Advanced Batocera Configuration Syntax

Most settings (including RetroArch-specific settings) can be adjusted from within the menus in EmulationStation. This includes per system and per game settings, access per system settings by pressing [SELECT] while in the system list and scrolling to **ADVANCED SYSTEM SETTINGS** and access per game settings by holding  while the game in question is selected (in Batocera **v31** and lower, this is instead tapping ).

However, there are a couple of (usually very recently implemented) settings that can only be set by manually editing files. This includes RetroArch's *core-specific* options which **are not** already available in the per system settings.

If you were looking for how to activate the official Nvidia drivers, check out its specific page instead.

If you were looking on how to change `batocera-boot.conf` settings, check out the boot partition editing page instead.

batocera.conf option keys and values

`/userdata/system/batocera.conf` is the main user-configuration file in Batocera, it can be accessed from the network share or the file manager. This will contain all the settings you may have set in **GAME SETTINGS**, **ADVANCED PER SYSTEM OPTIONS** and a few other places.

A surprising amount of configuration is handled by EmulationStation's internal system (independent of Batocera), the configuration file of which can be found at `/userdata/system/configs/emulationstation`.

The default `batocera.conf` can be found here on the Github.

`batocera.conf` uses the following syntax (objects in <> are required and have to be replaced, including the <> symbols themselves, objects in ( ) are optional and objects separated by a pipe | indicate exclusive or):

```
(<system>(["<ROM filename>.<ROM extension>"].)|folder["<path>"].)<key
name>(.<subkey 1 name> ...)=<value>
```

which can be boiled down in its simplest form to:

```
<option>=<value>
```

Batocera option

It can be a bit confusing seeing all the possibilities laid out like above so here's a simple example for a Batocera option:

```
audio.bgmusic=0
```

This will turn off the background music in the menu. Here, the key name is “audio”, the first subkey name is “bgmusic” and the value is “1”. No <system>, <ROM filename> or <ROM extension> is used.

System-specific option

Here's an example of changing a setting for every game of a specific system:

```
nes.bezel=none
```

This will turn off decorations for all games of the NES system. Here, the system is “nes”, the key name is “bezel” and the value is “none”. No <subkey # name>, <ROM filename> or <ROM extension> is used.

Game-specific option

Here's an example of changing a setting for a specific game of a system:

```
c64["Arkanoid - Revenge of Doh (USA, Europe).zip"].external_palette=vice
```

This will change the **EXTERNAL PALETTE** option for the Commodore 64 game “Arkanoid - Revenge of Doh” to use the “vice” palette. Here, the system is “c64”, the ROM filename and extension is “Arkanoid - Revenge of Doh (USA, Europe).zip”, the key name is “external_palette” and the value is “vice”. No <subkey # name> is used.

Folder-specific option (introduced in **v34**)

Here's an example of setting an option for games launched from a certain folder:

```
nes.folder["/userdata/roms/nes/Japan/"].shaderset=Mega-Bezel-Community-JP-Night
```

This will set the shader set used for all NES games in the /userdata/roms/nes/Japan folder to use the night variant of the Famicom decoration (Mega-Bezel-Community-JP-Night).

Game-specific and folder-specific setting syntaxes conflict with one another. A key using both at the same time will be invalid and ignored.

Key names, systems and values are not locale-specific, they will be the same regardless of which

language you are using. The option names themselves are only translated on ES itself when in the menu, the internal key names stay the same.

The absence of a key/value will imply its default "AUTO" setting will be used. In fact, this is the only way to use the “AUTO” setting.

Comments

Comments are treated a bit special with `batocera.conf`. A general comment that exists to explain what an option might do (a few of these are already included by default) is indicated by two hash symbols opening the line. For example:

```
## Set the audio device
```

which will simply be completely ignored by Batocera.

Then there are special “disable setting” comments, which are indicated by a single hash opening the line. For example:

```
#wifi.key=new key
```

These settings, which still commented out, will be ignored by Batocera. However, if using ''batocera-settings-get'' or ''batocera-settings-set'' from SSH, error code 11 will be returned (“Value error, key found but it is commented out”). In order to fix this, either manually remove the comment from the setting or activate the key entry by using the `uncomment` command.

How do I find out what available keys I can use are?

If you'd like to see what the internal names of the options are, the quickest way is to set the setting in ES itself and exiting out of the menu to refresh `batocera.conf`, seeing what new line was added to `batocera.conf`.

However, if you'd like to find the keys manually, you have to change your approach depending on which keys you are after:

- To find the global keys (ones that can be applied to *most* systems) manually, look through the code and figure out its logic in the launcher script (in the `/usr/lib/python#.#/site-packages/configgen/` folder).
- To find the emulator keys (of which apply to all systems that emulator supports, usually) manually, look through the code and figure out its logic in its configuration generator (most of which are in `/usr/lib/python#.#/site-packages/configgen/generators/`).
- To see system-specific keys, look at the `/usr/share/emulationstation/es_features.cfg` file. The `features` tag shows which “global” keys can be applied to your system, while the rest of the system-specific keys are specified in their own core option section.

Although option names that appear in the menu may change from time to time, the internal option names tend to be resilient to change. This is done to ensure that `batocera.conf` files created with newer versions will have the highest chance of being perfectly backward compatible with older versions of Batocera (with options introduced with newer versions of Batocera simply being ignored).

However, it is possible that the *values* of some options may be changed/amended, causing the “future” values to be treated as 'default' by versions of Batocera that don't support it. It is important to keep this in mind if you are using versions of Batocera that are very far apart.

"AUTO" setting behavior

As mentioned above, leaving a key absent will use its “AUTO” setting. But what is that, exactly? It's complicated, technically it could be anything depending on the platform, system, generator, etc. The understanding of Python and its PYAML syntax is required.

When an option is on “AUTO”, first Batocera checks if there is any “default” value for the option in `/usr/share/batocera/configgen/configgen-defaults.yml` and apply them to the configuration. For example, this is how the “default” emulator is selected for each system. Then, it checks `/usr/share/batocera/configgen/configgen-defaults-arch.yml` for any platform-specific defaults that should be applied. For example, this is how the “default” emulator is changed on platforms that don't have the same amount of emulators available to them (like RPi 0 and such).

When neither of the above files have a value for the key in question, the “AUTO” behavior is then dependent on configuration generator for that particular emulator.

RetroArch-specific options

RetroArch is a special exception, being as utilized as it is by Batocera it makes sense that it would have its own set of subkeys.

First is the general RetroArch settings. Don't be fooled, not all of RetroArch's settings have been integrated into Batocera, as every key needs to be manually coded in to be respected. Batocera is mostly on the ball with this, but sometimes things will slip through/be unavailable for other reasons (like menu option visibility, as the wrong menu option could break Batocera).

RetroArch settings are technically per-system settings, so the `<system>[\["<ROM filename>.<ROM extension>"\].` part at the start is still required, but for most purposes `global.` can be used.

The subkey name for RetroArch's settings is `retroarch` and the subkey after that is usually identical to RetroArch's key for the respective setting. Unlike RetroArch, no double quotes are required for the setting's value, however including them won't break anything. Here is the syntax:

```
<system>(["<ROM filename>.<ROM extension>"]|folder["<path>"]).retroarch.<RetroArch setting name>=<value>
```

which can be boiled down to (if desiring to apply it to all systems):

```
<system>.retroarch.<RetroArch setting name>=<value>
```

For example:

```
global.retroarch.video_hard_sync=true
```

RetroArch settings can also be applied to only a specific system. For example:

```
saturn.retroarch.video_hard_sync=true
```

A list of the currently known RetroArch settings and more details about their purpose can be found on its respective page.

RetroArch core-specific options

In addition to RetroArch's general settings, each libretro core has its own set of settings that can be configured as well. Unlike with RetroArch general settings, here any core setting can be set irrelevant of whether Batocera knows about it yet or not. The syntax is as follows:

```
<system>(["<ROM filename>.<ROM extension>"].)retroarchcore.<core
option>=<value>
```

which can be boiled down to:

```
<system>.retroarchcore.<core option>=<value>
```

For example:

```
global.retroarchcore.vice_external_palette=vice
```

Yes, this does mean you can set core options per system (some cores are capable of emulating multiple systems as well, maybe a setting doesn't need to be applied to all of them). For example:

```
c64.retroarchcore.vice_external_palette=vice
```

Outside of the core developer providing it themselves, there is no documentation on what internal names the core options use. Probably because the end-user should never have to be editing them like this in the first place.

You can manually deduced these options yourself by following the instructions on the Advanced RetroArch settings page.

/userdata/system/configs/emulationstation

To do: EmulationStation's advanced configuration.