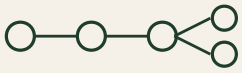


THE MISSING MANUAL FOR MAKE

The Builder's Companion Kit

The desk reference, the production checklist, and every scenario in the book — ready to import and keep next to your keyboard.

Companion resource for readers of
The Missing Manual for Make · **Brian Kasday**

MMS VEGAS · READER COMPANION KIT

Version 1.0 · First Edition · May 2026

FIRST — CHECK YOUR DOWNLOAD

What's in This Kit

The Companion Kit is a package, not a single file. Here's everything it contains. If any piece is missing, email `brian@mmsvegas.com` and I'll get it to you.

YOUR DOWNLOAD INCLUDES

- ☐ `The_Make_Builders_Companion_Kit.pdf` **THIS FILE**
  The desk reference, checklists, worksheet, and blueprint guide.
- ■ `01–06 · scenario blueprints (.json)` **BY EMAIL**
  The book's six worked examples, ready to import into your own Make account.
- ■ `Operations_Cost_Estimator.xlsx` **BY EMAIL**
  Estimate a scenario's monthly credit burn before you switch it on.

Why two of these arrive by email. The scenarios and the estimator are the working tools — they land in your inbox when you confirm your address, so a real one reaches you and you have a permanent copy you can't lose in a downloads folder. Everything in this PDF is yours to keep and print right now.

FIND YOUR MOMENT

Start Here

You won't read this front to back. Jump to whatever matches what you're doing right now.

IF YOU'RE…	GO TO
Planning a new scenario	Scenario Planning Worksheet
About to activate one	Production Readiness Checklist
Debugging a failure	Card 4 — Errors & the Five Handlers
Wrestling with data or arrays	Card 2 — Functions + Card 5 — Flow Control
Formatting a date	Card 3 — Date & Time Tokens
Worried about cost	The Limits card + the Operations Cost Estimator
Following a worked example	The Scenario Blueprints
Wondering if a detail has gone stale	Keep It Current (last page)

Bookmarks are enabled in this PDF — open the bookmark panel in your reader to jump to any card.

What this kit is

The book teaches you Make. This kit is the part of the book you can't keep open on a second monitor while you build — the numbers you look up, the functions you half-remember, the checklist you run before you flip a scenario live, and the scenarios from the worked examples, packaged as files you can import straight into your own account.

Nothing here replaces a chapter. It's the working layer that sits beside the reading — pull a card when you're building, run the checklist when you're shipping, import a blueprint when you'd rather adapt a working scenario than rebuild it from a screenshot.

WHAT'S INSIDE

- **Five reference cards** — the limits that bite, a functions cheat sheet, the date/time format tokens, the error types and the five handlers, and a flow-control decision guide.
- **The Production Readiness Checklist** — the book's Appendix B, formatted to print and pin next to your scenario list.
- **The scenario blueprints** — the book's worked examples as importable `.json` files, so you start from a working scenario instead of a blank canvas.

HOW TO USE IT

1. Print the reference cards and the checklist. They're built for paper and a pinboard, not a tab you'll lose.
2. When you reach a worked example in the book, import its blueprint (next-to-last page tells you how) and follow along in your own account.
3. Before any scenario goes from "I tested it" to "the business depends on it," walk the checklist. Every item.

A note on currency: Make ships changes constantly. This kit is current as of the book's edition — when Make changes something that matters, the corrected card comes to your inbox. That's the point of being on the list.

1

REFERENCE CARD

The Limits That Bite

The hard ceilings you'll hit in production. None of these are "best practice" suggestions — they're enforced by the platform. Know them before you design around them.

LIMIT	VALUE
Single-module timeout	**40 seconds**
Whole-scenario max execution	**40 minutes**
Sleep module — maximum delay	**300 sec (5 min)**
Data per module, per run	**5 MB**
Search module — result cap	**3,200 objects**
Data store — minimum / default size	**1 MB**
Data stores per organization	**1,000 max**
Free plan — monthly credits	**1,000**
Free plan — active scenarios	**2**
Free plan — minimum scheduling interval	**15 minutes**
Paid plan — minimum polling interval	**1 minute**

Don't conflate the two timeouts. The 40-second limit is per module run; the 40-minute limit is the whole scenario. They're different mechanisms with different fixes — split long work into a webhook-triggered subscenario for the first, break a giant batch into scheduled chunks for the second.

Data store space scales separately from count. You can have up to 1,000 stores, but total storage space grows with your plan (roughly 1 MB per 1,000 credits). The "wall" people hit is usually space, not the store count.

2

REFERENCE CARD

Functions Cheat Sheet

The functions you reach for most, by group. Full reference is Chapter 8 and Appendix D — this is the working subset. Arguments separated by semicolons; `[brackets]` mark optional ones.

GENERAL & LOGIC

`if(expr; val1; val2)`	Returns val1 if expr is true, else val2.
`ifempty(val1; val2)`	val1 unless it's empty — then val2. The safe-default workhorse.
`switch(expr; v1; r1; …; else)`	Maps a value to a result; final unpaired arg is the fallback.
`get(obj_or_array; path)`	Pulls a value by path — the way to reach into dynamic structures.

TEXT

`trim(text)`	Strips leading/trailing whitespace.
`replace(text; search; repl)`	Find-and-replace; accepts regex in the search slot.
`split(text; separator)`	Text to array. Pair with map/join.
`substring(text; start; end)`	Slice by character position.
`lower / upper / capitalize`	Case control. `startcase` for Title Case.
`contains(text; search)`	True/false substring test — common in filters.

ARRAYS

`map(array; key; [filterKey]; [vals])`	Extract one field from a complex array — the most-used array function by far.
`join(array; separator)`	Array to a single delimited string.
`deduplicate(array) / distinct`	Remove repeats; `distinct` can dedupe by key.
`sort(array; [order]; [key])`	Order an array, optionally by a nested key.
`length(array) / first / last`	Count and end-grabs.

DATE & TIME — tokens on Card 3

`formatDate(date; format; [tz])`	Render a date to text. Render / parse tokens on Card 3.
`parseDate(text; format; [tz])`	Text to date — give it the exact format it's in.
`addDays / addHours / setDay …`	Date math; full mirror set of add* and set* helpers.

Idiom worth memorizing: `ifempty(get(…; path); "fallback")` — reach into a maybe-missing field and never let a blank crash the next module.

CARD 2 · CONTINUED — PATTERNS YOU'LL REUSE

Functions answer "what exists." These answer "how do I actually do the thing." Copy, swap the field names, drop into a mapping.

Safe fallback — never let a blank break the next module

```
ifempty(get(1.data; "email"); "none@example.com")
```

Normalize an email before matching or storing

```
lower(trim(1.email))
```

Turn an array of objects into a readable list

```
join(map(1.array; "name"); ", ")
```

Keyword test, case-insensitive (great in a filter)

```
contains(lower(1.subject); "refund")
```

API-ready timestamp (ISO 8601)

```
formatDate(now; "YYYY-MM-DDTHH:mm:ssZ")
```

De-dupe a list by a key field

```
distinct(1.records; "customer_id")
```

First non-empty of several sources

```
ifempty(1.nickname; ifempty(1.first_name; "there"))
```

Count items, then branch on it in a filter

`length(1.line_items)` → filter: `greater than 0`

These map straight onto the worked examples — the CSV-to-Airtable and Shopify blueprints lean on `map`, `distinct`, and the safe-fallback idiom throughout.

3

REFERENCE CARD

Date & Time Format Tokens

These go in the `format` argument of `formatDate()` and `parseDate()`. Case matters — `M` is month, `m` is minute.

TOKEN	EXAMPLE	MEANING
`YYYY / YY`	2026 / 26	4- or 2-digit year
`Q`	1–4	Quarter of year
`M / MM`	3 / 03	Month number (MM zero-padded)
`MMM / MMMM`	Mar / March	Month abbreviation / full name
`D / DD`	5 / 05	Day of month (DD zero-padded)
`Do`	5th	Day of month with ordinal
`DDD`	1–365	Day of year
`ddd / dddd`	Mon / Monday	Day name
`W / WW`	1–53	ISO week of year
`E`	1–7	ISO day of week (Mon=1, Sun=7)
`H / HH`	9 / 09	Hour, 24-hour
`h / hh`	9	Hour, 12-hour (use with a / A)
`mm`	05	Minute, zero-padded
`ss`	09	Second, zero-padded
`a / A`	am / PM	Meridiem, lower / upper
`X / x`	1410715640 / …579	Unix timestamp (seconds / milliseconds)

Common patterns: `YYYY-MM-DD` (sortable date), `YYYY-MM-DDTHH:mm:ssZ` (ISO 8601 for APIs), `dddd, MMMM Do` (human-readable, e.g. "Monday, March 5th").

4

REFERENCE CARD

Errors & the Five Handlers

When a module errors, which handler you attach decides whether the scenario survives. Run the decision in order — stop at the first match.

Rollback

When partial work is worse than none. ACID-aware modules (databases, some CRMs) where "47 of 50 succeeded" leaves data inconsistent. Reverts committed changes. The default when incomplete executions are off.

Commit

When you want to keep what worked and stop cleanly. Best-effort batch work where partial success is fine and you don't want incomplete-execution noise.

Resume

When there's a real fallback value. Enrichment lookup fails → substitute a placeholder and continue. Not a tool for making errors disappear — that's silent corruption.

Ignore

When the bundle genuinely doesn't matter. Deleting an already-deleted record (404). Always pair with a filter on the specific error type — a naked Ignore is silent failure waiting to happen.

Break — the default

When none of the above apply, i.e. almost always. Pair with retry attempts: 3, delay: 15 min, for transient failures. Unrecoverable bundles wait in the Incomplete Executions queue for human review.

The production default that handles 90% of cases: Break + automatic retry + "Store incomplete executions" on at the scenario level. Reach for the others only when the situation specifically calls for it.

CARD 4 · CONTINUED — THE ERROR TYPES YOU'LL ACTUALLY SEE

ERROR	WHAT IT MEANS
`RateLimitError`	Too many requests to the third party (HTTP 429). Retry with delay.
`ConnectionError`	Third-party app unreachable (502/503/504). Transient — retry.
`ModuleTimeoutError`	No response inside the timeout window (40–60 sec).
`AccountValidationError`	Couldn't authenticate with stored credentials (401/403). Reconnect.
`DataError`	Data you sent failed validation on the third-party side.
`BundleValidationError`	A bundle failed type-checking — missing field or wrong type.
`IncompleteDataError`	Module retrieved only part of the expected data.
`DuplicateDataError`	Tried to create a record that must be unique and already exists.
`MaxFileSizeExceededError`	A file exceeds your plan's max file size.
`DataSizeLimitExceededError`	Organization ran out of data-transfer quota.
`OperationsLimitExceededError`	Out of credits for the billing cycle.
`InconsistencyError`	Rollback failed — another scenario already changed the same data store.
`RuntimeError`	Third-party error that fits no other category. Read the message.

Filter your error handlers by type. A handler that catches `RateLimitError` and retries is right; the same handler swallowing a `DataError` hides a real bug.

5

REFERENCE CARD

Which Flow-Control Module?

The four that confuse people most. Pick by what you're trying to do to the flow, not by what's nearest in the toolbar.

Filter

Drop bundles that don't qualify — the cheap one. Sits on the line between two modules. Put it as early as possible: every operation upstream of a filter is wasted on bundles that get dropped.

Router

Send the flow down two or more branches. Each route has its own filter; bundles take every route they qualify for. Use for "if this, do A; otherwise do B" branching.

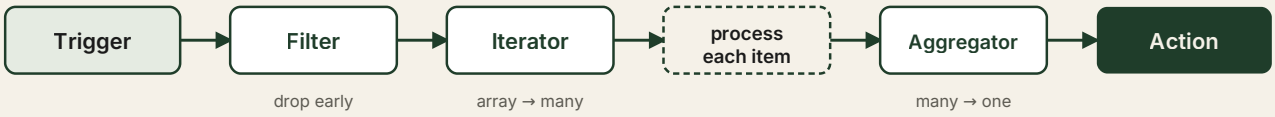
Iterator

Turn one bundle containing an array into many bundles. The split step — process each item in a list separately downstream.

Aggregator

Collapse many bundles back into one. The opposite of an Iterator. Array Aggregator for a structured array; Text Aggregator to join into a string. The custom-output Array aggregator is the pattern you'll reuse most.

The shape you'll build over and over:



If you find yourself reaching for a Router with many near-identical branches, you usually want a Filter plus a single path, or a `switch()` in your mapping instead. Routers multiply maintenance.

CARD 5 · CONTINUED — THE FOUR TRAPS

Most flow-control pain comes from reaching for the wrong tool out of habit. The fixes:

IF YOU'RE TEMPTED TO…	DO THIS INSTEAD
Build a dozen near-identical Router branches	One path + `switch()` in the mapping
Process every row, then filter later	Filter first — every upstream op on a dropped bundle is wasted
Write records one at a time in a loop	Aggregate, then bulk-write in one module
Add an Iterator just because there's an array	Only iterate if downstream work must happen per item

Rule of thumb: an Iterator and an Aggregator should almost always come as a pair. A scenario that splits an array but never recombines it is usually doing more operations than it needs to.

BEFORE YOU BUILD

Scenario Planning Worksheet

Ten minutes here saves an afternoon of rework. Answer these before you place the first module — most production failures trace back to a question nobody asked at the start. Print one per scenario.

SCENARIO NAME _____ OWNER _____

BUSINESS PURPOSE — WHAT DOES THIS MAKE HAPPEN?

WHAT COUNTS AS SUCCESS?

TRIGGER — WHAT STARTS IT? _____ HOW OFTEN / EXPECTED VOLUME _____

SOURCE SYSTEMS _____ DESTINATION SYSTEMS _____

WHAT CAN GO WRONG? (THE TOP TWO FAILURE MODES)

ERROR & RETRY PLAN (WHICH HANDLER — SEE CARD 4)

WHAT DATA MUST NEVER BE LOST?

TEST DATA SOURCE _____ GO-LIVE DATE _____

ROLLBACK / RECOVERY PLAN IF IT BREAKS

When this sheet is filled in, the Production Readiness Checklist (next) becomes a five-minute confirmation instead of a redesign.

✓

THE SINGLE MOST USEFUL PAGE IN THE BOOK

Production Readiness Checklist

A scenario that "works in testing" and one that "works in production" are not the same thing. Before any scenario goes from "I built it and tested it" to "the business depends on it," walk these items. Within each group, the first item matters most.

1 • Naming and discoverability

- ☐ The scenario has a **descriptive name** — not "Untitled," not "Copy of Copy of Lead Sync." Future-you has to find it under pressure.
- ☐ It has a **documented owner** — a custom property, the description field, or your docs.
- ☐ It lives in the **right team** — teams are scope boundaries for connections and data stores, not just folders.

2 • Test data separation

- ☐ Production scenarios run on **production data only** — no "test_" rows still flowing into the real systems.
- ☐ A **separate test version** exists for changes — clone before editing anything live and non-trivial.

3 • Error handling

- ☐ Every module calling an **external service has an error handler** — default to Break, retry 3, delay 15 min.
- ☐ **"Store incomplete executions" is on** at the scenario level. Without it, errored runs are discarded.
- ☐ **Error notifications reach a human** — built-in error emails or a dedicated alert path.

4 • Cost and performance

- ☐ The scenario's **per-run operation cost is known and acceptable** — run once, read the history, multiply by expected volume. (The estimator in this kit does the math.)
- ☐ **Filters appear early, not late.** Every operation upstream of a filter is wasted on dropped bundles.
- ☐ **Search modules have explicit Limit values** — defaults can run to the 3,200-record / 5 MB cap.

5 • Connections and credentials

- ☐ Every connection **verified within 90 days** — OAuth tokens lapse and passwords rotate silently.
- ☐ Connections **named for what they do**, not who made them — "Gmail – Notifications inbox," not "John's Gmail."
- ☐ Automations use **service-account connections**, not individuals' — so they don't break the day someone leaves.

6 • Recovery and continuity

- ☐ The **blueprint is exported and stored outside Make** — three-dot menu → Export Blueprint → into version control.
- ☐ A **documented recovery process** exists — what happens if this is down for an hour? A day? Forever?

- [ ] **Monitoring or a scheduled health-check** exists — built-in error/deactivation alerts or an external monitor.

Bonus — situational

- [ ] Third-party **rate limits documented and respected** (e.g. Airtable 5 req/sec per base).
- [ ] Critical webhooks have **backup polling reconciliation** — webhooks drop during outages.
- [ ] Multi-tenant scenarios verify **customer data isolation**; sensitive data is sanitized in logs.

Print this page. Stick it next to your scenario list. The discipline of running through it before activating anything is the single largest difference between scenarios that quietly work and scenarios that quietly break.

ACTIVATION SIGN-OFF

SCENARIO _____ REVIEWED BY _____ DATE _____

RISK RATING (Low) (Medium) (High)

NOTES _____

DO NOT ACTIVATE UNTIL

Every external-call module has an error handler · "Store incomplete executions" is on · test data is fully replaced with production data · the blueprint is exported and stored outside Make.

The first 24 hours after launch

Activation isn't the finish line. Watch these in the first day, while you can still catch a problem before it compounds.

- [ ] Watch the **first real run** end to end in the scenario history — don't assume the test run predicts production.
- [ ] Confirm the **operation cost per run** matches your estimate. A surprise here compounds fast at volume.
- [ ] Check the **Incomplete Executions queue** — anything landing there tells you a handler assumption was wrong.
- [ ] Verify **error notifications actually arrived** where you expected — trigger one on purpose if you have to.
- [ ] Spot-check the **destination system** — did the records land correctly, with nothing duplicated or dropped?

IMPORTABLE FILES

The Scenario Blueprints

Every worked example in the book, exported as a Make blueprint. Import the finished scenario and adapt it instead of rebuilding from the page. Each is annotated to its source chapter.

`01_csv-to-validated-airtable-batch.json`

CSV upload → validated Airtable batch

Parse an uploaded CSV, validate each row, write a clean batch to Airtable — Iterator / Aggregator and the early-filter pattern in one place.

CHAPTER 6 · FLOW CONTROL

`02_shopify-order-segment-fulfill.json`

Shopify order → segment, tag, fulfill

One order in; customer segmentation, an email-list tag, and a fulfillment action out. The Router-driven branching example.

CHAPTER 6 · FLOW CONTROL

`03_webhook-spam-survival.json`

Surviving a webhook spam attack overnight

The defensive build — rate-limit handling, filtering, and error handlers that keep a webhook scenario alive when traffic turns hostile.

CHAPTER 11 · ERROR HANDLING

`04_gmail-inbox-triage-agent.json`

Gmail inbox triage agent

An AI agent that reads, classifies, and routes inbound mail — the end-to-end agent build with tools and instructions.

CHAPTER 14 · AI AGENTS & MCP

`05_sales-outreach-agent.json`

Sales outreach agent

A trigger-driven agent that drafts and sequences outreach — the second AI worked example.

CHAPTER 14 · AI AGENTS & MCP

`06_custom-app-starter.json`

Custom-app build — starter scaffold

The scaffold from the end-to-end custom-app walkthrough — base, connection, and one working module to build on.

CHAPTER 17 · BUILDING CUSTOM APPS

Each blueprint is the scenario as built in the book — a starting point to adapt, not a turnkey drop-in. The next page is how to import and what to change before you run one.

HOW TO IMPORT A BLUEPRINT

1. In Make, create a **new** scenario, open the three-dot menu, and choose **Import Blueprint**.
2. Select the `.json` file — the scenario appears with its modules in place.
3. Reconnect each module to your accounts (connections don't travel with a blueprint), then test before activating.

WHAT TO REPLACE BEFORE YOU RUN IT

- **Connections** — every module needs reconnecting to your own authenticated accounts.
- **IDs and names** — Airtable base/table IDs, Google Sheet names, folder paths, list and tag names.
- **Webhook URLs** — generate your own; the blueprint's are placeholders.
- **Field mappings** — match the blueprint's fields to your real schema.

IF THE IMPORT FAILS

- Confirm you selected the `.json` file — not this PDF.
- Import into a new scenario, not an existing one.
- Reconnect every module before the first test run.
- Replace placeholder IDs, sheet/table names, and webhook URLs.
- Don't activate until all test data is swapped for production data — and clone before adapting heavily.

NEW TO MAKE?

You'll need a free account to import these.

Importing a blueprint requires a Make account — the free plan covers everything in this kit (1,000 monthly credits and 2 active scenarios is enough to follow all six worked examples). It takes about a minute to sign up.

`make.com/en/register?pc=mmsvegas`

Sign up takes about a minute. No credit card needed for the free tier.

KEEP IT CURRENT

Make changes. This kit changes with it.

A printed book is a snapshot. The platform it documents is not. That's the one real weakness of any reference like this — and the reason you're on the list.

When Make ships a change that touches anything in this kit — a renamed module, a moved limit, a new way to build agents — the corrected card lands in your inbox. You don't have to wonder whether your reference has gone stale. We watch the platform so you don't have to.

> *The money was never in knowing the tool. It's in shipping scenarios that don't break at 2 a.m. — and knowing exactly what to check before they go live.*

STAY CURRENT & GET THE UPDATES

mmsvegas.com/make-resources

Corrected cards as Make changes — straight to your inbox

I spent the first half of my career as a marketer hamstrung by the technical skills I didn't have — waiting on someone else to build the thing. Make is one of the tools that ended that for me. This kit is the part I wish someone had handed me on day one. — Brian Kasday

MMS VEGAS · THE MISSING MANUAL FOR MAKE · READER COMPANION KIT