Mosestyle Kodi Build — Checkpoint 1

Clean install flow using your uploaded files • Installer v1.5.2

---

0) Prereqs

- **Repo:** `https://github.com/mosestyle/kodi` (branch **master**)
- **`_repo_generator.py`** at the repo root (same level as `repo/`, `plugin.program.mosestylebuild/`, etc.)

---

1) Folder layout (in your repo)

```
kodi/
├─ builds/
│  └─ mosestyle-omega-1.0.0.zip         ← your manual build ZIP
│
├─ plugin.program.mosestylebuild/
│  ├─ addon.xml
│  ├─ default.py
│  └─ resources/
│     └─ settings.xml
│
├─ repository.mosestyle/
│  └─ addon.xml
│
├─ repo/
│  └─ zips/                             ← generated by _repo_generator.py
│     ├─ addons.xml
│     ├─ addons.xml.md5
│     ├─ repository.mosestyle/repository.mosestyle-1.0.0.zip
│     └─ plugin.program.mosestylebuild/plugin.program.mosestylebuild-1.5.2.zip
│
└─ index.html                           ← links to repo zip at repo *root*
```

---

2) Files (put these **exact** contents)

`index.html`

```
<!DOCTYPE html>
<a href="repository.mosestyle-1.0.0.zip">repository.mosestyle-1.0.0.zip</a>
```

`repository.mosestyle/addon.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<addon id="repository.mosestyle" name="Mosestyle Kodi Repo" version="1.0.0"
provider-name="Mosestyle">
  <requires>
    <import addon="xbmc.addon" version="12.0.0"/>
  </requires>
  <extension point="xbmc.addon.repository" name="Mosestyle Kodi Repo">
    <dir>
      <info
compressed="false">https://raw.githubusercontent.com/mosestyle/kodi/master/repo/zips/addons
.xml</info>

<checksum>https://raw.githubusercontent.com/mosestyle/kodi/master/repo/zips/addons.xml.md5<
```

```
/checksum>
      <datadir
zip="true">https://raw.githubusercontent.com/mosestyle/kodi/master/repo/zips/</datadir>
    </dir>
  </extension>
  <extension point="xbmc.addon.metadata">
    <summary>Mosestyle Kodi Repository</summary>
    <description>Add-ons maintained by Mosestyle.</description>
    <platform>all</platform>
  </extension>
</addon>
```

`plugin.program.mosestylebuild/addon.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<addon id="plugin.program.mosestylebuild" name="Mosestyle Build Installer" version="1.5.2"
provider-name="Mosestyle">
  <requires>
    <import addon="xbmc.python" version="3.0.0"/>
  </requires>
  <extension point="xbmc.python.script" library="default.py" />
  <extension point="xbmc.addon.metadata">
    <summary>Install Mosestyle's Kodi build</summary>
    <description>Downloads build, applies global settings, auto-enables add-ons and
switches skin.</description>
    <platform>all</platform>
  </extension>
</addon>
```

`plugin.program.mosestylebuild/resources/settings.xml`

```
<settings>
  <category label="Build">
    <setting id="build_url" type="text" label="Build ZIP URL"

default="https://raw.githubusercontent.com/mosestyle/kodi/master/builds/mosestyle-omega-1.0
.0.zip"
/>
    <setting id="fresh" type="bool" label="Fresh install (wipe most user data first)"
default="false" />
  </category>
  <category label="Automation">
    <setting id="enable_unknown_sources" type="bool" label="Enable 'Unknown sources'"
default="true" />
    <setting id="auto_enable_addons" type="bool" label="Auto-enable add-ons included in
build" default="true" />
    <setting id="auto_set_skin" type="bool" label="Auto-switch to build’s skin"
default="true" />
    <setting id="skin_id_override" type="text" label="(Optional) Force skin id (e.g.
skin.fentastic)" default="" />
    <setting id="auto_restart" type="bool" label="Restart Kodi when finished"
default="false" />
  </category>
</settings>
```

`plugin.program.mosestylebuild/default.py`

```
# Mosestyle Build Installer (v1.5.2) — Option A (manual exit, JSON-RPC only)
import xbmc, xbmcgui, xbmcaddon, xbmcvfs
import os, zipfile, urllib.request, shutil, json, re, time
from xml.etree import ElementTree as ET

ADDON = xbmcaddon.Addon()
HOME  = xbmcvfs.translatePath('special://home/')
PKGS  = xbmcvfs.translatePath('special://home/addons/packages/')
```

```
TMP_ZIP = os.path.join(PKGS, 'mosestyle_build.zip')

def log(msg): xbmc.log(f"[MosestyleBuild] {msg}", xbmc.LOGINFO)

def rpc(method, params=None):
    payload = {"jsonrpc":"2.0","id":1,"method":method}
    if params is not None: payload["params"] = params
    try:
        return json.loads(xbmc.executeJSONRPC(json.dumps(payload)))
    except Exception as e:
        log(f"JSON parse error: {e}")
        return {}

def download(url, dst):
    xbmcvfs.mkdirs(PKGS)
    with urllib.request.urlopen(url) as r, open(dst, 'wb') as f:
        f.write(r.read())

def safe_wipe():
    keep = {'addons','userdata','addons/packages'}
    for name in os.listdir(HOME):
        if name in keep: continue
        p = os.path.join(HOME, name)
        try:
            shutil.rmtree(p, ignore_errors=True) if os.path.isdir(p) else os.remove(p)
        except Exception as e:
            log(f"wipe top error {name}: {e}")
    u = os.path.join(HOME, 'userdata')
    if os.path.isdir(u):
        for name in os.listdir(u):
            if name.lower() in ('database','thumbnails'): continue
            p = os.path.join(u, name)
            try:
                shutil.rmtree(p, ignore_errors=True) if os.path.isdir(p) else os.remove(p)
            except Exception as e:
                log(f"wipe userdata error {name}: {e}")

def extract_to_home(zip_path):
    with zipfile.ZipFile(zip_path, 'r') as z:
        z.extractall(HOME)

def parse_zip_for_addons_and_skins(zip_path):
    addon_ids, skin_ids = [], []
    with zipfile.ZipFile(zip_path, 'r') as z:
        for n in z.namelist():
            if not n.lower().endswith('addon.xml'): continue
            if not re.search(r'(^|/)addons/[^/]+/addon\.xml$', n, re.IGNORECASE): continue
            try:
                root = ET.fromstring(z.read(n))
                aid = root.attrib.get('id')
                if aid and aid not in addon_ids:
                    addon_ids.append(aid)
                for ext in root.findall('extension'):
                  if ext.attrib.get('point') in ('xbmc.gui.skin','kodi.gui.skin'):
                        if aid not in skin_ids:
                            skin_ids.append(aid)
            except Exception as e:
                log(f"parse failed for {n}: {e}")
    return addon_ids, skin_ids

def enable_unknown_sources():
    if ADDON.getSettingBool('enable_unknown_sources'):
        rpc("Settings.SetSettingValue", {"setting":"addons.unknownsources","value":True})

def update_local_addons_and_wait(target_ids, timeout=120):
    xbmc.executebuiltin('UpdateLocalAddons')
    deadline = time.time() + timeout
    pending = set(target_ids)
```

```
    mon = xbmc.Monitor()
    while pending and time.time() < deadline and not mon.abortRequested():
        for aid in list(pending):
            details = rpc("Addons.GetAddonDetails", {"addonid": aid,
"properties":["enabled","name","version"]})
            if details.get("result",{}).get("addon"):
                pending.discard(aid)
        xbmc.sleep(500)
    if pending:
        log(f"Timeout waiting for addons to register: {sorted(pending)}")

def enable_addons(addon_ids):
    if not ADDON.getSettingBool('auto_enable_addons'):
        return
    for aid in addon_ids:
        rpc("Addons.SetAddonEnabled", {"addonid": aid, "enabled": True})
        details = rpc("Addons.GetAddonDetails", {"addonid": aid, "properties":["enabled"]})
        ok = bool(details.get("result",{}).get("addon",{}).get("enabled"))
        log(f"enable {aid}: {'OK' if ok else 'FAILED'}")

# ----- typed/aliased settings you wanted -----
TARGETS = [
    ("subtitles.align|subtitles.position", 2, ["int"]),                                #
Subtitles pos: Manual
    ("subtitles.languages", ["English","Swedish"], ["list_str","csv_str"]),            #
Sub download langs
    ("videoplayer.seeksteps|videoscreen.seeksteps", [-10,10], ["list_int","csv_str"]), #
Skip steps
    ("videoplayer.seekdelay|videoscreen.seekdelay", 750, ["int"]),                     #
Skip delay 750 ms
    ("videoscreen.adjustrefreshrate|videoplayer.adjustrefreshrate", 2, ["int"]),       #
Adjust refresh: On start/stop
    ("videoplayer.syncdisplay", False, ["bool"]),                                      #
Sync to display: Off
    ("videoplayer.stretch43|videoscreen.stretch43", 2, ["int"]),                       #
4:3 stretch = 16:9
    ("audiooutput.guisoundmode|audiooutput.guisounds", 0, ["int"]),                    #
GUI sounds: Never
    ("locale.audiolanguage", "English", ["str"]),                                      #
Preferred audio language
    ("locale.subtitlelanguage", "English", ["str"]),                                   #
Preferred subtitle language
    ("subtitles.moviesdefaultservice|subtitles.movieservice",
"service.subtitles.a4ksubtitles", ["str"]),
    ("subtitles.tvshowsdefaultservice|subtitles.tvshowservice",
"service.subtitles.a4ksubtitles", ["str"]),
    ("videoplayer.viewmode|videoscreen.viewmode", 6, ["int"]),                         #
Try View mode = Stretch 16:9
]

def encode_value(value, mode):
    if mode == "int":      return int(value)
    if mode == "bool":     return bool(value)
    if mode == "str":      return str(value)
    if mode == "list_str":
        if isinstance(value, (list,tuple)): return [str(x) for x in value]
        return [s.strip() for s in str(value).split(",") if s.strip()]
    if mode == "list_int":
        if isinstance(value, (list,tuple)): return [int(x) for x in value]
        return [int(s) for s in str(value).split(",") if s.strip()]
    if mode == "csv_str":
        if isinstance(value, (list,tuple)): return ",".join(str(x) for x in value)
        return str(value)
    return value

def apply_overrides():
    for ids, value, modes in TARGETS:
        applied = False
```

```
        for k in ids.split("|"):
            for m in modes:
                encoded = encode_value(value, m)
                r = rpc("Settings.SetSettingValue", {"setting": k, "value": encoded})
                if r.get("result") == "OK":
                    log(f"Override {k} [{m}] -> {encoded} OK")
                    applied = True
                    break
            if applied: break
        if not applied:
            log(f"Override FAILED: {ids} -> {value}")

def coerce(val):
    if isinstance(val, (bool,int,float)): return val
    s = (val if isinstance(val, str) else str(val)).strip().lower()
    if s in ("true","false"): return s == "true"
    try:
        return float(val) if "." in str(val) else int(val)
    except Exception:
        return val

def apply_global_settings_from_guisettings():
    path = xbmcvfs.translatePath('special://profile/guisettings.xml')
    if not xbmcvfs.exists(path):
        path = os.path.join(HOME, 'userdata', 'guisettings.xml')
        if not xbmcvfs.exists(path):
            log("No guisettings.xml found to apply.")
            return
    try:
        with xbmcvfs.File(path) as f: data = f.read()
        root = ET.fromstring(data)
        total = applied = 0
        for node in root.findall('.//setting'):
            sid = node.attrib.get('id')
            if not sid: continue
            value = node.text if (node.text is not None) else node.attrib.get('value')
            if value is None: continue
            total += 1
            r = rpc("Settings.SetSettingValue", {"setting": sid, "value": coerce(value)})
            if r.get("result") == "OK": applied += 1
        log(f"Applied {applied}/{total} settings from guisettings.xml")
    except Exception as e:
        log(f"apply_global_settings_from_guisettings error: {e}")

def set_skin(skin_id):
    rpc("Addons.SetAddonEnabled", {"addonid": skin_id, "enabled": True})
    r = rpc("Settings.SetSettingValue", {"setting":"lookandfeel.skin","value":skin_id})
    log(f"switch skin to {skin_id}: {r.get('result')}")

def main():
    xbmcgui.Dialog().notification('Mosestyle Build', 'Starting installer…',
xbmcgui.NOTIFICATION_INFO, 2500)

    url = ADDON.getSettingString('build_url').strip()
    fresh = ADDON.getSettingBool('fresh')
    if not url:
        xbmcgui.Dialog().ok(ADDON.getAddonInfo('name'), 'No Build ZIP URL set.'); return
    if not xbmcgui.Dialog().yesno('Mosestyle Build', f'Install build from:\n[COLOR
cyan]{url}[/COLOR]\n\nContinue?'):
        return

    try:
        enable_unknown_sources()
        download(url, TMP_ZIP)
        addon_ids, skin_ids = parse_zip_for_addons_and_skins(TMP_ZIP)

        if fresh and xbmcgui.Dialog().yesno('Fresh Install?', 'This wipes most of your
current Kodi data first (keeps Thumbnails/Database).\n\nProceed?'):
```

```
            safe_wipe()

        extract_to_home(TMP_ZIP)
        update_local_addons_and_wait(addon_ids)
        enable_addons(addon_ids)

        # Apply settings from your guisettings.xml, then apply the overrides list above
        apply_global_settings_from_guisettings()
        apply_overrides()

        if ADDON.getSettingBool('auto_set_skin'):
            target = ADDON.getSettingString('skin_id_override').strip() or (skin_ids[0] if
skin_ids else "")
            if target: set_skin(target)

        # Final short reminder (no restart text)
        xbmcgui.Dialog().ok(
            'Mosestyle Build',
            "[B]Don’t forget:[/B]\n"
            "• Subtitles = [COLOR cyan]Manual[/COLOR]\n"
            "• Video view mode = [COLOR cyan]Stretch 16:9[/COLOR]\n"
            "• FENLight = [COLOR cyan]Restore Picture[/COLOR]\n"
        )

    except Exception as e:
        xbmcgui.Dialog().ok('Mosestyle Build', f'Install failed:\n{e}')

if __name__ == "__main__":
    main()
```

---

3) Make your build ZIP (manual)

Create `builds/mosestyle-omega-1.0.0.zip` with only:

```
addons/
userdata/   (e.g., addon_data/, keymaps/gen.xml, guisettings.xml, etc.)
```

Do **not** include `userdata/Thumbnails/` or `userdata/Database/`.

---

4) Commit

Commit all changes (**build zip** + add-on folders + **index.html**).

---

5) Generate repository zips + copy link zip

From the repo root:

```
python _repo_generator.py
```

This creates/updates:

```
repo/zips/addons.xml
repo/zips/addons.xml.md5
repo/zips/repository.mosestyle/repository.mosestyle-1.0.0.zip
repo/zips/plugin.program.mosestylebuild/plugin.program.mosestylebuild-1.5.2.zip
```

Keeping the same versions? **Delete** the two generated zips above first, then run again.

Because your `index.html` links to a zip at the repo root, also copy the repo zip to root:

```
copy repo/zips/repository.mosestyle/repository.mosestyle-1.0.0.zip  →
repository.mosestyle-1.0.0.zip
```

---

6) Push to GitHub

Push your commits.
(The repo add-on uses raw.githubusercontent.com paths, so it works even without GitHub Pages.)

---

7) (Optional) Enable GitHub Pages

GitHub → **Settings → Pages**
- **Source:** Deploy from a branch
- **Branch:** master
- **Folder:** / (root)

Kodi File Manager URL (if enabled): https://mosestyle.github.io/kodi/

---

8) Install in Kodi

1. `Settings → System → Add-ons → Unknown sources` → **ON**.
2. Install the repository zip:
   - Add a source to `https://mosestyle.github.io/kodi/` (if Pages enabled) and select `repository.mosestyle-1.0.0.zip` from the repo root, **or**
   - Download `repository.mosestyle-1.0.0.zip` locally and **Install from zip**.
3. Install from repository → **Mosestyle Kodi Repo** → **Program add-ons** → **Mosestyle Build Installer** → **Install**.
4. Open the installer (Build URL should point to `builds/mosestyle-omega-1.0.0.zip`).
5. Run installer → short reminder dialog appears:
   - Subtitles = Manual
   - Video view mode = Stretch 16:9