SOULWIRE STUDIO

AI AUTOMATION FOR OPERATORS

The Content Engine Playbook

The prompts that wire it right the first time. Half filmed, half AI-generated.

A hands-off short-form pipeline that runs two lanes at once: videos the AI generates for you, and videos you film yourself. You build it once, then it sources, scripts, edits, and posts on a weekly rhythm while you just film and approve.

The full build, generalized so you can run it in your own business.

SECTION I

What You're Building

A hands-off short-form pipeline that runs two lanes at once: videos the AI generates for you, and videos you film yourself. You build it once, then it sources, scripts, edits, and posts on a weekly rhythm while you just film and approve. Each stage below is one prompt, written to paste straight into your AI builder, with the landmines baked in as guardrails. Swap any tool for your own; the wiring is what carries over.

SECTION II

What You'll Need

01 **Airtable**

The record of every post: status, script, keyword, and the finished video URL.

02 **Claude**

Writes the scripts, and if you like, wires the build itself.

03 **Higgsfield**

Generates the faceless AI-lane videos. Any text-to-video tool can stand in.

04 **Submagic**

Edits the clips you film: b-roll, captions, zooms, clean audio.

05 **Google Drive**

The drop folder you film into, and the source library you pull from.

06 **Telegram**

Where each finished clip lands for a one-tap approve or decline.

07 **Blotato**

Posts approved clips out to your socials.

08 n8n

Holds the wiring together and runs it on a schedule.

SECTION III

The Build

01 Start here: the architecture brief

Most bad builds come from a model that only sees one node at a time. Set the full picture and the two rules that govern everything, then build in sequence.

> THE RULE
>
> Establish the map and the money rules up front. A model that knows the destination wires each node to fit it.

COPY THIS:

```
I'm wiring a hands-off short-form content pipeline in n8n. Before we build any single piece, hold the whole shape in mind. We build one stage at a time: source and script, split into two lanes, the AI lane, the live lane, review in chat, and publish.

Stack: Airtable is the record of every post. Claude writes scripts. Higgsfield generates the faceless AI videos. Submagic edits the clips I film. Google Drive is the drop folder. Telegram is where I review. Blotato posts to my socials.

Two rules for the entire build:
- Never re-render or re-generate anything already produced. It costs real API money. When testing, read state; do not re-run render steps.
- Every paid API bills from its own wallet, separate from any web-app subscription. Before calling a failed job a bug, confirm that API wallet is funded.

Acknowledge this architecture back to me, then we start at stage one.
```

02 Source and script

The engine is only as good as its raw material and its rules. Point it at your real library, then hold the script to numbers that actually perform.

WHY THIS WORKS

The hook earns the first six seconds and the word count keeps you at 45 to 55 spoken. Spec beats taste when a machine writes at volume.

GOOD TO KNOW

Dedupe on angle, not just topic. One source can feed a dozen posts if each one opens on a genuinely different idea.

COPY THIS:

```
Build stage 1: source and script.

Sources: have Claude scan [my Google Drive folder of existing material] plus [the websites my niche reads] and [the forums my audience lives in]. Pull from files that already exist, not just new uploads.

For each source, extract up to 3 distinct angles and write one script per angle. Enforce these in the prompt:
- Script 110 to 135 words (45 to 55 seconds spoken). Never exceed 140.
- Hook: max 8 words. A bold claim or curiosity gap that reads as on-screen text and earns the first 6 seconds. Not a sentence, not clickbait.
- Dedupe against angles already used for that source.

Write each result to Airtable as a row with status = scripted. Cap the active queue so the system never overproduces.
```

03 Split into two lanes

This is the heart of the half-and-half system. Sort each post by content, hold a weekly ratio, and hand yourself a filename you never have to invent.

THE RULE

The machine assigns the keyword, not the human. A person guessing filenames is the single most common way this pipeline breaks.

COPY THIS:

```
Build stage 2: split every week into two lanes.

AI lane: faceless, no person on camera, generated automatically.
Live lane: talking-head, I film myself.

Classify: audience-question, first-person, opinion, or hot take goes to the live lane.
Historical, atmospheric, concept-explanation, or list goes to the AI lane. When unsure,
AI lane.

Enforce a weekly ratio of [your split, for example 2 AI and 3 live] with a hard cap,
topping each lane up based on what is already queued.

For every live-lane row, auto-generate a short, filename-safe keyword (for example
topic-slug-a1b2) and save it on the record. I will name my video file exactly that
keyword so the system can match it later. Never make me name files by hand.
```

04 The AI lane: the half you never touch

These posts write, render, caption, and queue themselves. Your only job is to keep the credits topped up and approve the result.

> GOOD TO KNOW
>
> Any text-to-video generator drops in here. What matters is the guard around it: render once, check credits first, honor the cap.

COPY THIS:

```
Build stage 3: the AI (faceless) lane.

For every AI-lane row, automatically produce a faceless video with no person on camera.
Send the script to Higgsfield to render the visuals, add a voiceover from the script,
and burn in captions. Save the finished video URL back to the row and move it to review.

Rules:
- Render only rows that have never been rendered. Re-rendering a finished video is pure
wasted spend.
- Check Higgsfield's credit balance before submitting. If it is short, leave the row
queued for the next run instead of failing.
- Respect the weekly cap from stage 2. Don't drain credits making more than you'll post.
```

05 The live lane: drop a file, it gets dressed

You film in your window, drop the clips in one folder, and the system does the rest: match, edit, save. The care is all in handling more than one at a time.

THE RULE

A folder poll hands you a batch, not one file. If any node assumes a single item, every clip after the first vanishes silently.

GOOD TO KNOW

The filename is a contract. Let the machine name it and you never think about this again.

COPY THIS:

```
Build stage 4: intake and edit the clips I film.

Watch one Google Drive folder. When a video lands, take the filename minus its extension
as the keyword, match it to the queued row, and send the video to Submagic to add
b-roll, captions, zooms, clean audio, and remove bad takes. Save the finished video URL
back to the row.

Wire these correctly the first time:
- Share the folder "anyone with the link" so Submagic can fetch the file.
- One folder poll returns all new files at once. Every downstream step must handle N
files in a single run, not just one.
- Filename to keyword must match exactly. A space vs a hyphen, or a "(1)" suffix, breaks
the match.
- Poll only during my working window (for example twice on my filming evening) to save
executions.
```

06 Review each clip in chat

Both lanes land here. The review step is where most builds quietly fail on a batch. The fix is to stop waiting for one reply and start listening for taps.

THE RULE

Send-and-wait is one-at-a-time by design. Drop five clips at once and four are stranded. Use callback buttons plus a listener for real batch review.

GOOD TO KNOW

A decline shouldn't be a dead end. Send it back to scripted so it re-enters the queue instead of rotting as a rejected row.

COPY THIS:

```
Build stage 5: let me review each clip in Telegram.

Send each finished clip as its own message with a Watch link and its own Approve and Decline buttons. I approve or decline each one independently, in any order.

Do not use a "send and wait for one reply" step. It handles only one item and silently strands the rest of a batch. Instead:
- Put callback buttons on each message carrying that row's id.
- Build a separate listener that catches the taps and updates each row.
- Approve sets status = approved.
- Decline sends it back to the queue (status = scripted) so I redo it. Don't leave dead rejected rows lying around.

Telegram parses message text as Markdown, so a value with an underscore (like youtube_shorts) throws "can't parse entities". Strip Markdown characters from any dynamic text, or send in plain or HTML mode.
```

07 Publish to the right brand

If you run more than one brand or channel through one publisher, the default behavior will post to the wrong one. Name the target explicitly and it never happens.

THE RULE

First account per platform is a coin flip on a multi-brand publisher. Match the account by name, or you post one brand's clip to another brand's channel.

COPY THIS:

```
Build stage 6: publish approved clips through Blotato to [my platforms, for example TikTok and YouTube].

Read this before you pick an account: my Blotato account holds profiles for several brands and channels. Do not select "the first account per platform"; that posts to the wrong one. Select the account whose name matches [this brand's name or handle] and post only to those. Skip any platform with no matching account.

Build the caption as caption, then a blank line, then hashtags. Post one approved clip per run on a daily drip, then mark it scheduled so it never double-posts.
```

08

The guardrails

Eight landmines, each one an hour to learn. Paste them into the build as standing rules and they cost you nothing.

1. Separate API wallets. A failed render is usually an empty API wallet, not a bug. Fund it, then debug.
2. Never re-render finished work. Verify by reading status, not by re-running the expensive step.
3. Paused runs are frozen in time. A waiting run uses its start-time snapshot; a fix you make now won't reach it. Start a fresh run to test.
4. Assume batches, not items. One trigger can carry many records; if a step assumes a single item, everything after the first disappears silently.
5. Pick accounts by name. On a multi-brand publisher, never take the first account. Match to the brand explicitly.
6. Pin schedules to one clock. If your platform runs in UTC, setting both a workflow timezone and a local time double-shifts every trigger. Convert once.
7. The filename is a contract. Keyword matching is exact; a space, a hyphen, or a "(1)" breaks it. Let the system name files.
8. Sanitize chat text. Underscores and asterisks break Markdown parsers mid-send. Clean dynamic text or send plain.

Rather have it built for you?

This guide documents one system from the Soulwire Studio workflow library. We build AI automations for businesses that run on process: intake, scheduling, content, compliance-heavy operations. If you would rather spend your time filming and approving than building pipelines, book a workflow audit and we will find the automation hiding in your business.

soulwirestudio.com · Book a workflow audit