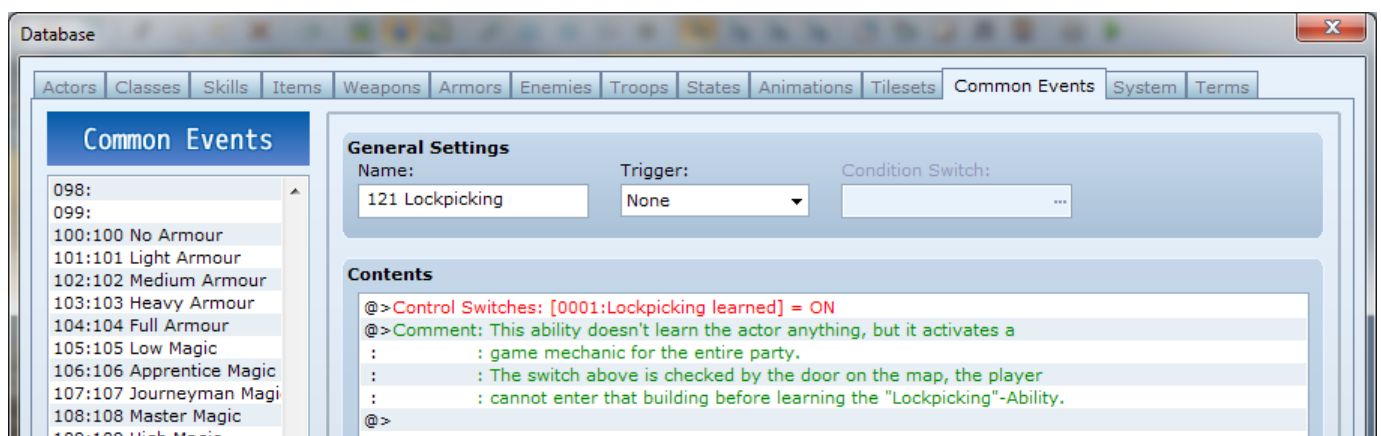
This Rogue-Ability will tell the script that the actor can learn all abilities allowed for the “class” rogue – even if that class does not exist in the official database class system. And as the result, only Eric can learn the lockpicking ability from the shady NPC bottom right in the demo.

I suggest that in your later games, you place all these “empty” abilities that do not need a common event to a different ID range than the regular abilities – in case you might need additional common events that should not be scattered through the rest of the list.

For the next example of an ability, let’s first have a look at the door event. The first two pages are interesting, the third is only the opened door:



As you can see on the second page, the door becomes “pickable” when the switch “lockpicking learned” is turned on. And how is that turned ON? Of course by learning the ability “Lockpicking”:



This is also an example why I said at the beginning that this tutorial assumes no ability can be unlearned and no actor can leave the party – you would need a way to check when the last actor with the ability “lockpicking” vanishes and then turn OFF the switch, or the maps would still allow lockpicking after all actors with that ability have been removed.

But a lot of other abilities can be implemented the same way, like allowing a laboratory event to be used for alchemy or crafting only after the ability has been gained or so on.

This concludes the examples for now – perhaps I’ll add more examples later, but this should be enough for a lot of people to make their own abilities for their games.

Credits:

Eugene222

For the Ability System Script

English Version: http://www.avarion.de/download/skripte/ability/ability_engl.txt

Deutsche Version: http://www.avarion.de/download/skripte/ability/ability_deut.txt

Tsukihime (<http://www.himeworks.com/>)

For the shop manager <http://www.himeworks.com/2013/02/22/shop-manager/>

Shaz (<http://ezmash.net/devblog/>)

For the Dynamic Feature Script added to the demo (<http://ezmash.net/devblog/?p=102>)

Andar (<http://forums.rpgmakerweb.com/index.php?/blog/92-avarion/>)

For the idea for this script, the demo, tutorial and translation

© Tutorial by Andar