Mosestyle Kodi Build — Clean Guide

Checkpoint 1 (uses your uploaded code exactly)

Mosestyle Kodi Build — Clean Guide (Checkpoint 1, using your uploaded code exactly)

This is the canonical baseline (Checkpoint 1) you asked me to save. It includes the full guide and all code files verbatim.

Outcome

- Public repo URL: https://mosestyle.github.io/kodis/

- Kodi reads:

- `/repo/addons.xml`

- `/repo/addons.xml.md5`

- Add-on ZIPs served from `/zips/...`

- Build ZIP served from `/builds/...`

- First-run config service applies GUI settings on next launch (JSON-RPC only)

1) Folder layout (before building)

Create this structure in your repo working copy (names exactly as below):

```
/ (repo root)
|- _repo_generator.py                  -> from upload
|- index.html                          -> from upload
|- /builds/
|  |- mosestyle-omega-1.0.0.zip        -> from upload
|
|- /repo/
|  |- /repository.mosestyle/
|  |  |- addon.xml                     -> from upload
|  |
|  |- /plugin.program.mosestylebuild/
|  |  |- addon.xml                     -> from upload
|  |  |- default.py                    -> from upload
|  |  |- /resources/
|  |     |- settings.xml               -> from upload
|  |
|  |- /service.mosestyle.setup/        -> folder name exactly as uploaded
|     |- addon.xml                     -> from upload (id is service.mosestyle.config)
|     |- service.py                    -> from upload
```

Note: The service source folder is `service.mosestyle.setup/`, and the add-on id inside its `addon.xml` is `service.mosestyle.config`. That's fine—the generator zips by add-on id.

2) Build the repository artifacts

From the repo root:

```
python _repo_generator.py
# or on Windows if needed:
py -3 _repo_generator.py
```

The script will:

- Zip each add-on under `zips/<addon_id>/<addon_id>-<version>.zip`

- Generate `repo/addons.xml` and `repo/addons.xml.md5`

- Copy `repository.mosestyle-<version>.zip` to the repo root

- Update `index.html`

3) Commit and push to GitHub

```
git add -A
git commit -m "Publish repo: uploaded code, built zips and addons.xml"
git push
```

4) Enable/verify GitHub Pages

GitHub → mosestyle/kodis → Settings → Pages

- Source: Deploy from a branch

- Branch: the branch you pushed (e.g., `main`)

- Folder: `/` (root)

Public URL: https://mosestyle.github.io/kodis/

Quick checks in a browser (optional):

- `https://mosestyle.github.io/kodis/repo/addons.xml`

- `https://mosestyle.github.io/kodis/repo/addons.xml.md5`

- `https://mosestyle.github.io/kodis/zips/repository.mosestyle/repository.mosestyle-1.0.0.zip`

- `https://mosestyle.github.io/kodis/zips/plugin.program.mosestylebuild/plugin.program.mosestylebuild-1.0.1.zip`

- `https://mosestyle.github.io/kodis/zips/service.mosestyle.config/service.mosestyle.config-1.0.2.zip`

- `https://mosestyle.github.io/kodis/builds/mosestyle-omega-1.0.0.zip`

- `https://mosestyle.github.io/kodis/` (landing page)

5) Install on Kodi (stable flow)

1) Settings → File Manager → Add source → `https://mosestyle.github.io/kodis/`

2) Add-ons → Install from zip file → your source →

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

3) Install from repository → Mosestyle Repository → Program add-ons → Mosestyle Build → Install

4) Open Mosestyle Build and run it:

- Remove guisettings.xml = OFF (default)

- Restart when finished = OFF (default)

- Confirm install → when "Build installed…" shows, press OK

5) Exit Kodi (power icon → Exit)

6) Start Kodi again → the service runs at startup and applies your GUI settings; it shows "First-run settings applied."

(Log: `special://profile/addon_data/service.mosestyle.config/applied.json`)

6) Update later

- Edit any source file under `/repo/...`

- Bump the version in that add-on's `addon.xml`

- Rebuild with `_repo_generator.py` and push

Kodi will update from your repo (or reinstall from your repo if needed).

7) Exact file contents (verbatim from your upload)

`_repo_generator.py`

```
#!/usr/bin/env python3
"""
Mosestyle repo generator (ultra-verbose, Windows-friendly)
- Zips each add-on under ./repo/<addon_id>/
- Writes ./repo/addons.xml and ./repo/addons.xml.md5
- Copies newest repository zip to repo root
- Writes/updates ./index.html
Run from the folder that contains:  _repo_generator.py, repo\ (sources)
"""

import os, sys, io, zipfile, hashlib, shutil
```

```
from xml.etree import ElementTree as ET
from datetime import datetime

def say(msg):
    print(f"[generator] {msg}", flush=True)

ROOT = os.path.abspath(os.path.dirname(file))
REPO_SRC = os.path.join(ROOT, "repo")
ZIPS_DIR = os.path.join(ROOT, "zips")
IGNORE = {".git", ".github", "pycache", ".DS_Store", "Thumbs.db"}

def read_text(p):
    with io.open(p, "r", encoding="utf-8") as f:
        return f.read()

def write_text(p, s):
    os.makedirs(os.path.dirname(p), exist_ok=True)
    with io.open(p, "w", encoding="utf-8", newline="\n") as f:
        f.write(s)

def md5_hex(s):
    return hashlib.md5(s.encode("utf-8")).hexdigest()

def collect_addons():
    if not os.path.isdir(REPO_SRC):
        say("ERROR: ./repo/ folder not found next to _repo_generator.py")
        return []
    entries = []
    for name in sorted(os.listdir(REPO_SRC)):
        path = os.path.join(REPO_SRC, name)
        if not os.path.isdir(path) or name in IGNORE:
            continue
        ax = os.path.join(path, "addon.xml")
        if not os.path.isfile(ax):
            say(f"SKIP: {name} (no addon.xml)")
            continue
        try:
            root = ET.parse(ax).getroot()
        except Exception as e:
            say(f"ERROR parsing {ax}: {e}")
            continue
        addon_id = root.attrib.get("id")
        version  = root.attrib.get("version")
        if not addon_id or not version:
            say(f"ERROR: {ax} missing id or version")
            continue
        entries.append((addon_id, version, ax, path))
    return entries

def zip_addon(src_dir, addon_id, version):
    out_dir = os.path.join(ZIPS_DIR, addon_id)
    os.makedirs(out_dir, exist_ok=True)
    out_zip = os.path.join(out_dir, f"{addon_id}-{version}.zip")
    with zipfile.ZipFile(out_zip, "w", compression=zipfile.ZIP_DEFLATED) as z:
        for root, dirs, files in os.walk(src_dir):
            dirs[:] = [d for d in dirs if d not in IGNORE]
            for f in files:
                if f in IGNORE: continue
                ap = os.path.join(root, f)
                rp = os.path.relpath(ap, src_dir)
                z.write(ap, arcname=os.path.join(addon_id, rp))
    say(f"ZIPPED: {addon_id}-{version} -> {os.path.relpath(out_zip, ROOT)}")
    return out_zip

def build_addons_xml(paths):
    parts = []
    for p in paths:
        t = read_text(p)
        if t.lstrip().startswith("<?xml"):
            t = t[t.find("?>")+2:]
        parts.append(t.strip())
    merged = '<?xml version="1.0" encoding="UTF-8"?>\n<addons>\n' + "\n\n".join(parts) + "\n</addons>\
    return merged

def write_index_html(latest_repo_zip_name):
    html = f"""<!doctype html>
<html lang="en"><meta charset="utf-8"><title>Mosestyle Kodi Repo</title>
<meta name="viewport" content="width=device-width,initial-scale=1">
<body style="font-family:system-ui,-apple-system,Segoe UI,Roboto,Ubuntu,Arial,sans-serif;max-width:720
<h1 style="margin:0 0 8px">Mosestyle Kodi Repository</h1>
<p>Install this zip in Kodi to add the repository:</p>
<p><a href="{latest_repo_zip_name}">{latest_repo_zip_name}</a></p>
<hr>
<p><strong>Repo index:</strong> <a href="repo/addons.xml">addons.xml</a> · <a href="repo/addons.xml.md
<p><strong>ZIP folders:</strong></p>
<ul>""" + "".join(
        f'<li><a href="zips/{d}/">{d}/</a></li>' for d in sorted(os.listdir(ZIPS_DIR)) if os.path.isdi
    ) + f"""</ul>
<p><strong>Builds:</strong> <a href="builds/">builds/</a></p>
<p style="color:#888">Updated: {datetime.utcnow().strftime("%Y-%m-%d %H:%M UTC")}</p>
</body></html>"""
    write_text(os.path.join(ROOT, "index.html"), html)
    say("UPDATED: index.html")

def main():
    say(f"Python: {sys.version}")
    say(f"cwd: {os.getcwd()}")
    say(f"script dir: {ROOT}")
    if not os.path.isdir(REPO_SRC):
        say("ERROR: Missing ./repo/ folder. Create it and put your add-ons in there.")
        return
    addons = collect_addons()
    if not addons:
        say("ERROR: No valid add-ons found under ./repo/. Expect folders like repository.mosestyle/, p
        return

    addon_xmls = []
    repo_zip_candidates = []
    for addon_id, version, addon_xml, src in addons:
        out_zip = zip_addon(src, addon_id, version)
        addon_xmls.append(addon_xml)
        if addon_id == "repository.mosestyle":
            repo_zip_candidates.append((version, out_zip))

    merged = build_addons_xml(addon_xmls)
    write_text(os.path.join(ROOT, "repo", "addons.xml"), merged)
    write_text(os.path.join(ROOT, "repo", "addons.xml.md5"), md5_hex(merged))
    say("WROTE: repo/addons.xml + repo/addons.xml.md5")

    if repo_zip_candidates:
        repo_zip_candidates.sort(key=lambda x: x[0])
        latest_ver, latest_zip = repo_zip_candidates[-1]
        dest_name = f"repository.mosestyle-{latest_ver}.zip"
        shutil.copy2(latest_zip, os.path.join(ROOT, dest_name))
        say(f"COPIED: {dest_name} to repo root")
        write_index_html(dest_name)

    say("DONE. Commit & push: /repo /zips repository.mosestyle-<ver>.zip index.html (and /builds if us

if name == "main":
    main()
```

`index.html`

```
<!doctype html>
<html lang="en"><meta charset="utf-8"><title>Mosestyle Kodi Repo</title>
<meta name="viewport" content="width=device-width,initial-scale=1">
<body style="font-family:system-ui,-apple-system,Segoe UI,Roboto,Ubuntu,Arial,sans-serif;max-width:720
<h1 style="margin:0 0 8px">Mosestyle Kodi Repository</h1>
<p>Install this zip in Kodi to add the repository:</p>
<p><a href="repository.mosestyle-1.0.0.zip">repository.mosestyle-1.0.0.zip</a></p>
<hr>
<p><strong>Repo index:</strong> <a href="repo/addons.xml">addons.xml</a> · <a href="repo/addons.xml.md
<p><strong>ZIP folders:</strong></p>
<ul><li><a href="zips/plugin.program.mosestylebuild/">plugin.program.mosestylebuild/</a></li><li><a hr
<p><strong>Builds:</strong> <a href="builds/">builds/</a></p>
<p style="color:#888">Updated: 2025-09-28 17:14 UTC</p>
</body></html>
```

`repo/repository.mosestyle/addon.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<addon id="repository.mosestyle"
       name="Mosestyle Repository"
       version="1.0.0"
       provider-name="Mosestyle">
  <requires>
    <import addon="xbmc.addon" version="12.0.0"/>
  </requires>
  <extension point="xbmc.addon.repository" name="Mosestyle Repository">
    <dir>
      <info compressed="false">https://mosestyle.github.io/kodis/repo/addons.xml</info>
      <checksum>https://mosestyle.github.io/kodis/repo/addons.xml.md5</checksum>
      <datadir zip="true">https://mosestyle.github.io/kodis/zips/</datadir>
    </dir>
  </extension>
  <extension point="xbmc.addon.metadata">
    <summary>Mosestyle add-ons and build installer</summary>
    <description>Repository for Mosestyle’s installer and first-run config service.</description>
    <platform>all</platform>
  </extension>
</addon>
```

`repo/plugin.program.mosestylebuild/addon.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<addon id="plugin.program.mosestylebuild"
       name="Mosestyle Build Installer"
       version="1.0.1"
       provider-name="Mosestyle">
  <requires>
    <import addon="xbmc.python" version="3.0.0"/>
    <import addon="service.mosestyle.config" version="1.0.0"/>
  </requires>
  <extension point="xbmc.python.script" library="default.py"/>
  <extension point="xbmc.addon.metadata">
    <summary>Install Mosestyle's Kodi build</summary>
    <description>Restores the build package; the first-run service applies GUI preferences on next sta
    <platform>all</platform>
    <news>
      [1.0.1] Safer skin switch: wait for "Keep skin?" prompt to close before restart to prevent freez
    </news>
  </extension>
</addon>
```

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

```
<settings>
  <category label="Build">
    <setting id="build_url" type="text"
             label="Build ZIP URL"
             default="https://mosestyle.github.io/kodis/builds/mosestyle-omega-1.0.0.zip" />
    <setting id="fresh" type="bool"
             label="Fresh install (wipe most user data first)"
             default="false" />
    <!-- OFF by default to avoid Kodi recreating default guisettings -->
    <setting id="delete_guisettings" type="bool"
             label="Remove guisettings.xml after restore (recommended)"
             default="false" />
  </category>

  <category label="Automation">
    <setting id="enable_unknown_sources" type="bool"
             label="Enable 'Unknown sources'"
             default="true" />
    <setting id="auto_set_skin" type="bool"
             label="Auto-switch to detected skin from build"
             default="true" />
    <setting id="skin_id_override" type="text"
             label="(Optional) Force skin id (e.g. skin.fentastic)"
             default="" />
    <!-- OFF by default to prevent freeze/race on restart -->
    <setting id="auto_restart" type="bool"
             label="Restart Kodi when finished"
             default="false" />
  </category>
</settings>
```

### `repo/plugin.program.mosestylebuild/default.py`

```
# Mosestyle Build Installer — safer skin-switch + restart
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')

SERVICE_ID = "service.mosestyle.config"

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):
    if not target_ids: return
    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","v
            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):
    for aid in addon_ids:
        rpc("Addons.SetAddonEnabled", {"addonid": aid, "enabled": True})

def ensure_service_enabled():
    rpc("Addons.SetAddonEnabled", {"addonid": SERVICE_ID, "enabled": True})

def set_skin(skin_id):
    if not skin_id: return
    rpc("Addons.SetAddonEnabled", {"addonid": skin_id, "enabled": True})
    rpc("Settings.SetSettingValue", {"setting":"lookandfeel.skin","value":skin_id})

def skin_prompt_active():
    return xbmc.getCondVisibility('Window.IsActive(yesnodialog)')

def wait_for_skin_prompt(timeout=120):
    mon = xbmc.Monitor()
    end = time.time() + timeout
    saw_prompt = False
    while time.time() < end and not mon.abortRequested():
        if skin_prompt_active():
            saw_prompt = True
        else:
            if saw_prompt:
                return True  # was visible, now gone
        xbmc.sleep(250)
    return not saw_prompt  # True if never saw it

def maybe_delete_guisettings():
    if not ADDON.getSettingBool('delete_guisettings'): return
    for path in [
        xbmcvfs.translatePath('special://profile/guisettings.xml'),
        os.path.join(HOME, 'userdata', 'guisettings.xml')
    ]:
        try:
            if xbmcvfs.exists(path): xbmcvfs.delete(path)
        except Exception as e:
            log(f"delete guisettings failed: {e}")

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

    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]\
        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 da
            safe_wipe()

        extract_to_home(TMP_ZIP)

        update_local_addons_and_wait(addon_ids)
        enable_addons(addon_ids)
        ensure_service_enabled()
        maybe_delete_guisettings()

        target_skin = ""
        if ADDON.getSettingBool('auto_set_skin'):
            target_skin = ADDON.getSettingString('skin_id_override').strip() or (skin_ids[0] if skin_i
            if target_skin:
                set_skin(target_skin)
                # Wait for the "Keep skin?" prompt to be handled before any restart
                wait_for_skin_prompt(timeout=120)

        xbmcgui.Dialog().ok('Mosestyle Build',
                            "Build installed.\nOn next start, Mosestyle's First-Run Config will apply
        if ADDON.getSettingBool('auto_restart'):
            xbmc.executebuiltin('RestartApp')

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

if name == "main":
    main()
```

### `repo/service.mosestyle.setup/addon.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<addon id="service.mosestyle.config"
       name="Mosestyle First-Run Config"
       version="1.0.2"
       provider-name="Mosestyle">
  <requires>
    <import addon="xbmc.python" version="3.0.0"/>
  </requires>

  <!-- IMPORTANT: run as a background service at startup -->
  <extension point="xbmc.service" library="service.py" start="startup" />

  <extension point="xbmc.addon.metadata">
    <summary>Applies Mosestyle GUI preferences on first start after install</summary>
    <description>Runs once, applies settings via JSON-RPC, writes a log, then disables itself.</descri
    <platform>all</platform>
  </extension>
</addon>
```

### `repo/service.mosestyle.setup/service.py`

```
# Mosestyle First-Run Config (v1.0.2)
# - JSON-RPC only (no file edits) => won't reset guisettings.xml
# - Robust enum handling for refresh rate, 4:3 mode, subtitle style
# - Robust language handling (names + ISO codes)
# - Waits for GUI to be ready; applies once; logs results; disables itself

import json, time, xbmc, xbmcaddon, xbmcvfs, os

ADDON    = xbmcaddon.Addon()
ADDON_ID = ADDON.getAddonInfo("id")
DATA_DIR = xbmcvfs.translatePath(ADDON.getAddonInfo("profile"))
RUN_FLAG = os.path.join(DATA_DIR, "done.flag")
LOG_FILE = os.path.join(DATA_DIR, "applied.json")

def rpc(method, params=None):
    payload = {"jsonrpc":"2.0","id":1,"method":method}
    if params is not None: payload["params"] = params
    return json.loads(xbmc.executeJSONRPC(json.dumps(payload)))

def wait_home(timeout=60):
    mon = xbmc.Monitor(); end=time.time()+timeout
    while time.time()<end and not mon.abortRequested():
        if xbmc.getCondVisibility('Window.IsActive(home)'):
            xbmc.sleep(600)  # let skin settle
            return True
        xbmc.sleep(200)
    return False

def get_maps():
    res = rpc("Settings.GetSettings", {"level":"expert"})
    settings = res.get("result",{}).get("settings",[])
    by_id    = {s["id"]: s for s in settings if "id" in s}
    by_lbl   = {(s.get("label","") or "").strip().lower(): s["id"] for s in settings if "id" in s}
    return by_id, by_lbl

def get_current(setting):
    return rpc("Settings.GetSettingValue", {"setting": setting}).get("result",{}).get("value",None)

def set_value(setting, desired):
    """Try a few representations to fit whatever this Kodi build expects."""
    cur = get_current(setting)

    # Normalize common enums
    attempts = []

    # refresh rate: off/onstartstop/always OR 0/1/2
    if setting in ("videoplayer.adjustrefreshrate","videoscreen.adjustrefreshrate"):
        if isinstance(desired, str):
            m = {"off":0, "onstartstop":1, "always":2}
            attempts = [desired.lower(), m.get(desired.lower(), desired)]
        else:
            attempts = [desired]
    # 4:3 display mode often int enum: 0=Normal, 1=Stretch 16:9, 2=Zoom
    elif setting in ("videoplayer.displayas","videoplayer.scalingmethod43"):
        if isinstance(desired, str):
            m = {"normal":0, "stretch 16:9":1, "stretch16:9":1, "zoom":2}
            attempts = [desired, desired.lower(), m.get(desired.lower(), desired)]
        else:
            attempts = [desired]
    # GUI sound mode: 0=Never,1=Only when video,2=Always
    elif setting == "audiooutput.guisoundmode":
        m = {"never":0, "onlywhenvideo":1, "only when video playing":1, "always":2}
        attempts = [m.get(str(desired).lower(), desired)]
    # Subtitle style: 0=Normal,1=Bold,2=Italic,3=Bold Italic
    elif setting == "subtitles.style":
        m = {"normal":0,"bold":1,"italic":2,"bold italic":3,"bolditalic":3}
        attempts = [m.get(str(desired).lower(), desired)]
    # Preferred languages accept names and codes
    elif setting in ("locale.audiolanguage","locale.subtitlelanguage"):
        if isinstance(desired, str):
            attempts = [desired, desired.title(), desired.lower(), "en"]
        else:
            attempts = [desired]
    # Subtitle download languages: list of names or codes
    elif setting == "subtitles.languages":
        if isinstance(desired, (list,tuple)):
            names  = [str(x).title() for x in desired]
            lower  = [str(x).lower() for x in desired]
            iso    = []
            for x in desired:
                s=str(x).lower()
                iso.append({"english":"eng","en":"eng","swedish":"swe","sv":"swe"}.get(s, s))
            attempts = [names, lower, iso]
        else:
            attempts = [[str(desired).title()],[str(desired).lower()]]
    else:
        attempts = [desired]

    # Cast to current type when possible
    def cast_like(val, ref):
        try:
            if isinstance(ref, bool):  return bool(val)
            if isinstance(ref, int):   return int(val)
            if isinstance(ref, float): return float(val)
            if isinstance(ref, list):  return list(val) if isinstance(val,(list,tuple,set)) else [val]
            if isinstance(ref, str):   return str(val)
        except: pass
        return val

    for a in attempts:
        v = cast_like(a, cur)
        r = rpc("Settings.SetSettingValue", {"setting":setting,"value":v})
        if r.get("result") == "OK":
            return v

    # final raw try
    r = rpc("Settings.SetSettingValue", {"setting":setting,"value":desired})
    return desired if r.get("result")=="OK" else None

WANTS = [
    # Player » Videos
    (["videoplayer.seeksteps","videoscreen.seeksteps"], "skip steps", [-10,10]),
    (["videoplayer.seekdelay","videoscreen.seekdelay"], "skip delay", 750),
    (["videoplayer.adjustrefreshrate","videoscreen.adjustrefreshrate"], "adjust display refresh rate",
    (["videoplayer.syncplayback","videoplayer.synctype"], "sync playback to display", True),
    (["videoplayer.minimizeblackbars","videoplayer.zoomamount"], "minimise black bars", 20),
    (["videoplayer.displayas","videoplayer.scalingmethod43"], "display 4:3 videos as", "stretch16:9"),

    # Player » Language
    (["locale.audiolanguage"], "preferred audio language", "English"),
    (["locale.subtitlelanguage"], "preferred subtitle language", "English"),

    # Player » Subtitles
    (["subtitles.align"], "position on screen", 0),
    (["subtitles.style"], "style", "bold"),
    (["subtitles.languages"], "languages to download subtitles for", ["English","Swedish"]),
    (["subtitles.pauseonsearch"], "pause when searching for subtitles", True),
    (["subtitles.autoselect"], "auto download first subtitle", True),
    (["subtitles.tvshowsdefaultservice"], "default tv show service", "service.subtitles.a4ksubtitles")
    (["subtitles.moviesdefaultservice"], "default movie service", "service.subtitles.a4ksubtitles"),

    # System » Audio
    (["audiooutput.guisoundmode"], "play gui sounds", "never"),

    # System » Add-ons
    (["addons.unknownsources"], "unknown sources", True),
    (["addons.updatefromrepo"], "update official add-ons from", "any"),

    # Media » General
    (["filelists.showparentdiritems"], "show parent folder items", False),

    # Interface » Regional
    (["locale.region"], "region default format", "Central Europe"),
]

def main():
    # run once
    if xbmcvfs.exists(RUN_FLAG):
        return

    xbmcvfs.mkdirs(DATA_DIR)
    wait_home(60)
    by_id, by_lbl = get_maps()

    applied, skipped = {}, []

    for cands, label, value in WANTS:
        # pick first candidate that exists; else by label text
        sel = next((cid for cid in cands if cid in by_id), None)
        if not sel:
            sel = by_lbl.get(label.lower())
        if not sel:
            skipped.append({"label":label, "reason":"setting not found", "candidates":cands})
            continue
        v = set_value(sel, value)
        if v is None:
            skipped.append({"id":sel, "label":label, "wanted":value, "reason":"set failed"})
        else:
            applied[sel] = {"label":label, "value":v}

    with open(LOG_FILE, "w", encoding="utf-8") as f:
        json.dump({"applied":applied, "skipped":skipped}, f, ensure_ascii=False, indent=2)

    xbmc.executebuiltin('Notification(Mosestyle,First-run settings applied,4000)')
    # mark done & disable self
    with open(RUN_FLAG, "w") as f: f.write("ok")
    rpc("Addons.SetAddonEnabled", {"addonid":ADDON_ID, "enabled": False})

if name == "main":
    main()
```

End of Checkpoint 1 guide.