

THE PRO-PACK PIPELINE SPECIFICATION

AUTOMATED AGENT WRAPPER ENGINE — REFERENCE SPECIFICATION v2.2

Notice: This document outlines an automated wrapper specification for developers engineering client-side pipeline code. It is a reference model for conditional orchestration around a probabilistic editor, not drop-in production code. Map parameters directly to the host specifications you call.

1. CORE PIPELINE ARCHITECTURE & DATA SCHEMA

This client-side code wrapper architecture enforces conditional logic and local disk sandboxing to govern sequential execution states across the Seedance 2.5 editing API. The client application wrapper dynamically interpolates runtime variables into the stored literal prompt templates before API transmission.

Table 1.1: Runtime Data Schema Matrix

Field Name	Type	Description / Runtime Variable Constraints
id	string	Unique alphanumeric step identifier for execution logging.
execution_condition	bool	Precomputed boolean flag derived by client script before loop.
disclaimer_text	string	Optional string containing raw compliance text tracking characters.
source_tag	string	Explicit reference label anchoring input master layer.
source_file	path	Absolute path to localized, numbered working copy file on disk.
references	list	Array defining multi-part asset bindings: [{"tag": "@Image1", "path": "p.png"}].
timestamp	string	Localized temporal constraint defining execution window.
mutation	string	Explicit string filled directly into the template text payload.
template	string	Immutable template file configuration stored as a raw string literal on disk.

Table 1.2: Core Specification Preservation Scope Constants

Implementers must ensure that the literal string value passed to the `preservation_scope` tracking variable maps exactly to:

`"identity, wardrobe, product shape, hands, camera move, lighting, motion, duration"`.

2. EXPLICIT API PROMPT REGISTRY

Note: There is no Prompt 3 inside this schema framework. IDs remain hard-coded to 1, 2, 4, 5 to ensure existing automation wrapper dictionary keys do not experience break crashes.

Prompt 1 (Frame-0 Fix)

Edit @Video1. @Video1 is the sole master. Change only frame 0 through the first cut so the opening composition matches this reference still: @Image1. Keep identity, wardrobe, product shape, hands, camera move, lighting, motion, and duration unchanged. Do not add frames before the story begins.

Prompt 2 (Product Lock Fix)

Edit @Video1. @Video1 is the sole master. {timestamp}, replace only the product to precisely match this reference asset: @Image1. Keep identity, wardrobe, product shape, hands, camera move, lighting, motion, and duration unchanged.

Prompt 4 (Object Erasure Fix)

Edit @Video1. @Video1 is the sole master. {timestamp}, {mutation}. Keep identity, wardrobe, product shape, hands, camera move, lighting, motion, and duration unchanged. Do not crop, blur, or rebuild the rest of the frame canvas.

Prompt 5 (Text Overlay Fix)

Edit @Video1. @Video1 is the sole master. {timestamp}, add this exact line as a lower-third caption, character for character, with no extra punctuation: {disclaimer_text}. Use white bold type with a dark outline. Do not cover the product or the face. Keep identity, wardrobe, product shape, hands, camera move, lighting, motion, and duration unchanged.

3. CLIENT-SIDE PIPELINE ENGINE REFERENCE IMPLEMENTATION

Note: System parameters and field schemas within this reference block are illustrative. Map them directly to the production endpoints you call.

python

```
"""
DEVELOPER REVISION REFERENCE SPECIFICATION v2.2
DESIGN INTENT: CLIENT-SIDE PIPELINE ORCHESTRATION FOR SEEDANCE 2.5 API RUNS
"""

import os
import sys
import json
import hashlib
import subprocess
import logging
import requests

class PipelineException(Exception): pass
class APIException(Exception): pass

def calculate_sha256(file_path: str) -> str:
    hasher = hashlib.sha256()
    with open(file_path, 'rb') as f:
        for chunk in iter(lambda: f.read(65536), b''):
            hasher.update(chunk)
    return hasher.hexdigest()

def execute_client_side_mute(source_path: str, output_path: str):
    if not os.path.exists(source_path):
        raise PipelineException("Local Ingest Failure: Source file missing from disk node.")
    command_args = ["ffmpeg", "-y", "-i", source_path, "-an", "-vcodec",
"copy", output_path, "-loglevel", "quiet"]
    subprocess.run(command_args, check=True)

def submit_to_seedance_api(endpoint_url: str, bearer_token: str,
master_video_path: str, references: list, prompt_text: str) -> bytes:
    headers = {"Authorization": f"Bearer {bearer_token}"}
    payload_data = {"task_type": "video_editing", "prompt": prompt_text}

    with open(master_video_path, "rb") as video_file:
        files_payload = [("master_video", (os.path.basename(master_video_path),
video_file, "video/mp4"))]
        open_handles = []
        try:
```

```

        for ref in references:
            f_handle = open(ref["path"], "rb")
            open_handles.append(f_handle)
            files_payload.append(("reference_images",
(os.path.basename(ref["path"]), f_handle, "image/png")))

        response = requests.post(endpoint_url, headers=headers,
data=payload_data, files=files_payload, timeout=300)
        response.raise_for_status()
        return response.content
    except requests.RequestException as network_err:
        raise APIException(f"Outbound API Multi-Part Submission Failed:
{str(network_err)}")
    finally:
        for h in open_handles: h.close()

def run_orchestration_pipeline(pipeline_schema: list, environment_paths: dict,
expected_hash: str, api_credentials: dict):
    logging.basicConfig(level=logging.INFO, format='[WRAPPER LOG] %(message)s')
    baseline_file = environment_paths["master_baseline"]
    workspace_dir = environment_paths["workspace"]

    current_hash = calculate_sha256(baseline_file)
    if current_hash != expected_hash:
        raise PipelineException(f"Ingest Rejection: Asset file mutation
detected. Found: {current_hash}")

    current_source = baseline_file
    executed_step_counter = 0

    for step in pipeline_schema:
        step_id = step["id"]
        if not step["execution_condition"] or (step_id == "prompt_5_caption"
and not step.get("disclaimer_text", "")):
            logging.info(f"Skipped step: '{step_id}' due to condition or empty
text bounds.")
            continue

        executed_step_counter += 1
        raw_template = step["template"]
        compiled_prompt = raw_template.format(
            timestamp=step.get("timestamp", ""),
            mutation=step.get("mutation", ""),
            disclaimer_text=step.get("disclaimer_text", "")
        )

        working_output_path = os.path.join(workspace_dir,
f"Project_V{executed_step_counter:02d}_edit.mp4")
        temp_api_output = working_output_path + ".tmp"

        try:
            raw_video_bytes = submit_to_seedance_api(
                endpoint_url=api_credentials["url"],

```

```

        bearer_token=api_credentials["token"],
        master_video_path=current_source,
        references=step["references"],
        prompt_text=compiled_prompt
    )

    with open(temp_api_output, "wb") as f:
        f.write(raw_video_bytes)

    file_exists = os.path.exists(temp_api_output)
    file_has_size = os.path.getsize(temp_api_output) > 0 if file_exists
else False

    if not file_exists or not file_has_size:
        logging.warning(f"API error: Server output corrupted for
'{step_id}'. Discarding and keeping prior state.")
        if os.path.exists(temp_api_output):
            os.remove(temp_api_output)
            executed_step_counter -= 1
            continue

    os.rename(temp_api_output, working_output_path)
    current_source = working_output_path
    logging.info(f"Wrote file successfully: {working_output_path}")

except APIException as api_err:
    logging.error(f"API error caught on step block '{step_id}':
{str(api_err)}")
    raise PipelineException("Pipeline Execution Suspended due to core
network drops.")

    if environment_paths.get("execute_post_mute_flag", False):
        final_silent_output = os.path.join(workspace_dir,
"Project_Final_Silent_Master.mp4")
        execute_client_side_mute(current_source, final_silent_output)
        current_source = final_silent_output
        logging.info(f"Local client mute pass completed:
{final_silent_output}")

    return current_source

```
