

Coding Rules

The coding principals for Batocera are as follows:

- Keep it simple
- Worse is better

In other words, don't try to handle all cases if it makes the software complicated and unmaintainable.

- Don't add new systems that are simply variations of other systems, aim to have one folder per system. An exception to this rule would be if a system required different configuration to launch, as asking the user to create two sets of custom emulator settings should be avoided.
- Don't put in [device](#)-specific `if` conditions into Batocera's `/etc/init.d` scripts. These should be applied via [udev rules](#) for proper identification. Ultimately, it would be preferable to fix the global script in the first place to be suitable for all situations.
- Build emulators from their source where possible.
- Don't add emulators that duplicate the function of other emulators. We must avoid redundancy as there is an upper limit as to how much data can be packed into the batocera update file.
- Avoid making patches if possible, encourage the source developer of the software to fix issues upstream.
- Avoid adding files to the `fsoverlay`. Fix the core root of the issue instead.



A timeline of when certain changes are allowed to be merged upstream can be found here: <https://batocera.org/timeline>

Project files

The [batocera.linux](#) git is based on a Buildroot tree. Ideally, a file is either owned by Batocera or by Buildroot. [Buildroot files must not be modified](#).

The following directories are owned by Batocera:

defconfig build files

Located at <https://github.com/batocera-linux/batocera.linux/tree/master/configs>, these are the defconfig files required to build Batocera for a particular platform. These files must be kept as small as possible. Other necessary alterations can be handled by packages from <https://github.com/batocera-linux/batocera.linux/blob/master/package/batocera/core/batocera-system/Config.in>.

The only information that you should find in this file are the physical architecture and the packages it requires (when it is technically not possible to put them in the `Config.in` with `KConfig`).

Board patches

Located at <https://github.com/batocera-linux/batocera.linux/tree/master/board/batocera>, this is the place where you can find files specific to each board. This directory must be kept as small as possible; ideally most files should be handled by packages.

Every platform has several key directories:

- **fsoverlay** Files that will be globally copied over to the final Batocera image. This directory is ideally empty. Avoid adding files here, instead, add them in a package. The same folder exists for each board: `<architecture>/fsoverlay`.
- **patches** Package patches. The same folder exists specific to each board: `<architecture>/patches`.
- **<architecture>/boot** Files specific to the boot.
- **<architecture>/linux_patches** Linux patches specific to the board.

Packages

Located at <https://github.com/batocera-linux/batocera.linux/tree/master/package/batocera>, these are the packages executable after Batocera has booted. Essentially, the “main part” of Batocera. Try to organize most auxiliary packages required for booting up successfully here.

Packages should be independent (executable outside of `Config.in` dependencies, accessible to any platform) and contained entirely within their own sub-folder. The idea is to keep these modular, so no one package is dependent on the other.



When you've become more familiar with the basics, you can refer to the [notable files/folders page](#) as a cheatsheet!



Windows Vista and above support symlinks, but this is not built-in by default. Here's an unofficial extension to explorer.exe that adds such functionality: <https://schinagl.priv.at/nt/hardlinkshellext/linkshellextension.html>

Global features/advanced system settings

Define global features and advanced system options in the `es_features.yml` file ([available on the Github here](#)). This is a human-readable file containing all the advanced settings available in ES.

Syntax (replace everything including less-than and greater-than symbols (<>)):

```
<emulator_1_shortname>: # We refer to the internal name of the emulator, not the system name.
```

```

  group: <group name> # An optional group to sort this feature under. This
is mostly used to categorize "advanced options" together, not much else.
  features: [<common feature 1>, <common feature 2>, <...>] # All the global
options applicable to this emulator.
  cfeatures: # Custom features, found in the advanced settings.
    <batocera.conf key name 1>: # This is what is written as the key
name to batocera.conf
      archs_include: [<arch>] # Platforms to limit this option to, if
applicable.
      prompt:          <ALL CAPS OPTION NAME> # All caps because
consistency with other labels in ES.
      description: <Explanation of the sentence that doesn't exceed
1000px across.>
      choices:
        "<Option 1 name>":    <batocera.conf value 1> # This is the
value written after the "=" in batocera.conf.
        "<Option 2 name>":    <batocera.conf value 2>
        "<...>":              <...> # etc. for additional values.
    <batocera.conf key name 2>: # Add as many unique keys as needed.
      <...>
# This section is optional. If there are settings in your emulator that only
apply to a specific
# system, you can add them in here.
  systems:
    <system 1 shortname>:
      <batocera.conf key name 3>: # Options that only apply to system 1
      <...> # As above.
<emulator 2 shortname>: # Another emulator's settings follow this, so on and
so forth...
  <...>

```

Here's an example:

```

dolphin:
  features: [ratio]
  cfeatures:
    gfxbackend:
      archs_include: [x86, x86_64, rpi4, odroidgoa, gameforce]
      prompt:          GRAPHICS BACKEND
      description: Choose your graphics rendering
      choices:
        "OpenGL": OGL
        "Vulkan": Vulkan
    perf_hacks:
      group: ADVANCED OPTIONS
      prompt:          PERFORMANCE HACKS
      description: Increase emulator performance, at the cost of
accuracy/stability
      choices:
        "Off": 0
        "On": 1

```

```


systems:
  wii:
    cfeatures:
      emulatedwiimotes:
        group: ADVANCED OPTIONS
        prompt:      EMULATE WIIMOTE
        description: Use your gamepad like a vertical Wiimote in
game
        choices:
          "Off":      0
          "On":       1

```

Spacing is everything here. Triple-check that you have used the correct amount of indentation. If your build fails to compile because of errors related to `es_features.yml` after you have made modifications to it then this is likely the culprit.



A new way to use options is available as presets, read more about it on [its pull request](#). This also introduced the ability to have global options in the per-system option. Don't do that in Batocera.

 **Fix Me!** (how is this done in Batocera, though?)

General practice isn't to be a one-for-one representation of the settings available to the emulator, that would defy the point of Batocera. Important settings that may need to be changed frequently/per game should be represented as is, but generally the more advanced settings should be simplified (with the help of the config generator setting all the correct values for that emulator's configuration files as necessary) and clearly explained in the advanced system settings. For example, the above PERFORMANCE HACKS setting actually changes a bunch of settings within Dolphin all at once. Here's a snippet of its config generator:

```

# Various performance hacks - Default Off
if system.isOptSet('perf_hacks') and
system.getOptBoolean('perf_hacks'):
    dolphinGFXSettings.set("Hacks", "BBoxEnable", "False")
    dolphinGFXSettings.set("Hacks", "DeferEFBCopies", "True")
    dolphinGFXSettings.set("Hacks", "EFBEmulateFormatChanges",
"False")
    dolphinGFXSettings.set("Hacks", "EFBScaledCopy", "True")
    dolphinGFXSettings.set("Hacks", "EFBToTextureEnable", "True")
    dolphinGFXSettings.set("Hacks", "SkipDuplicateXFBS", "True")
    dolphinGFXSettings.set("Hacks", "XFBSToTextureEnable", "True")
    dolphinGFXSettings.set("Enhancements", "ForceFiltering",
"True")
    dolphinGFXSettings.set("Enhancements",
"ArbitraryMipmapDetection", "True")
    dolphinGFXSettings.set("Enhancements", "DisableCopyFilter",
"True")

```

```

    dolphinGFXSettings.set("Enhancements", "ForceTrueColor",
    "True")
    else:
        if dolphinGFXSettings.has_section("Hacks"):
            dolphinGFXSettings.remove_option("Hacks", "BBoxEnable")
            dolphinGFXSettings.remove_option("Hacks", "DeferEFBCopies")
            dolphinGFXSettings.remove_option("Hacks",
"EFBEmulateFormatChanges")
            dolphinGFXSettings.remove_option("Hacks", "EFBScaledCopy")
            dolphinGFXSettings.remove_option("Hacks",
"EFBToTextureEnable")
            dolphinGFXSettings.remove_option("Hacks",
"SkipDuplicateXFBs")
            dolphinGFXSettings.remove_option("Hacks",
"XFBToTextureEnable")
            if dolphinGFXSettings.has_section("Enhancements"):
                dolphinGFXSettings.remove_option("Enhancements",
"ForceFiltering")
                dolphinGFXSettings.remove_option("Enhancements",
"ArbitraryMipmapDetection")
                dolphinGFXSettings.remove_option("Enhancements",
"DisableCopyFilter")
                dolphinGFXSettings.remove_option("Enhancements",
"ForceTrueColor")

```

Multi-line descriptions

Use of multi-line descriptions is discouraged, but for excessively complicated options requiring more detailed descriptions exceptions can be made.

A multi-line description can be enabled by prepending `| -` into the description field for the option and using ordinary YAML multi-line strings.

```

[...]
prompt:      HEADS UP DISPLAY
  description: | -
              Show context-sensitive info
              over the game
              and a 3rd line
              and a 4th line.
  submenu:   DECORATIONS
[...]

```

How emulator configuration applies

For each emulator, a config generator plugin must be written to update the emulator's config files before the emulator launches. There are two kinds of configuration: The one handled by Batocera and static, user-editable files. To let the user modify static configuration files:

- Don't base the configuration on templates (a.k.a. "magic" values, hard-coded values, etc.)
- Only update values controlled by Batocera

Configuration handled by Batocera comes from:

- the configgen plugin. hardcoded or dependant on non user values (the number of plugged pads for example)
- **configgen-defaults.yml** the default configuration
- **configgen-defaults-arch.yml** the configuration that overrides configgen-defaults.yml defaults for a given architecture
- **batocera.conf** The settings file saved to by EmulationStation or manually edited by the user. Contains keys such as `global.<setting>=<value>` or `<system>.<emulator>.<setting>=<value>`



Ideally, `batocera.conf` must be installed empty and keep only user-changed settings. Default system configurations must be only in YML files. EmulationStation is capable of editing the `batocera.conf` file.

Patches

On a package

Buildroot utilizes a great feature: patching programs.

Patches are separate files that will modify a program at compilation time. There are three locations where you can put patches:

- In the `package/batocera/<package_name>` directory. This is the preferred place for Batocera packages when the patch is required for all architectures. It is very visible for the package maintainer.
- In the `board/batocera/patches/<package_name>` directory. This is the preferred place when patching a Buildroot package that affects all architectures.
- In the `board/batocera/<architecture>/patches/<package_name>` directory. This is the preferred place when patch a package having an issue with a specific board. If the patch is not related to the board or has no impact for other boards (like adding a choice entry in a Makefile for a board), the preferred place is not this one. It is preferred to reduce the number of entries in this place to reduce the differences between boards.

On Buildroot

Avoid modifying Buildroot's files. This makes upgrading Buildroot more difficult. The following script: <https://github.com/batocera-linux/batocera.linux/blob/master/scripts/linux/buildrootdiff.sh> may help you to find altered Buildroot files. The number of files found using this must be reduced to the minimum amount required.

In case a modification to a Buildroot file is absolutely necessary, prefix each paragraph by '# batocera' to help during the merges.

When to merge buildroot

The good way to update buildroot packages is to merge the last official buildroot tree (stable if possible). During last month before a release, we avoid buildroot merges to avoid adding unpredictable bugs. There are some cases we can merge buildroot packages directly, because it worth to have up to date packages. It includes :

- mesa packages (gpu drivers)
- pipewire packages (because it is in a very active phase and solving regularly issues)
- linux-firmwares (because it is incrementing files, not real risk)
- rpi kernels (to be up to date)
- rust (because ruffle uses the last versions)
- linux LTS kernels

From:

<https://wiki.batocera.org/> - **Batocera.linux** - Wiki

Permanent link:

https://wiki.batocera.org/coding_rules

Last update: **2022/07/21 02:43**

