Procédure de Gestion des Versions

Informations qualité

Suivi des modifications majeures	5 mars 2015 - Nicolas Marchand - Création du document 1 décembre 2015 - Nicolas Marchand - Modification par rapport aux tests à passer suivant le niveau des mises à jour 14 décembre 2015 - Guillaume Natali - Modification des règles de numérotation 02 aout 2017 - Nicolas Marchand - Portage sur DoKuWiKi & Evolutions 28 août 2020 - Nicolas Marchand - Evolution raisons changement de version 20 novembre 2025 - Nicolas Marchand, Guillaume Natali - Dernière révision avant fusion 11 août 2026 - Nicolas Marchand - Fusion des procédures #06, #30 (`gestionsvn`) et « Gestion du dépôt Git » (Kevin Kopinski, `gestionsdepotgit`) en un seul document ; nouveau mécanisme de signalement des versions beta (4e chiffre porteur de la version visée, remplace le signal Y=99) ; alignement du client lourd et des serveurs sur cette même règle
Suivi des approbations	Cf cartographie
Objet	L'objet de ce document est de définir le système de versionnage utilisé pour l'ensemble des composants LoGeAs (LoGeAs Web, client lourd, serveurs), les différents types de versions, les actions à faire lors de la sortie d'une nouvelle version, et les modalités d'intégration du code dans le dépôt Git.
Destinataires	**- Validation des modifications :** Gérant **- Approbation du document** : Equipe dev & Equipe Ass

Objectif et périmètre

Cette procédure définit le système de versionnage utilisé pour l'ensemble des composants LoGeAs — **LoGeAs Web**, qui fait référence, ainsi que le **client lourd** et les **serveurs** (PGI, LoGeAs, Nono) — et la façon dont chaque version est enregistrée dans son dépôt Git avec identification unique et authenticité garanties.

Chaque composant conserve son propre numéro de version et son propre rythme de publication (ils ne sont pas synchronisés entre eux), mais tous suivent désormais **la même règle de numérotation**. Une matrice de compatibilité (fin de document) recense les versions connues de chaque composant.

Système de Numérotation

Le numéro de version comporte 3 chiffres pour une version publiée, et 4 chiffres pendant sa phase de test beta :

- **X (Majeur)** — numéro marketing/stratégique. Reflète des évolutions substantielles ou des

migrations technologiques (ex. changement de base de données).
- **Y (Mineur)** — nouvelles fonctionnalités, modifications de structure de données, changement de réglementation ou nouvelle certification.
- **Z (Release)** — change à **chaque publication du logiciel** (obligatoire). Ne réutilise jamais un numéro déjà publié.
- **W (Build, uniquement pendant une phase beta)** — numéro d'itération de la beta en cours de test.

Règle fondamentale

Une fois qu'une version **X.Y.Z** est publiée, son contenu ne doit **jamais** être modifié. Toute modification, même mineure, exige une nouvelle version.

Signalement d'une version beta (X.Y.Z.W)

Une version en cours de test beta porte directement le numéro **X.Y.Z que cette beta vise à publier**, suivi d'un 4e chiffre **W** qui numérote les itérations successives de cette beta (1, 2, 3…), remis à 1 à chaque nouveau cycle.

Exemple concret :

- Dernière version publiée : `11.0.25`
- Un correctif est en préparation, sans nouvelle fonctionnalité → il vise `11.0.26`. Sa 1re beta est numérotée `11.0.26.1`, un correctif suite à retour de test donne `11.0.26.2`, etc.
- Une fois validée, la beta `11.0.26.2` devient la release `11.0.26` (le 4e chiffre est retiré, le X.Y.Z ne change pas).
- Une évolution plus importante (nouvelles fonctionnalités) viserait `11.1.0` au lieu de `11.0.26` — ses betas seraient `11.1.0.1`, `11.1.0.2`, etc.

Pourquoi ce mécanisme plutôt que l'ancien signal Y=99 (utilisé auparavant, y compris dans le code du client lourd) : avec un flag générique, deux cycles beta consécutifs qui ne changent pas le mineur — le cas le plus fréquent — produisent le même numéro de beta, sans moyen de savoir à quelle version future chacun correspond. En faisant porter directement le numéro visé, l'ambiguïté disparaît et la progression reste strictement croissante, sans trou ni collision possible.

Critères de Changement de Version

Majeur (X) : évolutions substantielles ou migrations technologiques (ex. changement de base de données).

Mineur (Y) : nouvelles fonctionnalités, modifications de structure de données (ajout de champs), changements de réglementation pris en compte, ou nouvelle certification.

Release (Z) : à chaque publication du logiciel (publication obligatoire) — correctifs, corrections de bugs, ajustements sans changement fonctionnel structurant.

Build (W) : uniquement pendant une phase beta, itération de test au sein d'un même cycle. N'apparaît jamais sur une version publiée.

Actions Requises par Type de Version

Version Majeure

Avant développement : Séparation de la documentation utilisateur.

Avant livraison : Validation complète des tests ProjeQtOr, mise à jour de documentation technique/utilisateur, création d'actualités web, taguage Git (voir ci-dessous), archivage des binaires, modification des URLs de mise à jour, mise à jour EPUdF REGALE.

Après livraison : Notification utilisateurs, dépôt auprès EPUdF et coffre numérique.

Version Mineure

Avant livraison : Validation des tests, évolution documentation technique/utilisateur, mise à jour actualités web, taguage Git, archivage binaires, mise à jour EPUdF REGALE.

Après livraison : Notification utilisateurs, dépôt coffre numérique, extension certification logiciel.

Version Release

Avant livraison : Validation des tests pertinents, mise à jour documentation technique, actualités web, taguage Git, archivage binaires, mise à jour EPUdF REGALE.

Après livraison : Dépôt codes/binaires en coffre numérique.

Intégration du code (dépôt Git)

Cette section fusionne l'ancienne procédure « gestion du dépôt Git » (Kevin Kopinski, 10/11/2025). Sa version originale décrivait un flux par ticket GitLab et relecture systématique par un second développeur — ce flux n'a jamais correspondu à la pratique réelle et n'est plus d'actualité (confirmé en 2026-08, suite au départ de Kevin Kopinski) : pas de ticket, pas de merge request, travail réalisé directement sur le poste du développeur.

1. **Travail sur branche personnelle** : chacun développe sur sa propre branche, créée depuis la branche de version en cours (ex. `11.0.25`). Pas de ticket GitLab ni de merge request requis.
2. **Sauvegarde régulière** : commit et push de la branche personnelle au fil de l'eau, sans vérification imposée à ce stade (procédure manuelle).
3. **Intégration = validation** : la validation d'une fonction se fait au moment de son intégration dans la branche de version courante, via la procédure outillée `/fusion-dev` (assistant Claude

Code, dépôt logeas-web), qui :
- met à jour la branche de version depuis le dépôt distant
- fusionne la branche personnelle avec un commit de fusion portant le log complet des commits intégrés
- vérifie compilation, tests automatiques et style après fusion — c'est ce contrôle technique qui fait office de validation, il n'y a pas de relecture séparée par un second développeur
- publie le résultat vers le dépôt distant

Une version rassemble l'ensemble des changements intégrés sur la branche de version depuis la dernière publication.

Gestion des tags Git

Format des tags

- Beta : `X.Y.Z.W_beta` (ex. `11.0.26.1_beta`)
- Release : `X.Y.Z_release` (ex. `11.0.26_release`)

Ce format s'applique au dépôt LoGeAs Web (`logeas-web`). Les dépôts du client lourd et des serveurs (`logeas-web` historique, groupe Delphi) utilisent une convention de tag distincte, héritée de leur propre historique (ex. `NonoVersion11.0.1.0`, `LoGeAsVersion11.0.2.3Pré-release`) — non harmonisée avec le format ci-dessus pour l'instant.

Étapes d'exécution — LoGeAs Web

Outillées via les commandes `/release-beta` et `/release-client` (assistant Claude Code, dépôt logeas-web) :

1. **Vérification de branche** : la branche de version est propre et à jour avec le dépôt distant
2. **Génération du build** : compilation, incrémentation automatique du numéro (4e chiffre pour une beta, 3e chiffre pour une release)
3. **Commit dédié** : le changement de numéro de version est commité séparément du contenu fonctionnel
4. **Étiquetage** : tag Git annoté créé sur ce commit
5. **Publication** : push du commit et du tag vers le dépôt distant
6. **Relevé d'empreinte** : le hash de commit court est embarqué automatiquement dans `src/assets/version.ts`, consultable directement dans le logiciel livré
7. **Archivage** : conservation de la référence du tag dans le suivi de version (cartographie fonctionnelle)

Cas d'un tag posé manuellement

Respecter impérativement le format défini ci-dessus et ne jamais réutiliser un numéro X.Y.Z déjà présent dans un tag `_release` existant.

Numérotation des autres composants (client lourd, serveurs)

Le client lourd (Delphi, dépôt `logeas-web` du groupe historique) et les serveurs (PGI, LoGeAs, Nono) suivent désormais **la même règle de numérotation** que LoGeAs Web (voir ci-dessus), chacun sur sa propre ligne de version indépendante.

Alignement technique effectué : le code partagé `CommunGlobaux/CommonLoGeAs.pas` contenait un mécanisme spécifique au signal Y=99 (`if v[1]>=99 then inc(v[0])`, dans `GetPathData` et `InitVersion`), utilisé pour corriger le calcul du dossier de données et du libellé de version pendant une beta de version majeure. Cette correction devenant obsolète (plus aucune version n'est censée utiliser Y=99), les 2 occurrences ont été retirées. *(Changement réalisé le 2026-08-11, non encore commité au moment de la rédaction — à valider par compilation/test avant intégration.)*

Point de vigilance : un mécanisme de récupération active de la version de LoGeAs Web depuis le client lourd existe déjà en code (`TCommonLoGeAs.GetVersion_LogeasWeb`, appel prévu vers le serveur Nono, `getVersionGlobale`), mais son corps est actuellement commenté (désactivé). Il pourrait être réactivé plus tard pour passer d'une matrice de compatibilité déclarative à un contrôle actif à la connexion — non fait à ce stade.

Documents fusionnés dans cette page

Cette procédure remplace et fusionne :

- L'ancienne procédure #06 « Gestion des Versions de LoGeAs »
- La procédure #30 « Versionning des codes sources », page désormais obsolète
- La procédure « Gestion du dépôt Git », page désormais obsolète

Suivi Document

Dernière modification : 2026-08-11. Auteurs : Nicolas Marchand, Guillaume Natali. Validateurs : Gérant (modifications), Équipe dev et assistance (approbation).