@ ? d: -> =>

Lonelog : Directives pour les Extensions de la Communauté

Comment écrire des extensions qui fonctionnent avec le cœur du système

Version 1.1.0



Table des matières

Directives pour les Extensions de la Communauté Lonelog **1**
1. Les Points Non Négociables 2
2. Recommandations Fortes 6
3. Directives Structurelles 8
4. Pratiques de la Communauté 11
5. Liste de Contrôle Rapide de Conformité 13

Directives pour les Extensions de la Communauté Lonelog

Ces directives existent pour que toute extension Lonelog — qu'elle soit écrite pour un usage personnel, partagée sur Reddit ou publiée sur itch.io — soit cohérente avec le système de base et les autres extensions. Si un joueur connaît Lonelog, il doit être capable de prendre une extension conforme et de la comprendre immédiatement.

Il ne s'agit pas d'une liste de règles imposées de l'extérieur. C'est le raisonnement distillé derrière la conception des extensions officielles. Comprendre le *pourquoi* rend les contraintes faciles à suivre et à appliquer à des situations que les directives ne couvrent pas explicitement.

1. Les Points Non Négociables

Ce sont les contraintes qui définissent une extension Lonelog comme distincte d'un « fork » ou d'un système séparé. Violer l'une d'entre elles produit quelque chose qui peut être excellent, mais qui n'est pas une extension Lonelog.

1.1 Ne pas remplacer les symboles de base

Les cinq symboles de base sont le langage partagé de tous les journaux Lonelog :

Symbole	Rôle
`@`	Action du joueur
`?`	Question à l'oracle
`d:`	Jet de mécanique
`->`	Résultat de la résolution
`=>`	Conséquence

Une extension peut introduire de nouvelles **balises**, des **blocs structurels** et des **conventions de formatage**. Elle ne peut pas introduire un nouveau symbole qui ferait doublon, remplacerait ou entrerait en conflit avec l'un des cinq ci-dessus.

Conforme : Introduire `[F:Ennemi|PV 8]` pour baliser les adversaires en combat.
Non conforme : Introduire `!` comme nouveau symbole pour les attaques, en contournant `@` et `d:`.

Si vous ressentez le besoin d'un sixième symbole de base, c'est le signe qu'il faut revoir la conception. Les cinq symboles existants sont presque toujours suffisants lorsqu'ils sont utilisés avec de nouvelles balises ou conventions structurelles.

1.2 Ne pas entrer en conflit avec les balises existantes

La spécification de base définit ces préfixes de balises :

- `N:` — PNJ (NPCs)
- `L:` — Lieux (Locations)
- `E:` — Événements
- `PJ:` — Personnage joueur (PC:)
- `Fil:` — Fils conducteurs (Thread:)
- `Horloge:`, `Piste:`, `Compteur:` — Éléments de progression
- préfixe `#` — Balises de référence
- `Inv:`, `Richesse:` — Suivi des ressources (Extension de Ressources)
- `P:` — Pièces (Extension de Donjon)
- `F:` — Adversaires en combat (Extension de Combat)

Ne réutilisez pas ces préfixes pour d'autres usages. Si vous avez besoin d'un nouveau type de balise, utilisez un préfixe qui ne figure pas dans cette liste et documentez-le clairement dans la référence rapide de votre extension.

Conforme : `[P:Pièce4|active]` pour l'état d'une pièce (Extension de Donjon).
Non conforme : `[L:Pièce4|active]` — cela détourne la balise de Lieu pour une signification sémantique différente.

Attention aux collisions entre balises et marqueurs structurels. L'Extension de Donjon utilise `[P:]` pour les pièces ; l'Extension de Combat utilisait initialement `T#` pour les tours (Rounds) — la même lettre que pour les fils (Threads) dans certaines versions, causant une ambiguïté. La solution a été d'utiliser `Rd#` en anglais ou de veiller à la distinction en français. Lors de l'introduction d'un nouveau marqueur ou préfixe, vérifiez les collisions non seulement avec les balises existantes, mais aussi avec les conventions structurelles sur l'ensemble de l'écosystème des extensions.

En cas de doute sur un conflit de préfixe, vérifiez l'ensemble de l'écosystème avant de finaliser votre choix.

1.3 Inclure les métadonnées requises

Chaque extension doit s'ouvrir par un bloc YAML front matter incluant ces champs :

```yaml
---
title: "Lonelog : Extension [Nom de
l'Extension]"
subtitle: "[Description courte]"
author: [Votre nom ou pseudo]
version: [Version sémantique, ex: 1.0.0]
license: CC BY-SA 4.0
lang: fr
parent: Lonelog [version cible, ex:
v1.3.0]
requires: Notation de Base (§3)
---
```

Le champ `requires` doit lister les sections du cœur dont votre extension dépend. Si vous dépendez aussi d'une autre extension, listez-la : `requires: Notation de Base (§3), Extension de Suivi des Ressources (§1–§3)`.

Le champ `parent` lie votre extension à une version spécifique du cœur. Quand le cœur est mis à jour, les auteurs d'extensions peuvent évaluer si leur extension nécessite une mise à jour.

1.4 Utiliser CC BY-SA 4.0 (ou compatible)

Les extensions officielles et communautaires doivent utiliser la même licence que le cœur : **Creative Commons Attribution-ShareAlike**

4.0 International. Cela maintient l'écosystème ouvert — n'importe qui peut utiliser, adapter et partager les extensions tant qu'il cite l'auteur et partage dans les mêmes conditions.

Si vous avez une raison spécifique d'utiliser une licence différente, assurez-vous qu'elle soit compatible avec la CC BY-SA 4.0. N'utilisez pas de licences qui restreindraient les œuvres dérivées ou imposeraient des conditions propriétaires.

2. Recommandations Fortes

Ce ne sont pas des exigences absolues, mais elles distinguent une extension bien conçue d'une extension simplement fonctionnelle. Les extensions officielles les suivent toutes.

2.1 Étendre, ne pas inventer

Avant d'ajouter quoi que ce soit de nouveau, demandez-vous : un mécanisme existant peut-il gérer cela ?

- Besoin de suivre la santé des ennemis ? La balise `[N:]` accepte déjà des propriétés libres : `[N:Gobelin|PV 4]`.
- Besoin d'un compte à rebours ? `[Compteur:]` existe déjà.
- Besoin d'enregistrer un jet ? `d:` est déjà là.

Les meilleures extensions ajoutent le moins possible pour atteindre leur but. Chaque nouvelle balise ou convention est une chose que les joueurs doivent apprendre et retenir. Méritez cette charge cognitive.

2.2 Écrire pour le numérique et l'analogique

Lonelog est explicitement indépendant du format. Votre extension devrait l'être aussi. Chaque fois que vous introduisez un bloc structurel ou un modèle de notation, montrez comment il fonctionne à la fois en markdown numérique et sur un carnet analogique.

Si une convention ne se transpose vraiment pas à l'analogique (par exemple, si elle dépend de liens hypertextes), dites-le explicitement et proposez une alternative analogique si possible.

2.3 Passer du compact au détaillé

Le cœur de Lonelog fonctionne à plusieurs niveaux de détail — du raccourci ultra-compact sur une seule ligne aux journaux narratifs riches. Les extensions doivent respecter cette plage.

Pour chaque fonctionnalité de notation importante, fournissez au moins deux exemples : un minimal (jeu rapide, haute densité d'information) et un développé (narration riche, exploratoire). Les joueurs doivent pouvoir adopter votre extension quel que soit le niveau de détail qui correspond à leur style de jeu.

2.4 Montrer des exemples d'intégration

Au moins un exemple dans votre extension devrait démontrer l'utilisation de l'extension *aux côtés* du cœur de Lonelog dans une séquence de jeu réaliste — pas seulement les nouvelles fonctionnalités isolées. Les joueurs ont besoin de voir comment les pièces s'assemblent dans de réelles entrées de journal.

Si votre extension est conçue pour fonctionner avec une autre extension, incluez un exemple combiné.

2.5 Fournir une référence rapide

Chaque extension doit se terminer par une section de référence rapide : un tableau ou une liste compacte de chaque nouvelle balise, convention de symbole et bloc structurel introduit. C'est la première chose qu'un joueur expérimenté vérifiera après avoir lu l'extension une fois, et l'élément vers lequel il reviendra pendant le jeu.

3. Directives Structurelles

3.1 Structure de document recommandée

Suivez cet ordre de sections pour la cohérence de la bibliothèque d'extensions :

1. **Présentation** — Quel problème cette extension résout-elle ? Quand les joueurs devraient-ils l'utiliser ? (2–4 paragraphes)
2. **Ce que cette extension ajoute** — Un tableau : nouvelles balises, nouvelles conventions, nouveaux blocs structurels.
3. **Principes de conception** — 3–5 principes qui expliquent le *pourquoi* derrière les choix de conception.
4. **Sections principales** — Une section par fonctionnalité majeure, chacune avec format, exemples et conseils.
5. **Interactions entre extensions** — Comment cette extension fonctionne avec les autres (omettre si non applicable).
6. **Adaptations du système** — Comment l'extension s'adapte à des systèmes de JDR spécifiques (PbtA, OSR, etc.).
7. **Bonnes pratiques** — Exemples « À faire / À éviter » dans le style de Lonelog §7.
8. **Référence rapide** — Tableaux de tous les nouveaux éléments de notation.
9. **FAQ** — Questions anticipées sur les cas limites et les choix de conception.

Vous n'êtes pas obligé d'utiliser les neuf sections si certaines ne s'appliquent pas. La présentation, ce qui est ajouté, les sections principales, les bonnes pratiques et la référence rapide sont le minimum pour une extension utile.

3.2 Ton de l'écriture

La documentation de Lonelog a une voix spécifique : directe, pratique, encourageante, non prescriptive. Elle explique *pourquoi* les conventions existent, pas seulement *ce qu'elles sont*. Elle traite les

lecteurs comme des adultes capables qui adapteront la notation à leurs besoins.

Lorsque vous écrivez votre extension, adoptez ce ton. Évitez : - Un langage prescriptif comme « vous devez » ou « vous devriez toujours » (utilisez « faites » et « envisagez »). - Des suppositions sur le système de JDR utilisé par le lecteur. - Sous-entendre que les joueurs qui n'utilisent pas l'extension jouent de manière incorrecte.

3.3 Syntaxe des blocs structurels

Lorsque votre extension introduit un bloc structurel — une région délimitée qui enveloppe un mode de jeu (comme le combat, un état de session de donjon ou un instantané de ressources) — utilisez la convention des balises entre crochets :

```
[NOM DU BLOC]
...contenu...
[/NOM DU BLOC]
```

C'est le même modèle que `[COMBAT]` / `[/COMBAT]` dans l'Extension de Combat. Cela maintient la cohérence visuelle des blocs structurels avec les balises ordinaires et les rend faciles à rechercher (grep).

Pour les équivalents en **carnet analogique**, utilisez des séparateurs en pointillés :

```
--- NOM DU BLOC ---
...contenu...
--- FIN NOM DU BLOC ---
```

N'utilisez pas `=== ... ===` ou `--- ... ---` comme délimiteur de bloc principal (numérique). Réservez-les uniquement aux alternatives analogiques.

Conforme :

```
[STATUT DU DONJON]
[P:1|nettoyée|hall d'entrée]
[/STATUT DU DONJON]
```

Non conforme :

```
=== Statut du Donjon ===
[P:1|nettoyée|hall d'entrée]
```

Chaque bloc structurels doit avoir une balise de fermeture explicite. Les blocs ouverts (sans délimiteur de fermeture) ne sont pas conformes.

3.4 Qualité des exemples

Les exemples sont la partie la plus importante de toute extension. Ils doivent :

- Montrer des **séquences complètes**, pas des lignes isolées (action → jet → conséquence → balises).
- Utiliser une **fiction réaliste**, pas des substituts abstraits (`@ Attaquer le gobelin`, pas `@ ACTION`).
- **Progresser logiquement** — le lecteur doit sentir qu'il voit un réel moment de jeu.
- **Démontrer des cas limites** ainsi que le cas courant.

Évitez les exemples trop simples qui ne révèlent rien sur le comportement de la notation dans des conditions réelles.

4. Pratiques de la Communauté

4.1 Versionnage

Utilisez le versionnage sémantique : `majeur.mineur.correctif`.

- **Correctif** (1.0.0 → 1.0.1) : Corrections de fautes de frappe, clarifications qui ne changent pas le comportement.
- **Mineur** (1.0.0 → 1.1.0) : Nouvelles fonctionnalités optionnelles, exemples étendus, ajouts rétrocompatibles.
- **Majeur** (1.0.0 → 2.0.0) : Changements qui brisent la compatibilité avec les versions précédentes de l'extension.

Lorsque la version `parent` de votre extension change (ex: Lonelog passe de v1.1.0 à v1.2.0), examinez votre extension et mettez à jour le champ `parent`. Si le changement du cœur affecte le comportement de votre extension, considérez cela comme au moins une version mineure.

4.2 Nommage

Les noms de fichiers d'extension doivent suivre le modèle : `lonelog-[nom]-addon-fr.md` ou simplement `[nom]-addon-fr.md` au sein d'un dépôt spécifique à Lonelog. Cela rend l'origine claire lorsque les fichiers sont partagés en dehors de leur contexte de dossier.

Les titres d'extension dans le YAML front matter doivent suivre : `"Lonelog : Extension [Nom]"`.

4.3 Attribution et dérivés

Si vous vous appuyez sur l'extension communautaire de quelqu'un d'autre, citez-le dans votre document et respectez les termes de sa li-

cence. La CC BY-SA 4.0 exige l'attribution et que les dérivés portent la même licence.

Si vous publiez une extension qui incorpore substantiellement une autre, notez l'auteur original et la version dans vos métadonnées :

```
based-on: "[Nom de l'extension originale]
par [Auteur]"
```

4.4 Partage

Lors du partage d'extensions sur Reddit, itch.io ou d'autres plateformes : - Liez vers le document Lonelog de base pour que les nouveaux lecteurs aient le contexte. - Indiquez quelle version du cœur votre extension cible. - Notez toute dépendance envers d'autres extensions. - Invitez aux retours — l'usage collectif de la communauté améliore les extensions plus vite que le développement en solo.

5. Liste de Contrôle Rapide de Conformité

Avant de publier ou de partager votre extension, vérifiez :

Points non négociables : - [ ] Aucun nouveau symbole n'entre en conflit avec `@`, `?`, `d:`, `->`, `=>`. - [ ] Aucun préfixe de balise n'entre en conflit avec les balises existantes du cœur ou des extensions. - [ ] YAML front matter présent avec tous les champs requis. - [ ] Le champ `parent` spécifie la version du cœur ciblée. - [ ] La licence est CC BY-SA 4.0 (ou compatible).

Recommandations fortes : - [ ] Montre des exemples numériques et analogiques. - [ ] Inclut au moins un exemple d'intégration avec le cœur de Lonelog. - [ ] Fournit des variantes compactes et étendues pour les fonctionnalités majeures. - [ ] Se termine par un tableau de référence rapide. - [ ] Le ton de l'écriture correspond à la voix de Lonelog (pratique, non prescriptif, explicatif).

Structure : - [ ] La présentation explique le problème et quand utiliser l'extension. - [ ] Tableau « Ce que cette extension ajoute » présent. - [ ] Section bonnes pratiques avec des exemples « À faire / À éviter ». - [ ] La FAQ aborde les questions probables sur les cas limites. - [ ] Les blocs structurels utilisent la syntaxe `[BLOC]` / `[/BLOC]` (pas `===` ou `---` comme format principal).