Contents

Rules

A *rule* in arcadeEIP is the combination of a *function* and a *qualifier,* which together enable special processing that would typically require custom scripting.

Rules are specified in arcadeEIP.ini within an emulator section in this format:

```
rule_1=<function> arg_1 <qualifier> arg_2
```

More than one rule can be added to a single emulator section by indexing them.

```
rule_1=<change_root> [root_folder]_* <if_rom_in_subfolder> \roms
rule_2=<remove_param> -autosave <if_rom_in_list> paperboy,eyes,blitz,pong
rule_3=<use_proxy> [rom_name].txt <if_in_folder> D:\Emulators\MAME\proxies
```

Rules are processed in sequential order, so if more than one is used, make sure the order is appropriate. Use **`debug_mode`** to observe the effect of the rule prior to live use.

Select from several available built-in functions, or *build your own* external function module using the example script (**`fx_add_param.ahk`**) provided in the distribution.

Rule Dictionary

1. remove_param

Format:

`<remove_param>` *param{,param}…* **`<if_rom_in_list>`** *rom{,rom}…*
`<remove_param>` *param{,param}…* **`<if_rom_not_in_list>`** *rom{,rom}…*

Examples:

```
<remove_parm> -joy, -priority 1 <if_rom_in_list>targ,pong,tailg
<remove_parm> -joy, -priority 1 <if_rom_not_in_list>targ,pong,tailg
```

Function:

Remove one or more parameters from an emulator section's **`param_list`**.

arg_1:

A comma delimited list of parameters to remove.

Qualifiers:

`<if_rom_in_list>`	: remove parameters if the rom is in the **`arg_2`** list.
`<if_rom_not_in_list>`	: remove parameters if the rom is not in the **`arg_2`** list.

arg_2:

A comma delimited list of rom names.

---

2. append_param

Format:
<append_param> {*params*} [param_list] {*params*} **<if_rom_in_list>** rom[,rom]…
<append_param> {*params*} [param_list] {*params*} **<if_rom_not_in_list>** rom[,rom]…

Examples:
<remove_parm> -joy [param_list] -priority 1 <if_rom_in_list>targ,pong,tailg
<remove_parm> -joy -priority 1 [param_list] <if_rom_not_in_list>targ,pong,tailg

Rule:
Add new param(s) before or after the existing **param_list** for a specified rom or list of roms.

arg_1:
A string containing new parameters and the **[param_list]** template. Note that commas are not used as a delimiter in this rule. *If **[param_list]** is not included, then all the original parameters will be omitted from the new **param_list**.*

Qualifiers:

<if_rom_in_list>	: add the parameters if the rom is in **arg_2**
<if_rom_not_in_list>	: add the parameters if the rom is not in **arg_2**

arg_2:
A comma delimited list of rom names.

---

3. change_root

Format:
<change_root> search_spec **<if_rom_in_subfolder>** folder_spec

Examples:
<change_root> [root_folder]_* **<if_rom_in_subfolder>** \roms
<change_root> [root_folder]_* **<if_rom_in_subfolder>** \software\coleco

Function:
Search for a rom in a subfolder of a version-sequenced set of parent folders of a single emulator, such as MAME. If found, the rom will be run from the location found. If not found, it will be run from the default location.

Note: *When using this rule, you should also adjust your search_path to be compatible. In this case, change search_path to*: search_path=[root_folder]_*\roms*.zip

arg_1:
A string containing a search pattern. This should be the template value **[root_folder]**, which will resolve to the root folder of the emulator (i.e. the folder defined as part of the **exe_full_path**), followed by a separator string such as an underscore, and a wildcard (*) character, which represents

the version index.

For example, given this **arg_1** value:

```
[root_folder]_*
```

The rule will expect to find the default version of the emulator in the **[root_folder]:**

```
D:\Emulators\MAME
```

and the additional ones in parallel folders named according to the search pattern where the emulator's version is placed in the position of the wildcard character (*) :

```
D:\Emulators\MAME_194
D:\Emulators\MAME_202
D:\Emulators\MAME_230
```

Note that the version of the emulator found in the default folder does not matter. It could be lower, higher, or somewhere between the versions in the other folders. It is also perfectly fine to add or remove such folders in the future without changing the rule configuration. In fact, this is kind of the point—make it easy to support new versions of emulators without changing the configuration.

Qualifier and arg_2:

The Qualifier for this rule is always **<if_rom_in_subfolder>** and **arg_2** for this rule is always a relative path from the root where the emulator's roms can be found. For example, for the MAME examples above, the Qualifer and **arg_2** values would be:

```
<if_rom_in_subfolder> \roms
```

or if the emulator was, say, MAME's ColecoVision system, then it would be

```
<if_rom_in_subfolder> \software\coleco
```

General Example:
Suppose the following rule is added to a MAME emulator section in arcadeEIP.ini.

```
rule_1=<change_root> [root_folder]_* <if_rom_in_subfolder> \roms
```

When attempting to run a rom, arcadeEIP will first search for the rom in the **\roms** subfolder of each indexed parent in reverse numeric order, ending with the default location. It will run the rom from wherever it is first found. For example, given this command-line…

```
eip.exe mame berzerk.rom
```

arcadeEIP will search to see if the rom exists in one of these folders, and if found, run it from there:

```
D:\Emulators\MAME_230\roms      ← if found, run from here
D:\Emulators\MAME_202\roms      ← if found, run from here
D:\Emulators\MAME_194\roms      ← if found, run from here
```

D:\Emulators\MAME\roms ← if found, run from here

Use Case 1:
Suppose you are standardized on MAME v.0.182 and that it resides in a folder called **D:\Emulators\MAME** with all its roms in **D:\Emulators\MAME\roms**. Now suppose you find out that a few games have improvements in MAME v.0.222. Rather than pushing your entire collection to 222, you could just install a new copy of mame in **D:\Emulators\MAME_222** and place just those roms in its **\roms** folder. With the **<change_root>** rule in place, any game dropped in **D:\Emulators\MAME_222\roms** will automatically run in that version. By removing the rom from that folder, it will revert (roll) back to running in the default MAME with *no change to the configuration*.

Use Case 1:
Now suppose you find that a few of your games actually worked better back in MAME v.172. With this rule in place, you may just place (version compatible) copies of those roms in **D:\Emulators\MAME_172\roms**. And they will now automatically run from the *older* version of MAME with no change to any configurations *even though they may still reside in your master collection in **D:\Emulators\MAME\roms***.

---

4. use_proxy

Format:
<use_proxy> file_spec **<if_in_folder>** path

Format Examples:
```
<use_proxy> [rom_name].txt <if_in_folder> D:\Emulators\MAME\proxies
<use_proxy> [rom_name].txt <if_in_folder> [root_folder]\roms
<use_proxy> [rom_name].txt <if_in_folder> D:\Emulators\Demul\Demul_digital\roms
```

Function:
Use information in a *proxy text file* to either (a) override the parameters for a specific rom or (b) vector control to another emulator, or (c) override the entire launch sequence with another executable or series of executables. The specified folder may be the same as the one in which the original rom resides, or an independent folder (so as not to clutter your rom folder).

arg_1:
A file spec. This should be the **[rom_name]** template followed by a dot and the desired extension. For example **[rom_name].txt**. A .txt extension is recommended since it is probably already associated with a text editor on your machine, and distinct from your rom file extensions, but any extension is allowed.

Note that if the extension you choose is not a normal extension used by your emulator's roms and you are using a search path, be sure to add the extension (such as txt) to the search path extension list to ensure that the proxy file is found. For example,

```
search_path=[root_folder]_*\roms\*.zip,txt
```

Qualifiers:

<if_in_folder> : use the proxy file if one is found in the path defined by **arg_2**

arg_2:

The path in which the proxy file will be searched for. The **[root_folder]** template is recommended if using this rule in conjunction with the **<change_root>** rule to ensure compatibility (see examples above). If no proxy file is found, the rom is processed normally.

Types of Proxy Files:

There are three types of proxy files, which may be freely mixed in any given emulator. You do not need to specify which kind you are using since the contents of the file will be parsed and its type detected as long as you follow the rules below. You will, of course, need to know what each of these are in order to write you proxies.

1. Vector

A "Vector" type proxy file is one which consists of a single line specifying the **sys_key** of another emulator section followed by the rom you want to launch in that emulator. For example, suppose that you find that within your MAME emulator's rom collection, you would rather launch Star Wars Trilogy (swtrilgy) using the Supermodel emulator rather than MAME. To do this, simply add a text file to the rom folder having the same name as the rom, but having a .txt extension (or whatever extension you specified in the rule). Edit this file to contain the same you would normally give eip.exe to launch that rom in supermodel. For example:

Create this file and drop it in your MAME roms folder (or whatever folder you specified in the rule):

```
swtrilgy.txt
```

Now edit the file and the appropriate command line text to vector that game to the supermodel emulator:

```
supermodel swtrilgy.zip
```

With this file in place, whenever **swtrilgy** is launched like this:

```
eip.exe mame swtrilgy
```

It will actually be launched in (vectored to) Supermodel (this, of course, assumes that you have an emulator section already defined with **sys_key=supermodel**)

This type of proxy file can be very useful to unify the launch behavior where a single system's rom files are handled by multiple emulators.

Additional Comments:

• **Creating a Surrogate Rom Collection**: Another use case for vector type proxy files is to create a surrogate rom collection. That is, a single folder containing proxy files from multiple emulators that can appear as if they were a single collection of roms to a front-end. To aid in creating such a collection, arcadeEIP supports the **-createproxies** command line argument (see the command line

reference), which can be used to bath create large sets of proxy files.

• **<vector_to> rule:** Note that If you would like to vector a set of roms based on either their file extension or on what folder the reside in, then consider using the **<vector_to>** rule instead of the **<use_proxy>** rule.

2. Parameter

A "Parameter" type proxy file is one which consists of a single line that *completely* overrides whatever **param_list** (if any) was specified in arcadeEIP.ini. This can be very handy for emulators in which command-line parameters need to be highly specific to the rom that is being launched such as in Daphne or computer emulators.

*Ensure the parameter list you specify begins with the key, "**param_list=**"*

For example, the xvic.exe (VIC-20) emulator can require parameters that specify loading addresses. By giving each rom a proxy file (in a separate folder or the same folder as the roms), each game's launch parameters can be customized separately. For example, the proxy file **defender.txt** could contain this:

```
param_list=-chdir D:\Assets\Commodore_VIC_20\Roms\Library -fullscreen -ntsc
                                -cart6 "Defender-6000.prg" -cartA "Defender-a000.prg"
```

Tip: Although the existing param_list is completely replaced, the **[param_list]** template is supported by Parameter type proxy files (also used in the **<append_params>** rule); so, it is possible to combine existing parameters from **arcadeEIP.ini** with custom parameters in the proxy file to simplify the command line in those files. In addition, all the templates normally supported by **param_list** (**[rom]**, **[rom_nam]**, etc.) are also supported in the proxy file.

So, for example, in the emulator section for the vic20 emulator, if our param_list is configured like this:

```
param_list=-chdir D:\Assets\[asset_name]\Roms\Library -fullscreen -ntsc
```

then our proxy file above could be simplified to contain the **[param_list]** template followed by just the elements relevant to this specific rom.

```
param_list=[param_list] -cart6 "Defender-6000.prg" -cartA "Defender-a000.prg"
```

This is also the recommended method for launching Daphne games in arcadeEIP.

3. Batch

A "Batch" type proxy file is one which simply consists of one or more command-lines that will be launched in sequence. These are run *exclusively* and *in lieu of* the entire normal launch sequence (including all **run_apps** for this emulator). So, if you need to run these, they must be specified as part of the batch.

Batch command lines accept the "min" and "nowait" terms as comma delimited additions to the line. Here's a hypothetical batch file with three command-lines

```
D:\Emulators\Emu\emu.exe mygame - -option2, min
D:\Scripts\Thing.exe C:\myscript.cfg,nowait
D:\Apps\MyThing2.exe -pop 1,min,nowait
```

When using Batch proxies, it is strongly recommended that you use **debug_mode** and closely inspect the log file to ensure the batch sequence you intended is correct before running for real.

General Tip for all Proxy Types:
Keep in mind that it is perfectly legal to (a) mix proxy files of different types within a single emulator, and (b) that it is also legal to have more than one **<use_proxy>** rule within a single emulator section. For example, if you had a base emulator that was at, say, version 11, but wanted some roms to be run in earlier versions (say 9 or 10), then you could set up a proxy scheme like this within the base emulator section. This would keep the original rom folder unaltered, and still make it easy to see at a glance what roms get run with which version.

```
rule_1=<use_proxy> [rom_name].txt <if_in_folder> D:\Emulators\MyEmu\v9proxies
rule_2=<use_proxy> [rom_name].txt <if_in_folder> D:\Emulators\MyEmu\v10proxies
```

Or, if you preferred to keep the proxies in the same folder (say the base emulator's rom folder)…

```
rule_1=<use_proxy> [rom_name].v9 <if_in_folder> D:\Emulators\MyEmu\roms
rule_2=<use_proxy> [rom_name].v10 <if_in_folder> D:\Emulators\MyEmu\roms
```

---

5. vector_to

Format:
<vector_to> sys_key **<if_extension_in_list>** ext[,ext]…
<vector_to> sys_key **<if_rom_rom_in_folder>** path

Examples:
```
<vector_to> vpinball9 <if_extension_in_list> vpt
<vector_to> naomi_analog <if_rom_in_folder> D:\Emulators\Demul\Demul_analog\roms
```

Rule:
Vector a set of roms to use another emulator section. This is similar to the vector type **<use_proxy>** function but is intended to work on sets of roms rather than individual ones. The **<vector_to>** rule lacks some of the flexibility of using proxies, but works without proxy files and can be perfectly adequate in cases where roms to be vectored are already segregated by their file extension or by what folder they reside in.

arg_1:
A string containing a valid **sys_key** to another emulator section.

Qualifiers:

<if_extension_in_list>	: vector the rom if it's extension is found in the **arg_2** list.
<if_rom_in_folder>	: vector the rom if it is found in the **arg_2** folder.

arg_2:

If the qualifier is **<if_extension_in_list>** then this is one or more file extensions, comma separated, and without any dots or wildcards.

If the qualifier is **<if_rom_in_folder>** then this is a file path (including drive letter) with no trailing backslash.

---

6. Custom Functions

Summary: *Custom functions are effectively "plug-ins" and are created by the user. An example of a custom function is provided in the file* **fx_add_params.ahk**, *and must be AHK compiled as an exe and placed in the same file folder as* **eip.exe** *to use. Once this is done, a custom function may be added to an emulator section in* **arcadeEIP.ini** *just like any other rule.*

<fx_*custom_function*> *custom_arg_1* **<*custom_qualiier*>** *custom_arg_2*

<fx_*custom _function*>:
Custom functions can have any name, with it being recommended to compose them of two parts separated with an underscore (e.g. *verb_noun*). They *must* be prefixed with **fx_**. Each custom function defined in arcadeEIP.ini must be compiled into an executable file containing the function's implementation, such as **fx_add_params.exe** or **fx_swap_roms.exe** and located in the same folder as **eip.exe**. An example custom function AHK script called **fx_add_params.ahk** is included with the distribution and may be compiled and the commented code used as a tutorial.

***custom_arg_1*:**
This is a value entered by the end user which typically defines a target for the function. See existing functions for models.

<custom_qualifer>
This may contain any value chosen by the function designer. The qualifer typically indicates a condition or constraint. See existing functions for models.

***custom_arg_2*:**
This is a value entered by the end user which typically represents the specific terms of the qualifier (what the condition or constraint is). See existing functions for models.

Example:
See **fx_add_params.ahk** for an example and for further information about writing custom functions. This example file provides the implementation of a simple custom rule that, if compiled as an exe an named **fx_add_params.exe**, allows use of this rule for adding parameters to the end of the parameter list for specific roms.

```
rule_1=<fx_add_param> -joy <if_rom_in_list> dkong,defender
```