```
@ ? d: -> =>
```

Lonelog: Community Add-on Guidelines

How to Write Add-ons That Work with the Core

Version 1.2.0



Table of contents

Lonelog Community Add-on Guidelines **1**
1. The Non-Negotiables 2
2. Strong Recommendations 6
3. Structural Guidelines 8
4. Community Practices 11
5. Quick Compliance Checklist 13

Lonelog Community Add-on Guidelines

These guidelines exist so that any Lonelog add-on — whether written for personal use, shared on Reddit, or published on itch.io — reads consistently alongside the core system and other add-ons. If a player knows Lonelog, they should be able to pick up a compliant add-on and understand it immediately.

This isn't a list of rules imposed from outside. It's the distilled reasoning behind how the official add-ons were designed. Understanding the *why* makes the constraints easy to follow, and easy to apply to situations the guidelines don't explicitly cover.

1. The Non-Negotiables

These are the constraints that define a Lonelog add-on as distinct from a fork or a separate system. Violating any of them produces something that may be excellent, but isn't a Lonelog add-on.

1.1 Do Not Replace Core Symbols

The five core symbols, together with `tbl:` from the optional layers, are the shared language of all Lonelog logs:

Symbol	Role
`@`	Player action
`?`	Oracle question
`d:`	Roll or rule that gives a verdict
`tbl:`	Random result that gives content
`->`	Resolution outcome
`=>`	Consequence

An add-on may introduce new **tags**, **structural blocks**, and **formatting conventions**. It may not introduce a new symbol that duplicates, overrides, or conflicts with any of the above.

Compliant: Introducing `[F:Enemy|HP 8]` to tag foes in combat.
Non-compliant: Introducing `>>` as a new symbol for attacks, bypassing `@` and `d:`.

`!` is already taken: Partylog uses it for GM and world events in group play. An add-on may use `!` in that same sense, never for anything else.

If you find yourself wanting a sixth core symbol, that's a signal to reconsider the design. The existing five are almost always sufficient when used with new tags or structural conventions.

1.2 Do Not Conflict with Existing Tags

The core spec defines these tag prefixes:

- `N:` — NPCs
- `L:` — Locations
- `E:` — Events: notable things that happened (no counter)
- `PC:` — Player character
- `Thread:` — Story threads
- `Clock:`, `Track:`, `Timer:` — Progress elements
- `#` prefix — Reference tags
- `Inv:`, `Wealth:` — Resource tracking (Resource Add-on)
- `R:` — Rooms (Dungeon Crawling Add-on)
- `F:` — Combatant foes (Combat Add-on)
- `Unit:` — Units (Solo Wargaming Add-on)
- `Faction:`, `Party:` — Factions and party-level state (Partylog, Lorelog)

Do not reuse these prefixes for different purposes. If you need a new tag type, use a prefix that isn't in this list and document it clearly in your add-on's quick reference.

Compliant: `[R:Room4|status:active]` for room states (Dungeon Crawling Add-on). **Non-compliant:** `[L:Room4|active]` — this repurposes the Location tag for a different semantic meaning.

Watch for collision between tags and structural markers. The Dungeon Crawling Add-on uses `[R:]` for rooms; the Combat Add-on originally used `R#` for rounds — the same letter, causing ambiguity in combined logs. The solution was to change round markers to `Rd#`. When introducing any new marker or tag prefix, check for collisions not just with existing tags but with structural conventions (round markers, roster lines, etc.) across the full add-on ecosystem.

When uncertain whether a prefix conflicts, check the full add-on ecosystem before finalizing your choice.

1.3 Respect Core Syntax Rules

Some characters and positions have a fixed meaning in core Lonelog (see core Appendix D). Don't give them a second one:

- **`#`** marks a reference tag (`[#N:Jonah]`). Don't use it for comments or descriptions.
- **Brackets around a dice expression** are taken: before the dice, `+[…]`/`-[…]` is roll context; after the dice, `[…]` holds individual results.
- **Tags accumulate.** Your tags should follow core's update rules: a later tag adds values, `-value` removes one, `old>new` replaces one, `category:value` replaces a category's state, `Stat N` sets a number and `Stat±N` adjusts it, and counts are `×N` values, never part of the name. If your tag needs different behavior, say so explicitly.
- **`d:` gives a verdict, `tbl:` gives content.** If your add-on records a new kind of draw or roll, place it on that line, not by the mechanism that produced it.
- **Fields don't depend on order.** Every value in your tags should say what it is: a labelled number (`HP 6`), a count (`×12`), a category (`status:cleared`) or a plain value. Tags that track something owned are named after the owner (`[Wealth:Kael|Gold 45]`).
- **`->` marks an outcome only.** Don't use it as an arrow in prose, tags or action text.
- **Meta notes are `(label: text)`.** You may suggest new labels for your domain; don't redefine the core ones (`note:`, `retcon:`, …).

1.4 Include Required Metadata

Every add-on must open with a YAML front matter block that includes these fields:

```
---
title: "Lonelog: [Add-on Name] Add-on"
subtitle: "[Short description]"
author: [Your name or handle]
version: [Semantic version, e.g. 1.0.0]
license: CC BY-SA 4.0
lang: en
parent: Lonelog [version this add-on
targets, e.g. v1.7.0]
requires: Core Notation (§3)
---
```

The `requires` field must list the core sections your add-on depends on. If you also depend on another add-on, list it: `requires: Core Notation (§3), Resource Tracking Add-on (§1–§3)`.

The `parent` field pins your add-on to a specific core version. When the core updates, add-on authors can assess whether their add-on needs updating.

1.5 Use CC BY-SA 4.0 (or Compatible)

Official and community add-ons should use the same license as the core: **Creative Commons Attribution-ShareAlike 4.0 International**. This keeps the ecosystem open — anyone can use, adapt, and share add-ons as long as they attribute and share alike.

If you have a specific reason to use a different license, ensure it is compatible with CC BY-SA 4.0. Do not use licenses that would restrict derivative works or require proprietary terms.

2. Strong Recommendations

These aren't absolute requirements, but they distinguish a well-designed add-on from a merely functional one. The official add-ons follow all of them.

2.1 Extend, Don't Invent

Before adding anything new, ask: can an existing mechanism handle this?

- Need to track enemy health? The `[N:]` tag already takes freeform properties: `[N:Goblin|HP 4]`.
- Need a countdown? `[Timer:]` already exists.
- Need to record a roll? `d:` is already there.
- Need random content from a table, deck or generator? That's `tbl:`.

The best add-ons add as little as possible to achieve their goal. Every new tag or convention is something players have to learn and remember. Earn that cognitive load.

2.2 Write for Both Digital and Analog

Lonelog is explicitly format-agnostic. Your add-on should be too. Whenever you introduce a structural block or notation pattern, show how it works in both digital markdown and analog notebook formats.

If a convention genuinely doesn't translate to analog (e.g., it depends on hyperlinking), say so explicitly and provide an analog alternative if possible.

2.3 Scale from Compact to Detailed

Core Lonelog works at multiple levels of detail — from ultra-compact single-line shorthand to rich narrative logs. Add-ons should respect this range.

For every significant notation feature, provide at least two examples: one minimal (fast play, high-information-density) and one expanded (narrative-rich, exploratory). Players should be able to adopt your add-on at whatever detail level matches their play style.

2.4 Show Integration Examples

At least one example in your add-on should demonstrate the add-on being used *alongside* core Lonelog in a realistic play sequence — not just the new features in isolation. Players need to see how the pieces fit together in actual log entries.

If your add-on is designed to work alongside another add-on, include a combined example.

2.5 Provide a Quick Reference

Every add-on should end with a quick reference section: a table or compact list of every new tag, symbol convention, and structural block introduced. This is the first thing an experienced player will check after reading the add-on once, and the thing they'll return to during actual play.

3. Structural Guidelines

3.1 Recommended Document Structure

Follow this section order for consistency across the add-on library:

1. **Overview** — What problem does this add-on solve? When should players use it? (2–4 paragraphs)
2. **What This Add-on Adds** — A table: new tags, new conventions, new structural blocks
3. **Design Principles** — 3–5 principles that explain the *why* behind the design choices
4. **Core sections** — One section per major feature, each with format, examples, and guidance
5. **Cross-Add-on Interactions** — How this add-on works alongside others (omit if not applicable)
6. **System Adaptations** — How the add-on adapts to specific RPG systems (PbtA, OSR, etc.)
7. **Best Practices** — Do/Don't examples in the style of Lonelog §7
8. **Quick Reference** — Tables of all new notation elements
9. **FAQ** — Anticipated questions about edge cases and design choices

You don't have to use all nine sections if some don't apply. The overview, what's added, core sections, best practices, and quick reference are the minimum for a useful add-on.

3.2 Writing Tone

Lonelog's documentation has a specific voice: direct, practical, encouraging, non-prescriptive. It explains *why* conventions exist, not just *what* they are. It treats readers as capable adults who will adapt the notation to their needs.

When writing your add-on, match this tone. Avoid: - Prescriptive language like "you must" or "you should always" (use "do" and "consider") - Assumptions about which RPG system the reader uses - Implying that players who don't use the add-on are playing incorrectly

3.3 Structural Block Syntax

When your add-on introduces a structural block — a delimited region that wraps a mode of play (like combat, a dungeon session status, or a resource snapshot) — use the bracket tag convention:

```
[BLOCK NAME]
...contents...
[/BLOCK NAME]
```

This is the same pattern as `[COMBAT]` / `[/COMBAT]` in the Combat Add-on. It keeps structural blocks visually consistent with ordinary tags and makes them grep-friendly.

For **analog notebook** equivalents, use dashed separators:

```
--- BLOCK NAME ---
...contents...
--- END BLOCK NAME ---
```

Do not use `=== ... ===` or `--- ... ---` as the primary (digital) block delimiter. Reserve those for analog alternatives only.

Compliant:

```
[DUNGEON STATUS]
[R:1|status:cleared|entry hall]
[/DUNGEON STATUS]
```

Non-compliant:

```
```

```
=== Dungeon Status ===
[R:1|status:cleared|entry hall]
```

Every structural block must have an explicit closing tag. Open-ended blocks (no closing delimiter) are not compliant.

3.4 Example Quality

Examples are the most important part of any add-on. They should:

- Show **complete sequences**, not isolated lines (action → roll → consequence → tags)
- Use **realistic fiction**, not abstract placeholders (`@ Attack the goblin`, not `@ ACTION`)
- **Progress logically** — the reader should feel they're seeing a real moment of play
- **Demonstrate edge cases** as well as the common case

Avoid examples that are so simple they don't reveal anything about how the notation behaves under real conditions.

Don't put tags inside table cells. A tag like `[F:Thug|HP 6]` contains `|`, which ends a Markdown table cell. Escaping it as `\|` works on GitHub, but Pandoc (and so the PDF and EPUB builds) prints the backslash. Use a list instead: `- `[F:Thug|HP 6]` — a foe with 6 HP`.

4. Community Practices

4.1 Versioning

Use semantic versioning: `major.minor.patch`.

- **Patch** (1.0.0 → 1.0.1): Typo fixes, clarifications that don't change behavior
- **Minor** (1.0.0 → 1.1.0): New optional features, expanded examples, backward-compatible additions
- **Major** (1.0.0 → 2.0.0): Changes that break compatibility with previous versions of the add-on

When your add-on's `parent` version changes (e.g., Lonelog updates from v1.6.0 to v1.7.0), review your add-on and update the `parent` field. If the core change affects your add-on's behavior, treat it as at least a minor version bump.

4.2 Naming

Add-on file names should follow the pattern: `lonelog-[name]-addon.md` or simply `[name]-addon.md` within a Lonelog-specific repository. This makes the origin clear when files are shared outside their folder context.

Add-on titles in the YAML front matter should follow: `"Lonelog: [Name] Add-on"`.

4.3 Attribution and Derivatives

If you build on someone else's community add-on, credit them in your document and respect their license terms. CC BY-SA 4.0 requires attribution and that derivatives carry the same license.

If you publish an add-on that substantially incorporates another, note the original author and version in your metadata:

```
based-on: "[Original Add-on Name] by
[Author]"
```

4.4 Sharing

When sharing add-ons on Reddit, itch.io, or other platforms: - Link to the core Lonelog document so new readers have context - State which core version your add-on targets - Note any add-on dependencies - Invite feedback — the community's collective use improves add-ons faster than solo development

5. Quick Compliance Checklist

Before publishing or sharing your add-on, verify:

Non-negotiables: - [ ] No new symbols that conflict with `@`, `?`, `d:`, `tbl:`, `->`, `=>`, or with Partylog's `!` - [ ] No second meaning for `#`, for brackets around dice expressions, or for core meta-note labels - [ ] New tags follow core's update rules (accumulate, `-value`, `old>new`, `category:value` for states, `×N` for counts), or document why not - [ ] No tag prefixes that conflict with existing core or add-on tags - [ ] YAML front matter present with all required fields - [ ] `parent` field specifies the core version targeted - [ ] License is CC BY-SA 4.0 (or compatible)

Strong recommendations: - [ ] Shows both digital and analog examples - [ ] Includes at least one integration example with core Lonelog - [ ] Provides compact and expanded variants for major features - [ ] Ends with a quick reference table - [ ] Writing tone matches Lonelog's voice (practical, non-prescriptive, explanatory)

Structure: - [ ] Overview explains the problem and when to use the add-on - [ ] "What This Add-on Adds" table present - [ ] Best practices section with Do/Don't examples - [ ] FAQ addresses likely edge case questions - [ ] Structural blocks use `[BLOCK]` / `[/BLOCK]` syntax (not `===` or `---` as primary format)

Version History:

- v1.2.0: Aligned with Lonelog v1.7: `tbl:` listed with the core symbols, new §1.3 on reserved syntax, updated checklist.
- v1.1.0: Previous release