build-summary-context.py

```
#!/usr/bin/env python3
"""
build-summary-context.py
Extracts summarizer-relevant fields from a TranscriptOMatic YAML meta file,
stripping transcription-specific fields (aliases, safe, lang, deedname, etc.)
that confuse LLMs when used as summary context.

Output is plain text, structured for readability by an LLM.

Usage:
    python3 build-summary-context.py --game <slug>

    YAML resolved from <script-dir>/../meta/<slug>.yaml
    Output printed to stdout (pipe into summarize-meeting.sh)

Options:
    --game SLUG     Game slug (required)
"""

import sys
import yaml
import argparse
from pathlib import Path


def load_yaml(path):
    with open(path, encoding="utf-8") as f:
        return yaml.safe_load(f)


def format_list(value):
    if not value:
        return None
    if isinstance(value, str):
        return value.strip() or None
```

```
    if isinstance(value, list):
        items = [str(v).strip() for v in value if v]
        return ", ".join(items) if items else None
    return str(value).strip() or None


def build_context(data):
    lines = []

    # Game info
    game = data.get("game", "")
    system = data.get("system", "")
    setting = (data.get("setting") or "").strip()
    if game:
        lines.append(f"Game: {game}")
    if system:
        lines.append(f"System: {system}")
    if setting:
        lines.append(f"Setting: {setting}")
    if data.get("notes"):
        lines.append(f"Notes: {data['notes'].strip()}")
    lines.append("")

    # Characters
    characters = data.get("characters", {}) or {}
    if characters:
        lines.append("== CHARACTERS ==")
        for primary_key, entry in characters.items():
            if not isinstance(entry, dict):
                continue
            short = entry.get("short") or ""
            name_en = format_list(entry.get("name_en"))
            titles = format_list(entry.get("titles"))
            roles = format_list(entry.get("roles"))
            groups = format_list(entry.get("groups"))

            # Build display name
            display = primary_key
            if short and short != primary_key:
                display = f"{primary_key} (called: {short})"
```

```
        if name_en:
            display += f" / also known as: {name_en}"

        parts = [f"- {display}"]
        if titles:
            parts.append(f"  Titles: {titles}")
        if roles:
            parts.append(f"  Role: {roles}")
        if groups:
            parts.append(f"  Groups: {groups}")
        lines.extend(parts)
    lines.append("")

# Players and GM (out-of-character)
gm = data.get("gm", {}) or {}
players = data.get("players", {}) or {}
ooc = {}
if isinstance(gm, dict):
    ooc.update(gm)
if isinstance(players, dict):
    ooc.update(players)
if ooc:
    lines.append("== PLAYERS (out-of-character) ==")
    for name in ooc.keys():
        lines.append(f"- {name}")
    lines.append("")

# Groups
groups = data.get("groups", {}) or {}
if groups:
    lines.append("== GROUPS & ORGANISATIONS ==")
    for name, entry in groups.items():
        if not isinstance(entry, dict):
            lines.append(f"- {name}")
            continue
        desc = (entry.get("description") or "").strip().replace("\n", " ")
        if desc:
            lines.append(f"- {name}: {desc}")
        else:
            lines.append(f"- {name}")
```

```
        lines.append("")

    # Locations
    locations = data.get("locations", {}) or {}
    if locations:
        lines.append("== LOCATIONS ==")
        for name, entry in locations.items():
            if not isinstance(entry, dict):
                lines.append(f"- {name}")
                continue
            desc = (entry.get("description") or "").strip().replace("\n", " ")
            sig = (entry.get("significance") or "").strip()
            detail = " — ".join(filter(None, [desc, sig]))
            if detail:
                lines.append(f"- {name}: {detail}")
            else:
                lines.append(f"- {name}")
        lines.append("")

    # Terms (descriptions only, no aliases)
    terms = data.get("terms", {}) or {}
    if terms:
        lines.append("== TERMS ==")
        for name, entry in terms.items():
            if not isinstance(entry, dict):
                lines.append(f"- {name}")
                continue
            desc = (entry.get("description") or "").strip().replace("\n", " ")
            if desc:
                lines.append(f"- {name}: {desc}")
            else:
                lines.append(f"- {name}")
        lines.append("")

    return "\n".join(lines)


def main():
    script_dir = Path(__file__).resolve().parent
    meta_dir = (script_dir / ".." / "meta").resolve()
```

```
    parser = argparse.ArgumentParser(
        description="Generate a filtered summary context from a YAML meta file."
    )
    parser.add_argument("--game", required=True, metavar="SLUG",
                        help="Game slug — resolves to meta/<slug>.yaml")
    args = parser.parse_args()

    yaml_path = meta_dir / f"{args.game}.yaml"
    if not yaml_path.exists():
        print(f"□ YAML not found: {yaml_path}", file=sys.stderr)
        sys.exit(1)

    data = load_yaml(yaml_path)
    print(build_context(data))


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

Created 2026-04-14 14:13:59 UTC by Mela
Updated 2026-04-14 14:14:20 UTC by Mela