Kit de documentation

Projet — Guide de mise à jour — FR

1 Pourquoi une injection complète du Kit ?

Ce guide est pour vous, pas pour l'*IA*. Il décrit comment réinjecter une version mise à jour du Kit dans un projet existant sans casser le travail en cours.

Le Kit évolue comme un tout. Une nouvelle version n'apporte pas seulement une *feuille de style* corrigée ou un document reformulé: elle embarque une chaîne complète de dépendances. Une *feuille de style* est toujours livrée avec sa Reference associée, au même *horodatage*. Le *Registry* déclare la nouvelle version active. Le *prompt* technique peut contenir de nouvelles règles qui s'appuient sur les nouvelles fonctions. La structure commune peut introduire de nouveaux contrats normatifs que les fichiers projet doivent respecter.

Absorber une mise à jour à la carte — remplacer une *feuille de style* sans sa Reference, oublier le *Registry*, garder un ancien *prompt* qui ne connaît pas les nouvelles règles — produit des incohérences silencieuses. L'*IA* lit alors des documents qui se contredisent, applique des règles qui ne correspondent plus aux fonctions disponibles, et la structure des documents produits se dégrade sans alerte.

Règle simple: une mise à jour du Kit s’obtient en un seul geste, sous forme d’archive, à l’adresse déclarée au *Registry* dans projectZip.downloadUrl. Tous les fichiers du lot remplacent leurs prédécesseurs; pas de sélection à la carte.

Cette règle est délibérément plus large que le strict nécessaire. Décider fichier par fichier ce qui est utile suppose de connaître les dépendances — c'est précisément ce qu'on ne veut pas exiger de vous. Quelques documents explicatifs de trop dans le *contexte* ne coûtent rien; un fichier manquant produit une incohérence qui ne se manifestera qu'à la *génération* suivante.

💡 *Une mise à jour remplace les fichiers du Kit et laisse les vôtres en place. Votre travail en cours est préservé tant que vous n'avez pas modifié un fichier du Kit lui-même.*

2 Règle de non-compatibilité

Le Kit n'assure aucune compatibilité, ni ascendante ni descendante. Toute nouvelle livraison repart de la *feuille de style* active et de sa Reference associée, script from scratch. Les scripts des sessions précédentes ne sont ni consultés ni adaptés — ils n'existent pas pour la session en cours.

C'est aussi la raison pour laquelle les documents Kit ne portent pas de numéro de version dans leur texte: il n'existe jamais deux versions en circulation. Un document décrit l'état actuel, et l'historique vit dans les blocs de changelog en tête des fichiers de code.

3 Fichiers concernés

Deux catégories de fichiers coexistent dans votre projet: les fichiers Kit, remplacés intégralement à chaque mise à jour, et les fichiers propres à votre projet, jamais touchés.

- **Fichiers Kit — tout, ensemble:** une mise à jour du Kit s'injecte intégralement, telle qu'elle est livrée. La liste complète figure dans

Guide d'initialisation §3 — elle ne varie pas entre initialisation et mise à jour. Aucun examen fichier par fichier n'est nécessaire.

- **Fichiers de votre projet — intouchés:** aucun fichier portant votre préfixe n'est touché par une mise à jour du Kit. Ils restent exactement tels que vous les avez laissés. Seule exception: à l'étape 4 ci-dessous, si un changement structurel introduit par la mise à jour nécessite une migration — et uniquement sur votre confirmation explicite, migration par migration.

4 Procédure

La mise à jour se fait en cinq étapes. Pour les mises à jour courantes — corrections, nouvelles fonctionnalités mineures — le remplacement des fichiers ne nécessite pas d'ouvrir immédiatement une session: la détection de version s'effectue automatiquement au début de la session suivante.

Exception: si la mise à jour introduit un changement structurel majeur — nouveau contrat de *module de données*, nouveau flag au *Registry*, refonte d'un export — une session de validation est obligatoire avant de reprendre le travail documentaire. Le protocole §4.4 s'applique alors.

4.1 Étape 1 — Récupérer les nouveaux fichiers

Récupérer les fichiers Kit mis à jour depuis votre source de référence. Conserver les anciens fichiers sur votre poste jusqu'à la fin de la procédure.

💡 *Conserver toujours la version précédente de la feuille de style jusqu'à validation complète de la nouvelle version en session.*

4.2 Étape 2 — Vérifier les versions

Avant de remplacer, vérifier que le *Registry* fait partie du lot téléchargé et que les numéros de version déclarés correspondent aux versions réelles des nouveaux fichiers.

Vérification	Où trouver l'information
Version du nouveau fichier de style	Premières lignes du fichier .js — le commentaire “Version : x.xx”.
Version déclarée au *Registry*	Kit - *Registry.js*, section stylesheets, champ version de chaque *feuille de style*.
Horodatage* du *couplet	Kit - *Registry.js*, section stylesheets, champ ts. Doit être identique à l'*horodatage* porté par le nom du fichier .js et par celui de sa Reference.

Un fichier de style porte son numéro de version à deux endroits: le commentaire d'en-tête, lisible d'un coup d'œil, et la constante exportée qui est réellement inscrite dans chaque document produit. Un écart entre les deux est invisible à l'exécution mais fausse toute vérification manuelle. L'audit correspondant est décrit dans *Quality Control* §4.2.

💡 *Le Registry est la seule source de vérité pour les versions actives. Une version n'est jamais retenue de mémoire: elle est relue dans le Registry à chaque début de session.*

4.3 Étape 3 — Remplacer dans le projet

Dans l'arbre de votre projet, remplacer les anciens fichiers Kit par les nouveaux, *Registry* compris. L'ordre n'a pas d'importance pour cette étape.

Au début du fil suivant, le *Registry* est lu et tout écart entre les versions déclarées et les versions réellement présentes dans l'arbre est détecté automatiquement.

4.4 Étape 4 — Adapter les fichiers projet

Une mise à jour Kit peut introduire des évolutions qui impactent la structure des fichiers projet: nouveau champ obligatoire dans un *module de données*, nouveau flag, refonte d'un export. Lors de la première session de travail après la mise à jour, le protocole ci-dessous s'applique systématiquement avant toute autre tâche.

- **Diagnostic d'écart:** la structure des fichiers projet existants est comparée à la structure normative de *Structure commune*. Chaque écart est signalé sous la forme "Écart détecté dans [fichier]: [description]. Migration proposée: [action]", et attend votre confirmation explicite. Aucune modification silencieuse.
- **Modules de données:** le contrôle propre aux modules — champs obligatoires, conformité des exports, format multilingue — suit le protocole de *Structure commune* §6.7.
- **Fichiers optionnels absents:** pour chaque fichier optionnel décrit dans *Structure commune* et absent de votre projet, la création vous est proposée explicitement. Jamais de création silencieuse.
- **Fichiers optionnels existants:** un fichier optionnel déjà présent n'est jamais supprimé, même si son flag vaut false. La suppression d'un fichier projet existant n'intervient que sur instruction explicite de votre part.

Note: *Aucun fichier projet n'est jamais supprimé de manière autonome, quelle que soit la raison — flag désactivé, fichier jugé obsolète, ou autre. La suppression est toujours votre décision, formulée dans le chat.*

4.5 Étape 5 — Vérifier le plancher de rendu

La plupart des mises à jour n'affectent pas l'apparence des documents déjà produits. Certaines si: un espacement modifié, une largeur changée, une couleur remplacée. Vos documents anciens diffèrent alors de vos documents récents, et le classeur devient un assemblage de deux mises en page.

Le Kit signale ces cas par un plancher, minDocumentVersion, déclaré dans Kit - *Registry.js* — *Pipeline HTML* §3.1. Il porte la dernière version qui a changé le rendu. Chaque document produit garde en mémoire la version qui l'a produit.

- **Le plancher n'a pas bougé:** rien à faire. Vos documents restent valides, quelle que soit la version qui les a produits.
- **Le plancher a monté:** tout document produit par une version antérieure se régénère. Le contenu ne change pas — il est extrait du fichier source existant et rendu par la version active. Le générateur de site refuse de publier un document sous le plancher, et le dit avec le nom du fichier.

Note: *Une campagne de régénération se contrôle document par document: kit_check_markup.py compare le marquage entre le fichier source et le document régénéré. Sur un parc entier, ce contrôle a déjà attrapé un document parfaitement valide dont le contenu était celui d'un autre.*

5 Session d'absorption

Une fois l'*arbre du projet* à jour, ouvrir un nouveau fil de travail et coller le texte ci-dessous en premier message.

Ce texte déclenche l'absorption structurée de la mise à jour: relecture de toutes les sources de vérité, détection des écarts de version entre le *Registry* et les fichiers réels, application du protocole §4.4 sur vos fichiers projet existants, et proposition des migrations nécessaires une par une. Aucune modification de vos fichiers n'intervient sans votre confirmation explicite.

```
ABSORPTION D'UNE MISE À JOUR DU KIT
===================================

CONTEXTE :
------------------------------

L'arbre du projet porte les fichiers Kit mis à jour, Registry
compris. Aucun fichier propre au projet ne doit être modifié
sans confirmation explicite.

LECTURE OBLIGATOIRE — ORDRE STRICT :
------------------------------

Avant toute autre action, lire dans cet ordre :

1. Kit - Registry.js
   — source de vérité des versions actives.
2. Kit - Projet - Prompt (TS).docx
   — instructions techniques permanentes ; peut contenir
     de nouvelles règles.
3. Kit - Projet - Guide de mise à jour (TS).docx
   — ce guide, notamment §4.4 protocole d'adaptation.
4. Kit - Projet - Structure commune (TS).docx
   — contrat normatif des fichiers projet, référence du
     diagnostic d'écart.
```

```
5. Kit - Stylesheet - General - Code.js + Reference.docx
   — couplet principal.
6. Les autres couplets stylesheet (Glossary, YAML, HTML)
   — Code.js + Reference.docx à chaque fois.

DÉTECTION DE VERSION — RAPPORT OBLIGATOIRE :
-----------------------------

Pour chaque feuille de style, comparer :
- version déclarée dans Kit - Registry.js
- version lue en tête du fichier .js
- constante de version exportée par le fichier .js
- horodatage du couplet (.js et Reference .docx)

Rapport attendu pour chaque feuille de style :
- version Registry     : x.xx
- version en-tête .js  : x.xx
- constante exportée   : x.xx
- horodatage couplet   : AAAA-MM-JJ - HHhMM
- cohérent : oui / non

DIAGNOSTIC D'ÉCART SUR LES FICHIERS PROJET :
-----------------------------

Après la lecture Kit complète, comparer la structure des
fichiers projet existants avec la structure normative de
Kit - Projet - Structure commune. Pour chaque écart, signaler
sous la forme :

    « Écart détecté dans [fichier] : [description].
      Migration proposée : [action]. »

Lister tous les écarts avant d'appliquer la moindre migration.
Ne rien modifier tant que le rapport complet n'a pas été livré.

CONDITIONS D'ARRÊT :
-----------------------------

Si l'une des conditions suivantes est réunie, arrêt complet
et signalement explicite :

- Kit - Registry.js absent ou non à jour
- un fichier .js Kit présent sans sa Reference .docx,
  ou l'inverse, ou avec des horodatages désaccordés
- version Registry et version du fichier .js divergentes
- un fichier Kit attendu absent de l'arbre de travail
- un fichier projet existant contredit un contrat de
  Structure commune de manière ambiguë

APRÈS RAPPORT COMPLET — ATTENDRE LE GO :
-----------------------------

Aucune modification d'un fichier projet existant ne s'opère
sans confirmation explicite, migration par migration. Aucun
fichier projet n'est supprimé de manière autonome, quelle que
soit la raison.
```

```
Pour les fichiers optionnels absents, demander explicitement
s'il faut les créer. Jamais de création silencieuse.
```

Après le rapport de lecture et le diagnostic d'écart, répondre “go” pour chaque migration que vous validez, ou “skip” pour celles que vous souhaitez reporter. Seules les migrations validées sont appliquées.

6 Cas particulier — changement de langue

Si vous souhaitez changer la langue de documentation de votre projet, une seule action est nécessaire: modifier le champ de langue dans votre *Registry*, qui en est la source unique de vérité, et ranger le fichier mis à jour dans l’arbre.

Tous les documents produits à partir de cette session utiliseront la nouvelle langue. Les documents existants ne sont pas modifiés. Le renommage de vos fichiers pour refléter les catégories et sujets de la nouvelle langue est décrit dans *Convention de nommage* §5.4.

À ne pas confondre avec la publication d'un document en plusieurs langues, qui est une décision par document et n'affecte pas la langue du projet — *Convention de nommage* §2.4.

7 Ce qui ne change jamais

Certains éléments de votre projet sont entièrement isolés des mises à jour du Kit.

Note: *“Kit-X - Registry.js” et “Kit - Registry.js” se ressemblent dans une liste de fichiers, et seul le second est remplacé. Relisez ce que vous êtes sur le point d’écraser avant de valider: une sélection trop large emporte votre extension avec le Kit. Sauvegarder vos fichiers Kit-X avant chaque mise à jour coûte une minute.*

Élément	Impact d'une mise à jour du Kit
Vos fichiers projet	Aucun — jamais touchés, modifiés uniquement sur instruction explicite.
Vos scripts de *génération* existants	Non réutilisables — le script est réécrit from scratch à chaque session.
Vos documents de contenu déjà produits	Aucun — ce sont des fichiers statiques.
Votre glossaire et vos *modules de données*	Aucun — fichiers projet, présence conditionnée par vos flags.
Vos conventions spécifiques au projet	Aucun — elles vivent dans la section dédiée de votre *prompt* projet.
Vos fichiers Kit-X	Aucun — le préfixe Kit-X est le vôtre. Ce sont vos extensions du Kit: stylesheets supplémentaires, palette de couleurs, outils. Ils valent pour tous vos projets et vous suivent de l’un à l’autre.