Kit de documentation

Projet — Structure commune — FR

1 Introduction

Ce document définit la structure obligatoire de tout *projet consommateur* du Kit de documentation. Il fait foi pour les fichiers que le projet doit posséder, la forme normative de chaque fichier de données, et les règles de cycle de vie qui gouvernent leur évolution.

Deux situations sont couvertes par le même contrat: l'initialisation d'un nouveau projet et la mise en conformité d'un projet existant après une évolution du Kit. Dans les deux cas, les fichiers projet sont créés ou régénérés depuis les squelettes fournis ci-dessous.

L'audience est l'utilisateur actif du Kit qui gère un ou plusieurs *projets consommateurs*. L'*IA* lit ce document en début de session de travail projet pour valider la conformité avant toute *génération*.

2 Inventaire des fichiers projet

Tableau normatif des fichiers qu'un projet actif doit posséder. La colonne Flag indique le nom du drapeau dans [Préfixe] - Registry.requires; la colonne Défaut indique la valeur par défaut lors de l'initialisation d'un projet. Un fichier sans flag est obligatoire sans exception.

Fichier projet	Rôle	Flag	Défaut
[Préfixe] - Projet - *Prompt* (TS).docx	Instructions techniques spécifiques au projet	—	Obligatoire
[Préfixe] - *Registry.js*	Source de vérité projet — versions, flags, TS	—	Obligatoire
[Préfixe] - Settings.js	Réglages propres au projet — hors périmètre du Kit	—	Si besoin
[Préfixe] - Projet - Todos (TS).docx	Entrées ouvertes, dettes et registre daté des décisions	—	Obligatoire
[Préfixe] - Glossary - Terms (TS).js	Module glossaire — source du glossaire et du *pipeline* passe 2	glossary	true

Fichier projet	Rôle	Flag	Défaut
[Préfixe] - Glossaire - Termes - LANG (TS).docx	Glossaire projet — *couplet* avec Terms.js. Un document par langue publiée si le glossaire est traduit. Nommage: Convention de nommage §2.1 et §2.4	glossary	true
[Préfixe] - Brands (TS).js	Module marques — *pipeline* passe 1	brands	false
[Préfixe] - Text Highlights (TS).js	Fragments mis en évidence — *pipeline*, entre marques et glossaire	textHighlights	false
[Préfixe] - Variable Friendly Names (TS).js	Module variables — *pipeline* passe 3	variableNames	false
[Préfixe] - HASS Entities (TS).js	Module entités HASS — *pipeline* passe 4	hassEntities	false
[Préfixe] - Colors (TS).js	Couleurs du projet — remplacent celles du Kit, §6.5	colors	false
[Préfixe] - Document Titles (TS).js	Libellés d'affichage des documents par langue, §6.8	documentTitles	false
[Préfixe] - Documentation - *Cover Sheet* (TS).docx	Constantes *PCL* — *couplet* avec le PNG	coverSheet	true
[Préfixe] - Documentation - *Cover Sheet* (TS).png	*Page de couverture* du *classeur papier*	coverSheet	true
[Préfixe] - Documentation - Reading Guide (TS).docx	Guide de lecture du *classeur papier*	readingGuide	true

Fichier projet	Rôle	Flag	Défaut
[Préfixe] - Documentation - *Pipeline* HTML (TS).docx	Spécification publication HTML du projet	—	Obligatoire

Note: *Un fichier dont le flag vaut false est absent du projet. Les générateurs* ***Node.js*** *lisent Registry.requires au démarrage et sautent les require() et setters correspondants — voir §6.6. Tous les fichiers portent un horodatage dans leur nom, à la seule exception de [Préfixe] - Registry.js, qui est lui-même la source des horodatages. Un document publié en plusieurs langues porte en outre un tag de langue avant son horodatage — Convention de nommage §2.4.*

3 Dépendances Kit partagées

Les stylesheets du Kit — General, Glossary, YAML, HTML — sont des bibliothèques techniques partagées par tous les projets. Ils ne sont jamais copiés dans un projet. Les scripts générateurs projet les référencent directement par require() dans leur emplacement Kit.

Le fichier Kit - *Registry.js* est la source unique des versions actives des stylesheets. Les projets le lisent pour détecter une mise à jour Kit et savoir si une régénération de *couplet* projet s'impose. Aucun projet ne stocke localement la version d'un *stylesheet* — la vérité est toujours dans Kit - *Registry.js* au moment de la *génération*.

Le Kit dispose également de son propre glossaire Kit - Glossaire.docx, distinct du glossaire projet. Un projet porte ses propres termes, dans son module de termes: les générateurs chargent celui du projet, jamais celui du Kit.

Un troisième groupe peut apparaître dans le répertoire de travail: les fichiers préfixés Kit-X. Ce sont les extensions écrites par l'utilisateur du Kit — stylesheets supplémentaires, palette de couleurs, registre de ces stylesheets. Ils ne sont ni des fichiers du Kit, remplacés à chaque mise à jour, ni des *fichiers du projet*, propres à lui: ils appartiennent à leur auteur et le suivent d'un projet à l'autre.

- **Un projet les lit sans les posséder:** un générateur peut importer un *stylesheet* Kit-X comme il importe ceux du Kit. Le *Registry* du projet n'en déclare pas les versions — celles-ci vivent dans Kit-X - *Registry.js*. Nommage: *Convention de nommage* §3.3. Recette d'écriture: *Étendre le Kit*.

4 Prompt projet

Le fichier [Préfixe] - Projet - *Prompt* (TS).docx contient les instructions techniques spécifiques au projet. Il complète le *Prompt* — jamais ne le duplique. Il est lu par l'*IA* en début de chaque session de travail sur le projet.

Variables à personnaliser: préfixe du projet, project.documentAuthor et project.webAuthor (noms affichés dans le pied de page du document et de la page), project.docLanguage (FR, EN, DE ou LU — défaut EN pour les *projets consommateurs*, source de vérité unique dans [Préfixe] - *Registry.js* section

project), conventions rédactionnelles propres au domaine du projet, éventuelles règles narratives spécifiques.

Les règles techniques communes à tous les projets — formatage, protocoles session, convention de nommage, chaîne de validation — vivent dans *Prompt* et ne sont pas reproduites dans le *prompt* projet. Le *prompt* projet ne contient que ce qui diffère d'un projet à l'autre.

5 Registry projet

Chaque projet dispose de son propre [Préfixe] - *Registry.js*. Source de vérité pour l'identité, les flags de fichiers requis, les domaines famille, les timestamps des documents générés, les flags de rendu, la mention de compilation web et les métadonnées de déploiement du site HTML.

5.1 Rôle et contenu

Le *Registry* projet tient les sections obligatoires ci-dessous, et une section optionnelle. Chacune a une règle de mise à jour propre — voir §5.3.

- **project:** identité et langue du projet. Champ project.docLanguage (FR, EN, DE, LU) — source de vérité unique de la langue, lue par les stylesheets pour la sélection *L10N* et par le *renderer Cover Sheet* pour les *PCL* strings localisées. projectShareable, à false, dit si le projet est offert en reprise: à true, la passe de publication produit l'archive du projet et l'icône de téléchargement apparaît. allowNonKitTemplates, à false, ouvre le droit de créer des modèles hors des formes du Kit — *Étendre le Kit* §9. Trois drapeaux booléens s'y ajoutent aussi, à false par défaut: autoAddGlossary, autoAddTextHighlights et autoAddBrands. À false, l'*IA* signale un candidat et attend la confirmation; à true, elle l'ajoute au module concerné et le signale en fin de section — *Kit Prompt* §13.2.
- **requires:** flags de présence des fichiers optionnels — coverSheet, readingGuide, glossary, textHighlights, documentTitles, brands, variableNames, hassEntities, colors. Lu par les générateurs pour conditionner require() et *setters*.
- **familyDomains:** tableau de hostnames sans schéma. Domaines famille dont les liens, et ceux de leurs sous-domaines, restent capturés dans la *WebView* de l'app *Android*. Tout autre domaine est routé vers le navigateur système. Lu par style.setFamilyDomains() en tête de gen-[projet]-site.js. Section stable — modifiée uniquement lors d'un changement de périmètre famille.
- **documents:** mapping du slug vers son *horodatage*, pour chaque document généré du projet. Mis à jour à chaque livraison de document — le TS reflète la dernière régénération. Un document publié en plusieurs langues possède une entrée par variante, dont le slug porte le suffixe de langue: philosophy-fr, philosophy-de. Le *déploiement delta*

compare par slug, donc chaque langue se republie indépendamment. La clé piece, facultative, nomme le fichier d'une pièce d'annexe: il vit à la racine du projet, la passe le copie sous assets/pieces/ du site, sous son nom d'origine, et le bouton de téléchargement de la page le sert à la place du PDF du document — *Pipeline HTML* §6.3. Deux clés facultatives s'y ajoutent, écrites par le générateur de site: contentTs, l'*horodatage* du dernier changement de texte, et contentHash, l'*empreinte* du texte qui l'a produit. Un *Registry* qui ne les porte pas fonctionne; le générateur les pose à la première passe.

- **rendering:** flags pilotant la *génération* de livrables. Sous-sections coverSheet.sectionHMode (FIXED ou DYNAMIC) consommée par kit_render_cover_sheet.py, et toc.generate / toc.hidden pilotant la TOC HTML latérale via renderDocument et getCSS. La section est facultative — le *renderer* applique un fallback gracieux FIXED en son absence. La déclarer explicitement reste recommandé pour rendre le choix lisible. languageSelector s'y règle aussi — §15 —, et indexIcon déclare l'illustration de la page d'accueil, quatrième icône du projet avec favicon.svg, favicon.ico et apple-touch-icon.png, toutes quatre à sa racine. homeHref donne la cible de la maison de la seule page d'accueil, qui sort du *sous-site*; la maison d'une page de document ramène toujours au sommaire de sa langue, sans réglage —, ainsi qu'indexIcon: le projet qui a posé index-icon.svg à sa racine le déclare ici, et la page d'accueil affiche l'image à gauche du sous-titre. Absent, rien ne change.
- **compiledWith:** mention de compilation affichée dans le pied de page des pages du *sous-site*. Objet indexé par code langue, transmis au *stylesheet* HTML par le générateur de site via config.compiledWith. Bloc absent, null, ou chaîne vide pour une langue: aucune mention n'est rendue. Ce champ ne pilote que le web — aucun document.docx produit par le Kit ne porte de mention de compilation, le *stylesheet* General ayant supprimé la clé *L10N* correspondante.
- **deploy:** métadonnées du *sous-site* HTML — typiquement lastDeploy. Mis à jour par gen-[projet]-site.js lors d'un déploiement complet ou delta.
- **stylesheets:** OPTIONNELLE. Versions et horodatages des stylesheets écrits par le projet, sur le *modèle* de la rubrique du même nom dans Kit - *Registry.js*. Absente si le projet n'en écrit aucun. Elle ne liste que les stylesheets du projet: ceux du Kit restent déclarés dans le *Registry* du Kit, source unique de leurs versions — §3. Recette d'écriture: *Étendre le Kit*.

5.2 Structure normative

Squelette à instancier lors de l'initialisation d'un projet. Les commentaires rappellent la sémantique de chaque section. Le fichier n'a pas de *timestamp* dans son nom — il est lui-même la source des timestamps.

```
'use strict';
// ==============================
// [Préfixe] - Registry.js — Source de vérité du projet
//
// Versions stylesheet Kit utilisées : lues depuis Kit - Registry.js.
// Ce fichier projet ne duplique pas les versions Kit.
// ==============================

module.exports = {

  // ---- Identité et langue ---------------------------------
  // project.docLanguage pilote la sélection L10N des stylesheets
  // General/Glossary/HTML et les constantes PCL du Cover Sheet.
  project: {
    docLanguage: 'EN',  // 'EN' (défaut consommateurs) | 'FR' | 'DE' | 'LU'
    documentAuthor:   'Prénom Nom',          // pied de page des .docx et PDF
    webAuthor:      'Prénom Nom',        // pied des pages web — '' : site non
signé
    documentSiteBase:  'https://[sous-site]',  //  racine  des  renvois  entre
documents
    projectShareable:      false,  // projet offert en reprise
    allowNonKitTemplates:  false,  // Étendre le Kit §9
    autoAddGlossary:       false,  // Kit Prompt §13.2
    autoAddTextHighlights: false,
    autoAddBrands:         false,
  },

  // ---- Fichiers requis — flags ----------------------------
  // Règle : si flag = false, le fichier correspondant est absent
  // du projet et le générateur saute le require() + setter().
  requires: {
    coverSheet:    true,   // défaut true
    readingGuide:  true,   // défaut true
    glossary:      true,   // défaut true
    textHighlights: false, // défaut false
    documentTitles: false, // défaut false
    brands:        false,  // défaut false
    variableNames: false,  // défaut false
    hassEntities:  false,  // défaut false
    colors:        false,  // défaut false
  },

  // ---- Domaines famille — routage hyperliens HTML ---------
  // Liens vers ces domaines (et sous-domaines) restent dans la
  // WebView de l'app Android famille. Liens externes ouvrent
  // dans le navigateur via target="_blank" rel="noopener".
  // Lu par style.setFamilyDomains() en tête de gen-[projet]-site.js.
  familyDomains: ['sliver.lu', 'hexi.lu'],

  // ---- Documents projet — timestamp dernière génération ---
  documents: {
```

```
    // 'slug-document': { ts: 'AAAA-MM-JJ - HHhMM' },
  },

  // ---- Rendering — flags pilotant la génération -----------
  // Lus par les renderers avec fallback gracieux si la section
  // est absente — voir Cover Sheet §3 ligne SECTION_H_MODE.
  rendering: {
    homeHref: null,           // maison de la page d'accueil seule — Structure
commune §15
    languageSelector: 'document', // 'document' (défaut) | 'site' — Structure
commune §15
    indexIcon: false,          // true : index-icon.svg s'affiche sur la page
d'accueil
    coverSheet: {
      sectionHMode: 'FIXED',  // FIXED (défaut) | DYNAMIC
    },
    toc: {
      generate: true,    // false -> bloc <nav class="toc"> absent du HTML
      hidden:   false,   // true  -> CSS .toc { display: none; } injecté
    },
  },

  // ---- Mention de compilation — pied de page HTML ---------
  // Rendue dans le pied de page des pages du sous-site, transmise
  // par gen-[projet]-site.js via config.compiledWith.
  // Objet indexé par code langue. null, bloc absent ou chaîne vide
  // pour une langue : aucune mention rendue.
  // Ne concerne JAMAIS les .docx — voir §5.1.
  compiledWith: null,
  // Exemple si le projet souhaite l'afficher :
  //   compiledWith: {
  //     FR: 'Compilé avec Claude AI',
  //     EN: 'Compiled with Claude AI',
  //     DE: 'Kompiliert mit Claude AI',
  //     LU: 'Kompiléiert mat Claude AI',
  //   },


  // ---- Archive de telechargement --------------------------
  // Present si le projet publie son archive ; absent sinon.
  projectZip: {
    version:  '0.01',            // monte avec la structure ou les regles
    filename: '[sous-site]-0.01.zip',
    updated:  'AAAA-MM-JJ',      // suit le numero de version
  },
   // ---- Déploiement site HTML ------------------------------
  deploy: {
    siteName: '[sous-site]',       // nomme l'archive et les pages d'index
    siteInfrastructure: 'parent',  // icônes, robots.txt et .htaccess du site
parent
    lastDeploy: null,  // ISO 'AAAA-MM-JJThh:mm:ss' — écrit par le générateur
  },

};
```

Note: *Le squelette a été élargi pour intégrer les sections project, familyDomains, rendering et compiledWith — historiquement absentes alors qu'elles étaient consommées par les*

stylesheets et le renderer Cover Sheet. Un projet conforme à l'ancien squelette pouvait crasher sur kit_render_cover_sheet.py par KeyError. La protection est désormais double: squelette complet ici, et fallback gracieux côté renderer.

5.3 Cycle de vie

La section project est stable — modifiée uniquement si le projet change de langue principale, cas rare et explicite documenté au *Guide de mise à jour* §6.

La section requires est stable sur la durée de vie du projet — modifiée uniquement lors de l'ajout ou du retrait d'une dépendance fonctionnelle. Un projet qui commence à documenter *Home Assistant* bascule hassEntities de false à true et crée le module correspondant.

La section familyDomains est stable — modifiée uniquement lors d'un changement de périmètre famille.

La section documents est mise à jour à chaque livraison d'un document — le script générateur du document, ou l'utilisateur après réception, inscrit le nouveau TS.

La section rendering est stable — modifiée pour basculer entre FIXED et DYNAMIC sur le *Cover Sheet*, cas rare, ou pour basculer la TOC HTML. Norbert ne modifie jamais *Registry.js* manuellement: il demande à l'*IA* dans le projet respectif, qui effectue la bascule.

La section compiledWith est stable — elle relève d'une décision éditoriale du projet, pas d'un état technique. La modifier n'impose aucune régénération de document: la valeur est lue au moment de la *génération* du site.

La section deploy.lastDeploy est écrite exclusivement par gen-[projet]-site.js lors d'un déploiement. Aucun autre script ni l'utilisateur ne l'éditent manuellement.

5.4 Réglages propres au projet — Settings.js

Le *Registry* du projet ne porte que les blocs que le Kit spécifie: project, requires, stylesheets, familyDomains, documents, rendering, compiledWith, projectZip, deploy. Un projet y renseigne des valeurs, il n'y ajoute ni clé ni bloc. Ce qui lui appartient vit dans un fichier distinct, [Préfixe] - Settings.js, que le Kit ne spécifie ni ne lit.

- **Une frontière de fichier, non de discipline.** Un bloc propre logé dans le *Registry* oblige à juger, au cas par cas, si son emplacement est légitime. Dans un fichier séparé, une clé inconnue du *Registry* est fautive par construction, et un contrôle peut le dire sans arbitrer.
- **Documenté dans le Prompt projet.** Chaque rubrique de Settings.js y porte son nom, ses clés, l'effet de chacune et l'effet de son absence. Une clé non documentée est une clé perdue à la session suivante.

- **Lu avec un arrêt bruyant.** Le générateur qui lit un réglage échoue en le nommant s'il manque ou n'a pas le type attendu, plutôt que de retomber sur un défaut silencieux.

Note: *kit_check_registry.py refuse toute clé de premier niveau absente de la liste du Kit. Il s'exécute avant toute génération — Quality Control §3. La séparation ne vaut que si quelque chose la vérifie: sans contrôle, elle serait une seconde convention aussi facultative que la première.*

Note: *La mise à jour du Kit ne touche ni le Registry du projet ni son Settings.js — Guide de mise à jour. Aucun mécanisme ne les protège: c'est la procédure qui ne les remplace pas, et c'est pourquoi elle énumère ce qu'elle remplace.*

Note: *Les extensions Kit-X sont autre chose: elles appartiennent à leur auteur, suivent d'un projet à l'autre, et déclarent leurs versions dans Kit-X - Registry.js — §3.*

6 Modules de données

Le projet peut posséder des *modules de données* — chacun activé conditionnellement par un flag dans Registry.requires. Les règles structurelles de chaque module sont normatives et ne peuvent pas être interprétées. Tout écart entre un fichier existant et la structure ci-dessous constitue une anomalie à corriger avant toute *génération*.

Cette section est le domicile unique du contrat des *modules de données*. Les autres documents du Kit y renvoient sans le redire.

6.1 Glossary Terms

Fichier: [Préfixe] - Glossary - Terms (TS).js. Flag Registry.requires.glossary, défaut true. *Couplet* avec le ou les documents glossaire — tous les membres partagent le même *timestamp* à toute régénération. Nommage des documents: *Convention de nommage* §2.1 et §2.4.

Le module est chargé en tête de chaque script générateur et nourrit la passe 2 du *pipeline* — les termes sont rendus automatiquement en italique teal dans tout le texte courant.

Champs obligatoires par entrée: chaque élément de glossaryEntries porte quatre champs dont trois sont requis.

Champ	Type	Obligatoire	Description
balise	string	Oui	*Ancre* HTML stable — _slugify du terme dans la langue d'origine, unique dans le fichier. Hors langue et jamais modifiée une fois publiée: elle identifie le concept, pas le mot
term	string ou objet	Oui	Terme canonique. Chaîne si le glossaire est monolingue, objet indexé par code langue sinon
aliases	tableau ou objet	Non	Formes alternatives pointant vers la même *balise*. Tableau si monolingue, objet indexé par code langue sinon
definition	string ou objet	Oui	Définition en langage courant, jamais de *Markdown*. Chaîne si monolingue, objet indexé par code langue sinon

Un projet qui publie son glossaire en plusieurs langues indexe term, aliases et definition par code langue. La *balise*, elle, ne change jamais: elle est dérivée du terme dans sa langue d'origine et sert d'*ancre* publique. C'est ce qui permet au sélecteur de langue de passer d'une page à l'autre en conservant la position du lecteur — la variante allemande d'un document pointe sur la même *ancre* que la française.

Les deux formes cohabitent sans migration. Une chaîne signifie monolingue, un objet signifie multilingue. Un projet bascule terme par terme, au moment où il traduit.

- **Tri par langue:** l'ordre alphabétique des entrées se calcule sur le terme dans la langue rendue. Les glossaires de deux langues n'ont donc pas le même ordre, ce qui est correct pour un glossaire.
- **Champ manquant — échec bruyant:** si une langue publiée n'a pas de valeur pour term ou definition, le générateur s'arrête et nomme la *balise* fautive. Pas de repli silencieux sur la langue du *Registry*: un glossaire amputé sans avertissement est exactement le genre de dégât que le Kit combat.
- **Aliases traduits:** les alias d'une langue alimentent la détection du *pipeline* dans les documents de cette langue. Sans alias allemands, aucun terme ne se colore en teal dans un document allemand — le *pipeline* devient muet là où on l'attend.
- **Terme de glossaire seul:** le champ facultatif detection: false garde l'entrée dans le glossaire, sur papier et sur le site, sans qu'elle nourrisse les passes de détection — pour un mot courant qu'on veut définir sans le colorer partout. buildSearchTerms l'écarte.
- **Composés à trait d'union:** un composé dont la seconde partie commence par une minuscule — zone-based, Tasmota-flashed — n'est reconnu que s'il est déclaré en alias du terme. Sans cela, la frontière le traite comme un nom de fichier ou de paquet, qu'elle protège: python-docx. Une seconde partie à majuscule reste une limite de mot — HTML-*Stylesheet*.

Exports obligatoires: glossaryEntries, tableau d'objets, et buildSearchTerms(LANG), fonction retournant les paires surface et anchor triées longest-first pour la langue demandée. Un glossaire monolingue peut conserver l'export historique glossarySearchTerms.

Squelette à instancier lors de l'initialisation ou lors d'une absorption de règle Kit qui élargirait le contrat.

```
'use strict';
// ==============================
// [Préfixe] - Glossary - Terms (TS).js
// Source de vérité du glossaire projet.
// Couplet avec [Préfixe] - Glossaire (TS).docx — même timestamp.
// Projet multilingue : un Glossaire par langue publiée, tous au
// même timestamp que ce fichier.
// ==============================

const glossaryEntries = [

  // --- Forme monolingue : term, aliases et definition sont
  //     des chaînes. Forme historique, toujours valide.
  // {
  //   term:       'Terme canonique',
  //   aliases:    ['alias 1', 'alias 2'],
  //   balise:     'terme-canonique',       // _slugify(term), unique
```

```
  //   definition: 'Définition en langage courant, sans Markdown.',
  // },

  // --- Forme multilingue : term, aliases et definition sont
  //     indexés par code langue. La balise reste hors langue.
  // {
  //   balise: 'hallucination',             // ancre publique, invariante
  //   term: {
  //     FR: 'Hallucination',
  //     DE: 'Halluzination',
  //   },
  //   aliases: {
  //     FR: [],
  //     DE: ['Konfabulation'],
  //   },
  //   definition: {
  //     FR: 'Phénomène par lequel un modèle génère...',
  //     DE: 'Phänomen, bei dem ein Modell...',
  //   },
  // },

];

// Construction longest-first de la liste de recherche.
// LANG est la langue du document en cours de génération.
// Une chaine et un tableau valent pour toutes les langues — forme
// monolingue, Structure commune §6.1. Sans ce test, v[LANG] rend
// undefined sur un tableau d'alias et les alias disparaissent sans
// message. 2026-09-18.
const pick = (v, LANG) =>
  (v === undefined || v === null) ? undefined
  : (typeof v === 'string' || Array.isArray(v)) ? v
  : v[LANG];

function buildSearchTerms(LANG) {
  const out = [];
  glossaryEntries.forEach(e => {
    if (e.detection === false) return;  // terme de glossaire seul
    const term = pick(e.term, LANG);
    if (term === undefined) {
      throw new Error(`Glossaire : terme absent en ${LANG} pour la balise ${
e.balise}`);
    }
    out.push({ surface: term, anchor: e.balise });
    const al = e.aliases ? pick(e.aliases, LANG) : [];
    (al || []).forEach(a => out.push({ surface: a, anchor: e.balise }));
  });
  out.sort((a, b) => b.surface.length - a.surface.length);
  return out;
}

module.exports = { glossaryEntries, buildSearchTerms };
```

– **glossaryRubriques.** Les rubriques du glossaire: numéro, titre et accroche par langue. Le champ rubrique de chaque terme y renvoie. Un titre de rubrique est du contenu de glossaire, pas une décision de rendu

— l'écrire dans un générateur le mettrait hors de portée de l'auteur et le ferait diverger entre le papier et le web.

- **glossaryDocument.** L'identité du document et ses textes d'encadrement: titre du projet, auteur, langue faisant autorité, segments du nom de fichier par langue de projet, et par langue de rendu l'accroche, la note d'ouverture, l'introduction, la note finale et la note de traduction. L'*artefact* qui produit le glossaire ne connaît donc ni le nom du projet ni sa langue: il les lit ici.

Note: *Le champ nomBase suit Convention de nommage §2.4: catégorie et sujet restent dans la langue du projet, seul le tag final marque la langue du contenu. Un projet anglophone produit "AI - Glossary - Terms - EN", pas "AI - Glossaire - Termes - EN".*

6.2 Brands

Fichier: [Préfixe] - Brands (TS).js. Flag Registry.requires.brands, défaut false. Fichier absent si le projet ne documente pas de marques.

Export obligatoire: brandEntries, tableau plat de noms de marques. Rendu small caps bold via la passe 1 du *pipeline* dans tout le texte courant. Ordre recommandé: longest-first pour éviter les recoupements entre marques imbriquées.

```
'use strict';
// ==============================
// [Préfixe] - Brands (TS).js
// Noms de marques du projet — rendus small caps bold (passe 1).
// ==============================

const brandEntries = [
  // 'Nom de marque 1',
  // 'Nom de marque 2',
];

module.exports = { brandEntries };
```

6.3 Variable Friendly Names

Fichier: [Préfixe] - Variable Friendly Names (TS).js. Flag Registry.requires.variableNames, défaut false. Fichier absent si le projet ne documente pas de variables techniques.

Export obligatoire: variableNameEntries. Noms de variables et paramètres techniques. Rendus en italique dark green VARIABLE_NAME_COLOR via la passe 3 du *pipeline*.

```
'use strict';
// ==============================
// [Préfixe] - Variable Friendly Names (TS).js
// Noms de variables techniques — italique dark green (passe 3).
// ==============================
```

```
const variableNameEntries = [
  // 'MA_VARIABLE',
  // 'autreNomVariable',
];

module.exports = { variableNameEntries };
```

6.4 HASS Entities

Fichier: [Préfixe] - HASS Entities (TS).js. Flag Registry.requires.hassEntities, défaut false. Fichier absent si le projet ne documente pas *Home Assistant*.

Export obligatoire: hassEntityEntries. Noms d'entités *Home Assistant*. Rendus en italique violet HASS_ENTITY_COLOR via la passe 4 du *pipeline*.

```
'use strict';
// ==============================
// [Préfixe] - HASS Entities (TS).js
// Entités Home Assistant — rendues italique violet (passe 4).
// ==============================

const hassEntityEntries = [
  // 'light.salon',
  // 'sensor.temperature_salon',
];

module.exports = { hassEntityEntries };
```

6.5 Couleurs

Fichier: [Préfixe] - Colors (TS).js, pour les couleurs propres à ce projet. Un second niveau existe hors projet — Kit-X - Colors (TS).js — pour celles qui suivent leur auteur d'un projet à l'autre; il n'est piloté par aucun drapeau, il est présent ou il ne l'est pas. Flag Registry.requires.colors, défaut false. Fichier absent si le projet garde les couleurs du Kit.

Export obligatoire: colorEntries, objet à deux tables — docx pour les documents, html pour les pages web. Chaque table ne porte que les clés remplacées; une clé absente garde le défaut du *stylesheet*. Les noms de clés et leurs défauts sont tabulés dans les Reference des deux stylesheets.

```
'use strict';
// ==============================
// [Prefixe] - Colors.js
// Couleurs du projet — remplacent celles du Kit.
// ==============================

// Deux tables distinctes : le contraste sur fond blanc imprime ne se
// regle pas comme le contraste sur ecran.
const colorEntries = {
  docx: { HEADING_COLOR: '8B0000' },
  html: { HEADING_COLOR: '#8B0000', CODE_BG: '#FAFAFA' },
};

module.exports = { colorEntries };
```

Note: *Trois niveaux se superposent, du plus général au plus particulier: les défauts du stylesheet, puis Kit-X - Colors.js s'il existe, puis celui du projet. Chaque niveau ne redéfinit que ce qu'il veut changer. Une palette placée dans Kit-X suit l'utilisateur d'un projet à l'autre; une palette de projet ne vaut que pour lui.*

6.6 Appel conditionnel en tête de générateur

Tout script générateur projet charge *Registry* au démarrage et sollicite chaque *module de données* uniquement si son flag est activé. Ce pattern est la contrepartie technique directe des flags Registry.requires — il remplace la règle antérieure selon laquelle les *setters* étaient appelés systématiquement.

```
// ---- Tête de générateur projet — chargement conditionnel ----
const style    = require('./Kit - Stylesheet - General - Code.js');
const Registry = require('./[Préfixe] - Registry.js');

// Glossaire — défaut true
if (Registry.requires.glossary) {
  const { glossarySearchTerms } = require('./[Préfixe] - Glossary - Terms
(TS).js');
  style.setGlossaryTerms(glossarySearchTerms);
}

// Marques — défaut false
if (Registry.requires.brands) {
  const { brandEntries } = require('./[Préfixe] - Brands (TS).js');
  style.setBrands(brandEntries);
}

// Variable Friendly Names — défaut false
if (Registry.requires.variableNames) {
  const { variableNameEntries } = require('./[Préfixe] - Variable Friendly
Names (TS).js');
  style.setVariableNames(variableNameEntries);
}

// HASS Entities — défaut false
if (Registry.requires.hassEntities) {
  const { hassEntityEntries } = require('./[Préfixe] - HASS Entities
(TS).js');
  style.setHassEntities(hassEntityEntries);
}
```

Note: *Si un flag vaut false, le fichier correspondant est absent du projet et aucun require() ni appel de setter n'est effectué pour ce module. Les passes pipeline inactives restent silencieuses — les fonctions texte continuent de fonctionner sans erreur pour les autres passes actives. La discipline est auditée par kit_check_setters.py, voir Quality Control §3.4.*

6.7 Protocole de contrôle après une injection Kit

Quand l'utilisateur signale une injection Kit, l'*IA* applique ce protocole aux *modules de données* avant tout autre travail. Il complète le diagnostic

d'écart général du *Guide de mise à jour* §4.4, dont il est la déclinaison propre aux modules.

- **Lire §6 intégralement:** la structure normative peut avoir évolué avec le Kit injecté.
- **Contrôler chaque module présent:** vérifier la présence de tous les champs obligatoires et la conformité des exports au contrat de §6.1 à §6.4.
- **Contrôler Glossary Terms en particulier:** présence du champ *balise* sur chaque entrée, et format surface et anchor sur glossarySearchTerms. Un projet issu d'un Kit antérieur peut porter un format ancien où la *balise* est absente et où glossarySearchTerms expose termKey au lieu d'anchor. La migration ajoute la *balise* à chaque entrée par _slugify(term), collision-safe, et met l'export à jour.
- **Signaler chaque écart:** sous la forme “Écart détecté dans [fichier]: [description]. Migration proposée: [action].”
- **Attendre la confirmation:** aucune migration n'est appliquée sans accord explicite, migration par migration.

6.8 Module Libellés de documents

Le nom de fichier est un identifiant: il ne change pas d’une langue à l’autre. Le libellé affiché, lui, suit la langue du lecteur — sur la page d’accueil du site, dans un renvoi d’un document à un autre, et dans le sous-titre d’une page de garde. Sans ce module, un lecteur allemand lit des titres français.

Fichier: [Préfixe] - Document Titles (TS).js, piloté par le drapeau requires.documentTitles. Il exporte les tables suivantes.

- **titleEntries.** Une entrée par document, clé = nom de fichier sans *horodatage* ni suffixe de langue. Valeurs: le sujet en FR, DE, EN et LU; alias, formes courtes employées dans le corpus; aliasSection, formes d’un seul mot qui ne comptent que devant un renvoi de section; slug, identifiant de la page HTML ou null si le document n’est pas publié; multilingue, vrai si le slug porte un suffixe de langue.
- **categoryLabels.** Les catégories de nom de fichier traduites — la table de *Convention de nommage* §5.1 rendue exécutable. Elle sert à composer le libellé complet d’un document et le sous-titre de sa page de garde.
- **sectionTitles.** Les titres d’*intercalaire* du classeur, par numéro et par langue. Le sommaire de la *page de couverture* et l’index par langue du site les lisent ici. Ils vivaient dans le générateur de site, hors de portée du *renderer* de couverture, qui affichait donc des titres français dans un classeur allemand.

- **coverHeader.** Les deux lignes du bandeau de la *page de couverture*, par langue. Elles viennent des *constantes de rendu* du .docx, qui n'en porte qu'une version: un rendu allemand affichait donc un bandeau français.
- **Le champ lecture de titleEntries.** Le nom d'un document tel qu'on l'écrit dans une phrase — Le guide de mise à jour. Saisi à la main, langue par langue: un article et parfois une préposition ne se dérivent pas du sujet. Le sommaire du classeur et l'index du site l'emploient; le nom de fichier reste la forme technique.

Une valeur vide fait retomber le libellé ET l'adresse sur la langue du projet: servir un libellé français en pointant une page qui n'existe pas serait pire que ne pas traduire. La règle de peuplement: tout document de la langue du projet y figure.

Note: *Le générateur passe titleEntries et categoryLabels au stylesheet par setDocumentTitles, comme pour la langue, le glossaire et les couleurs: ce sont les deux seules dont il ait besoin, puisqu'il compose des libellés de documents. sectionTitles et coverHeader ne servent qu'au renderer de la page de couverture et au générateur de site, qui lisent le module directement. Le stylesheet n'importe aucun fichier de données.*

6.9 Text Highlights — mise en évidence

[Préfixe] - Text Highlights (TS).js exporte highlightEntries, une liste plate de fragments. Chaque fragment déclaré est rendu en italique à toutes ses occurrences, dans tous les documents du projet, sans marquage. Drapeau textHighlights, défaut false.

- **Ce qui y entre.** Une entité qui en est une à chaque occurrence et qui n'est ni une marque ni un terme du glossaire: un journal, une organisation, un nom de *modèle* d'*intelligence artificielle*, un logiciel tiers.
- **Précédence.** La passe se place après les marques et avant le glossaire. Un fragment déjà porté par l'une de ces tables garde son rendu: une entrée en double ne produit rien de plus et priverait un terme de son lien.

Note: *La passe marque toutes les occurrences. Une insistance dont la portée dépend de la phrase n'a donc rien à faire dans cette table: elle s'écrit ainsi à l'endroit voulu — General Reference §5.5 et §5.6.*

7 Glossaire projet

Le document glossaire est la face publique du glossaire projet. Il forme un *couplet* avec [Préfixe] - Glossary - Terms (TS).js — les deux fichiers partagent le même *timestamp* et sont toujours régénérés ensemble.

Le glossaire est toujours régénéré depuis Terms.js, jamais l'inverse. Le fichier .docx est un rendu visuel du module.js — toute modification de contenu, ajout, retrait ou

modification d'un terme, passe d'abord par le .js, puis la régénération du .docx suit mécaniquement avec un *timestamp* frais commun.

Un projet qui publie son glossaire en plusieurs langues produit un document par langue. Le *couplet* devient alors un module de termes et n documents, tous porteurs du même *horodatage* et régénérés ensemble. Ajouter un terme, ou corriger une seule langue, impose donc de régénérer l'ensemble. Le nommage de ces documents relève de *Convention de nommage* §2.1 et §2.4: catégorie et sujet dans la langue du projet, tag de langue après le sujet. Ce document ne le redit pas — une règle a un domicile unique.

Chaque variante possède sa propre entrée dans Registry.documents avec son *horodatage*, si bien que le déploiement HTML ne republie que les pages réellement modifiées.

Les fichiers sont absents du projet si Registry.requires.glossary vaut false. Le *pipeline* de formatage fonctionne sans eux — aucun terme n'est italicisé en teal dans ce cas.

8 Cover Sheet projet

Couplet [Préfixe] - Documentation - *Cover Sheet* (TS).docx et [Préfixe] - Documentation - *Cover Sheet* (TS).png — constantes *PCL* et *page de couverture* générée. Flag Registry.requires.coverSheet, défaut true. Les deux fichiers partagent le même *timestamp*.

Le rendu PNG est produit par le *renderer* *Python* kit_render_cover_sheet.py — stateless et déterministe. Toute modification visuelle passe exclusivement par les constantes *PCL* du .docx, jamais par le code du *renderer*. Le contrat complet du *renderer* et l'*inventaire* de toutes les constantes *PCL* sont documentés dans *Cover Sheet*.

8.1 Squelette du .docx

Le fichier [Préfixe] - Documentation - *Cover Sheet* (TS).docx contient les sections normatives du *modèle* Kit. Chaque projet duplique le Kit Cover Sheet.docx puis personnalise uniquement §2 et §3; les autres sections restent identiques au *modèle* Kit.

Section	Contenu	Personnalisation projet
§1 Objet	Description du rôle du *Cover Sheet* — *page de couverture* du *classeur papier* projet	Aucune — copie conforme du *modèle* Kit
§2 Organisation du classeur	Tableau des rubriques du projet — numéro, titre *intercalaire*, documents reliés, Tab	Personnalisation complète — voir §8.2

Section	Contenu	Personnalisation projet
§3 *PCL* — Constantes spécifiques au projet	Tableau des constantes *PCL* personnalisées par le projet	Voir §8.3 — typiquement HEADER_TEXT seul
§4 Mise à jour de ce document	Règles de mise à jour du *couplet* — TS frais commun au .docx et au .png	Aucune — copie conforme du *modèle* Kit

8.2 Personnalisation §2 — rubriques du projet

Le tableau §2 du *Cover Sheet* projet liste les rubriques du projet, pas celles du Kit. Le nombre de rubriques numérotées varie librement d'un projet à l'autre, dans la limite de TAB_COUNT, qui vaut 12 onglets.

TAB_NUMBERED_MAX n'est pas une limite à respecter mais une valeur dérivée: le *renderer* la calcule automatiquement en comptant les rubriques actives du §2. La ligne *PCL* correspondante n'est utile que pour réserver explicitement des onglets supplémentaires, par exemple cinq rubriques actives mais huit onglets colorés pour préparer trois rubriques futures.

Pour chaque rubrique: numéro d'*intercalaire*, titre, liste des documents reliés; la couleur découle du numéro — §11. Les documents listés correspondent aux documents publiés du projet — un document figure dans une seule rubrique.

Note: *La liste des rubriques et leurs documents associés est définie en interactif lors de l'initialisation du projet — voir Guide d'initialisation §5.*

8.3 Personnalisation §3 — constantes de rendu spécifiques

Le tableau §3 du *Cover Sheet* projet liste uniquement les constantes *PCL* personnalisées par le projet. Toutes les autres — dimensions A4 300 dpi, couleurs, marges, polices, positions des onglets — sont héritées du Kit *Cover Sheet* §3 sans modification ni duplication.

En pratique, les seules constantes systématiquement personnalisées sont HEADER_TEXT_1 et HEADER_TEXT_2: les deux lignes affichées dans le bandeau orange de la *page de couverture*. La coupure entre les deux est décidée à l'écriture, pas calculée au rendu — le sujet sur la première ligne, sa qualification sur la seconde. Un texte court peut ne renseigner que HEADER_TEXT_1 et laisser la seconde vide.

Leur valeur suit la convention typographique du projet — typiquement en casse de phrase ou Title Case, par exemple Immobilier — patrimoine et acquisitions, Photogear, Roodt-Hass — domotique. L'usage du tout-majuscules est déconseillé, effet de cri typographique en lecture.

Exception Kit: le Kit affiche “KIT de documentation” en première ligne et “assistée par *intelligence artificielle*” en seconde. L'acronyme KIT est en capitales par signalétique délibérée — le Kit est la référence canonique pour tous les *projets consommateurs*, pas un projet comme un autre. Cette exception ne se transpose pas aux projets.

Si le projet a besoin de personnaliser une autre constante *PCL*, la nouvelle valeur figure dans §3 du *Cover Sheet* projet avec mention explicite “override Kit”.

8.4 Recette de duplication

Lors de l'initialisation d'un nouveau projet ou de l'ajout du *Cover Sheet* à un projet existant, la procédure suit cinq étapes.

- Dupliquer le Kit Cover Sheet.docx vers [Préfixe] - Documentation - *Cover Sheet* (TS).docx avec un *timestamp* frais.
- Conserver §1 Objet et §4 Mise à jour tels quels — pas de modification.
- Réécrire §2 Organisation du classeur avec les rubriques du projet selon §8.2.
- Réduire §3 *PCL* aux seules constantes personnalisées, typiquement HEADER_TEXT. Les autres constantes Kit sont héritées implicitement.
- Générer le PNG via kit_render_cover_sheet.py en passant le .docx projet et le *Registry* projet en arguments. Le *couplet* est livré ensemble.

Note: *Toute modification ultérieure du Cover Sheet projet — ajout de rubrique, changement de HEADER_TEXT, override d'une PCL — régénère le couplet entier.docx et.png avec un nouveau timestamp commun.*

8.5 Cas non requis

Si Registry.requires.coverSheet vaut false, le projet n'a pas de *classeur papier*. Cas rare en pratique — tous les projets documentés à ce jour possèdent leur *Cover Sheet*. Quand le flag est false, le *couplet Cover Sheet* est absent du projet et le *renderer* kit_render_cover_sheet.py n'est pas appelé.

8.6 Relation avec la landing HTML

La structure du *Cover Sheet* papier et celle de la landing HTML du projet sont déclarées séparément. Le §2 du .docx *Cover Sheet* est la source canonique de l'organisation du *classeur papier*; la constante SECTIONS du gen-[préfixe]-site.js est la source canonique de la landing web. Aucun script ne lit l'une pour piloter l'autre. Leur concordance est un choix du projet: le Kit les tient identiques, un projet peut composer un classeur allégé. Que les deux index diffèrent est voulu par la conception, et non un défaut à corriger: le papier porte un classeur dans la langue de base, le site publie ce qui se lit en ligne, dans toutes les langues. L'optique — couleurs, esprit et clé de lecture — est conservée dans tous les cas. Voir *Pipeline HTML* §2.5 pour la règle complète.

Toute synchronisation silencieuse entre les deux structures par hypothèse de drift est cataloguée *anti-pattern* dans *Quality Control* §6.

9 Pipeline HTML projet

Le fichier [Préfixe] - Documentation - *Pipeline* HTML (TS).docx est propre à chaque projet — il documente les décisions de publication spécifiques au *sous-site* du projet. Il ne remplace pas *Pipeline HTML*, qui reste la source de vérité du format et des règles techniques partagées.

Contenu minimal obligatoire: sept décisions de publication doivent y figurer.

- **Inventaire des documents:** liste des documents du projet éligibles à la publication HTML — une ligne par document.
- **Flag grisé ou non-grisé:** pour chaque document, décision d'afficher la ligne en gris italique — document listé mais non lié sur la *landing page* — ou en lien cliquable vers sa page HTML.
- **Flag avec ou sans PDF:** pour chaque document publié, décision de générer une version PDF et d'afficher l'icône correspondante dans la navbar du document et sur la *landing page*.
- **Sous-domaine du projet:** URL racine du *sous-site* — utilisée par renderLandingPage.homeHref et par les liens retour de chaque page document.
- **Landing quote:** texte de citation affiché sur la *landing page* du *sous-site*. Définition verbatim en §2.4 du *Pipeline HTML* projet, même emplacement qu'en Kit *Pipeline HTML* §2.4 pour le site Kit. Le générateur de site du projet reprend ce texte tel quel dans sa constante LANDING_QUOTE.
- **Annexes:** liste des PDF de catégorie Annexe déposés à la racine du projet, chacun couplé à un .docx Kit court qui décrit son contenu. La pièce se déclare par la clé piece au *Registry*; le générateur la copie sous assets/pieces/, sous son nom d'origine, et génère le .html depuis le .docx couplé, dont le bouton sert la pièce. Voir *Pipeline HTML* §6.3.
- **Manuels:** liste des PDF de catégorie Manuel — documents originaux côté constructeur ou producteur — chacun couplé à un .docx Kit court qui décrit le matériel et l'usage. Même mécanisme de *pipeline* que les annexes.

Pipeline HTML fournit le *modèle* de format pour ces sections ainsi que les règles techniques communes — structure du dossier html, CSS externe obligatoire, assetsBase, conventions du glossaire HTML. Le *Pipeline HTML* projet complète ces règles avec les décisions propres au projet.

10 Chaîne de dépendance et cycle de vie

10.1 Chaîne principale

Les *fichiers du projet* s'articulent selon une chaîne de dépendances strictes. Les violer produit des documents techniquement valides mais visuellement faux.

- **Modules de données vers pipeline:** Brands.js alimente setBrands et donne le small caps bold en passe 1; Glossary Terms.js alimente setGlossaryTerms et donne l'italique teal en passe 2; Variable Friendly Names.js alimente setVariableNames et donne l'italique dark green en passe 3; HASS Entities.js alimente setHassEntities et donne l'italique violet en passe 4. La détection des URLs en passe 5 est automatique, sans module.
- **Stylesheet Kit vers document produit:** chaque document.docx est généré par un script **Node.js** qui fait require() du *stylesheet* Kit actif.
- **PCL vers PNG:** le rendu visuel de la *page de couverture* dépend exclusivement des constantes *PCL* du .docx *Cover Sheet*, interprétées par le *renderer Python*.
- **PCL vers landing page:** les mêmes constantes *PCL* alimentent HEADER_TEXT et TAB_COLORS sur la landing HTML.
- **Stylesheet HTML vers CSS:** getCSS() produit le fichier assets/kit-style.css référencé en externe par toutes les pages HTML du *sous-site*.

10.2 Absorber une mise à jour Kit

Quand l'*IA* signale une mise à jour Kit, la procédure d'absorption suit trois étapes, ordonnancées par dépendances. Aucun raccourci autorisé.

- Lire Kit - *Registry.js* pour obtenir les versions actives de General, Glossary, YAML et HTML, ainsi que les timestamps des documents Kit.
- Comparer chaque version active avec celle utilisée lors de la dernière *génération* d'un *couplet* projet — la StylesheetVersion injectée en *empreinte* du .docx projet est la référence.
- Pour chaque *couplet* projet impacté par un écart de version, régénérer le *couplet* avec le *timestamp* frais correspondant. La régénération suit la chaîne §10.1.

Note: *Les fichiers projet n'étaient historiquement mis à jour qu'après les fichiers Kit. Ce modèle est supprimé. La règle active est: un projet absorbe une mise à jour Kit dès qu'il est sollicité, selon la procédure ci-dessus. Le protocole détaillé de diagnostic d'écart est au Guide de mise à jour §4.4; sa déclinaison propre aux modules de données est en §6.7.*

11 Gouvernance

Les règles ci-dessous gouvernent l'évolution de tout *projet consommateur* du Kit. Aucune exception n'est accordée sans instruction explicite.

- **Préfixe Kit réservé:** L'*IA* ne crée ni ne modifie aucun fichier avec le préfixe Kit sauf si l'utilisateur travaille explicitement sur une évolution du Kit lui-même.
- **Couplets inviolables:** *stylesheet* Kit.js et Reference.docx; Kit Cover Sheet.docx et.png; projet Cover Sheet.docx et.png; projet Glossary Terms.js et Glossaire.docx. Toute régénération d'un membre entraîne la régénération de l'autre avec le même *timestamp* frais. *Inventaire* complet: *Prompt* §6.
- **Script générateur from scratch:** aucun *générateur ad hoc* gen-*.js n'est réutilisé d'une session à l'autre. Chaque session repart du *stylesheet* Kit actif et de son Reference.docx, script réécrit intégralement. Les artefacts stables — §12 et *Quality Control* §3 — n'y sont pas soumis: livrés avec le Kit pour qu'un projet dispose dès le départ de tout le nécessaire, ils n'évoluent que lorsque leur logique change.
- **Renderer PCL uniquement:** toute modification visuelle du *Cover Sheet* passe par les constantes *PCL* du .docx, jamais par le code du *renderer*. Le *renderer* est stateless et déterministe: mêmes constantes *PCL* égalent même PNG.
- **Flags Registry — source unique:** le *Prompt* projet peut référencer les flags présents dans Registry.requires mais n'en duplique jamais la valeur. Tout autre document qui mentionne un flag doit renvoyer à *Registry* comme référence.
- **Infrastructure du sous-site:** le .htaccess et le robots.txt d'un *sous-site* appartiennent au projet **sliver.lu**. Aucun générateur du Kit ni de projet ne les produit, et ils ne transitent jamais par un ZIP de déploiement. *Inventaire* des fichiers attendus et interdits associés: *Prompt* §16.
- **Livraison en une archive:** les fichiers produits au cours d'une session se livrent en une archive unique portant l'état complet du projet, jamais fichier par fichier. Le destinataire remplace un dossier au lieu de recoller des pièces, et sa sauvegarde ne peut pas diverger de la version de travail.
- **Composition d'une livraison:** les archives suivantes, remises dans le même *échange*. L'archive du projet complet, toujours — elle porte l'état entier et se substitue au dossier précédent. L'archive de publication du site, dès qu'une passe a été faite, complète ou incrémentale. Une archive de plus si le projet porte une extension Kit-X: elle se livre à part, car elle ne se range pas au même endroit — elle suit son auteur, non le projet — §3.

Note: *L'ordre de dépôt est indiqué quand il compte. L'archive de téléchargement va dans assets/downloads/ avant le ZIP du site: déposée après, l'icône de téléchargement renvoie une page absente le temps que l'archive arrive.*

- **Archive du projet:** l'archive de l'arbre entier — outils, feuilles de style, modules, documents, *Registry* —, sous le nom déclaré au *Registry* dans projectZip.filename. Elle est produite par la passe de publication quand projectShareable vaut true, posée sous assets/downloads/ du site, et c'est elle qui se livre: il n'y a plus d'archive de travail distincte.

- **Projet complet autonome:** l'archive porte tout ce qu'il faut pour que le projet tourne chez celui qui la reçoit, sans rien aller chercher dans une session.
- **Modules npm:** un package.json à la racine déclare chaque module et sa version minimale. Chaque installation se fait à neuf, par *npm* install, jamais par copie d'un node_modules.
- **Numéro de version:** il monte quand la structure du projet ou les instructions qui règlent son comportement changent: organisation des fichiers et des rubriques, conventions de nommage, *Prompt*, règles de contrôle, feuilles de style. Le second chiffre suit ces changements; le premier ne monte que si un projet qui utilise le Kit doit être migré pour le suivre. Une correction de texte, une traduction ou une republication du site ne le font pas monter. Chaque montée s'inscrit au changelog du *Registry*, avec sa raison. La date projectZip.updated change avec lui. Toute livraison qui change un fichier monte le numéro, correction de texte comprise: deux archives portant le même numéro et un contenu différent rendent les sous-projets indéchiffrables.
- **Régénération du site:** une passe de *génération* du site se livre elle aussi en une archive, que la passe soit complète ou incrémentale. La distinction gouverne ce que l'archive contient, pas la forme sous laquelle elle est remise.
- **Nomenclature des archives:** une archive livrée porte un *horodatage* frais, et son nom suit le patron de sa famille — *Convention de nommage* §4.5, qui en tabule les cas. L'archive du projet fait exception: son nom vient de Registry.projectZip.filename et porte un numéro de version, parce qu'elle est une adresse publiée — le lien du site ne change qu'avec ce numéro.
- **Structure d'index identique:** la documentation d'un projet se présente comme celle du Kit — sommaire par rubriques, chaque document en HTML et en PDF, icônes de langue sur les documents qui ont des variantes, icône de téléchargement de l'archive du projet. La seule déviation admise porte sur les couleurs du thème — barre, titres, liens —, par le module de couleurs. Les six couleurs d'onglet, reprises en cycle depuis le numéro de la rubrique, appartiennent au papier: elles y distinguent les *intercalaires* du classeur. La page d'accueil ne les porte pas — le numéro d'une rubrique s'y écrit dans COVER_NUM_COLOR, une couleur unique pour tout le site, réglable par projet comme celle de la barre.
- **Journal de projet:** chaque projet tient un journal des entrées ouvertes, des dettes et des décisions datées. Structure et contenu au §14. C'est le seul document du projet autorisé à porter un historique.

12 Artefacts stables exécutables projet

Un *projet consommateur* peut produire ses propres artefacts stables exécutables: scripts réutilisables session après session qui vivent dans l'*arbre du projet*, distincts des *générateurs ad hoc* recréés à chaque session. Exemple type: trading_generate_history_report.py.

12.1 Convention de nommage

Nomenclature Unix snake_case. Préfixe du projet en minuscules suivi d'un tiret bas, nom en snake_case, pas d'*horodatage* dans le nom. Exception formelle à *Convention de nommage* §2, documentée dans ce même document §3.3. Exemples projet: trading_generate_history_report.py, hexi_validate_recipe.py. Critère d'application: *artefact stable* exécutable — *Python*, CLI **Node.js**, shell. Tout autre fichier suit le patron principal.

12.2 Rôle — validateur ou renderer

Deux catégories possibles. Un *validateur* retourne exit code 0 si tout est conforme, non-zéro sinon, et bloque la livraison en cas d'échec. Un *renderer* produit un *artefact* déterministe: même entrée et même *contexte* donnent la même sortie. Si le besoin projet n'entre dans aucune de ces deux catégories, c'est probablement un *générateur ad hoc* plutôt qu'un *artefact stable* — utiliser le préfixe gen- et la *Convention de nommage* §3.6. Correspond au découpage Kit documenté dans *Quality Control* §3.

12.3 Cycle de vie

L'*artefact stable* projet vit en permanence dans l'*arbre du projet*, pas dans celui du Kit. Il n'est pas régénéré à chaque session — seulement quand sa logique évolue. Son évolution est tracée dans un bloc changelog en tête du fichier lui-même, au même titre que les stylesheets Kit. Pas de document externe de suivi imposé — la convention Unix standard s'applique. L'*artefact* ne porte pas d'*horodatage* dans son nom.

12.4 Recette de création

Quand un projet identifie un besoin récurrent qui mériterait un *artefact stable*, la session qui le crée suit sept étapes.

- Confirmer avec l'utilisateur que le besoin est bien récurrent et non ponctuel.
- Nommer le fichier selon §12.1.
- Écrire le script autonome en minimisant les dépendances externes.
- Documenter les dépendances en en-tête du fichier.
- Documenter le contrat d'usage en docstring: entrées, sorties, codes de retour, exemples d'appel.
- Initialiser le bloc changelog en tête du fichier.
- Livrer via present_files pour que l'utilisateur le range dans l'*arbre du projet*.

12.5 Évolution ultérieure

Quand un *artefact stable* doit évoluer, bug ou extension, la session lui est dédiée et n'est pas mélangée à d'autres livrables. La source est lue directement depuis l'*arbre du projet* — ils s'y lisent directement. La

modification incrémente la version dans le bloc changelog en tête du fichier avec description de la modification. *Livraison séquentielle* standard: validation, copie vers outputs, present_files.

Note: *Un artefact stable appartient à la livraison du projet complet, au même titre que les documents et les feuilles de style. Une archive qui l'omettrait ne permettrait pas de repartir de zéro, et le manque ne se verrait qu'au moment de restaurer.*

13 Bug report Kit

Un *projet consommateur* peut rencontrer un comportement Kit qui mérite remontée: rendu inattendu d'une fonction *stylesheet*, divergence entre le code.js et son Reference.docx, *anti-pattern* non encore catalogué, faux positif d'un *validateur*. Le canal de remontée structuré est un document court dédié, produit par le projet et livré au *chat* du projet Kit pour analyse et correction. Il remplace les briefs informels qui ont historiquement servi — même contenu, format unifié.

13.1 Objet et déclencheurs

Le *bug report Kit* est produit quand le *projet consommateur* détecte un comportement Kit qui ne peut pas être résolu localement dans le projet. La règle *Prompt* §5.3 s'applique: un contournement local ne remplace jamais une correction à la source. Si la cause racine est dans le Kit, c'est le Kit qui doit corriger.

Trois déclencheurs typiques.

- **Bug de stylesheet:** une fonction *stylesheet* ne rend pas comme documenté dans son Reference.docx — divergence entre le comportement runtime et le contrat. Exemple: le paramètre bgColor de sectionBanner, mal nommé, corrigé en accentColor sur remontée du projet Sorso.
- **Bug de pipeline:** une étape du *pipeline* produit un résultat incohérent ou silencieusement amputé. Exemple: gen-kit-site.js n'appelait pas les *setters* data du *stylesheet* HTML — site Kit historiquement sans markup de marque ni lien glossaire.
- **Anti-pattern non catalogué:** une erreur récurrente détectée dans le projet qui mériterait une entrée dans *Quality Control* §6 et idéalement un check automatique en *Couche 2*.

13.2 Nomenclature

Le *bug report* suit le patron standard de la *Convention de nommage* §2.1: [Préfixe] - Project - Kit *bug report* (TS).docx. Catégorie Project conformément à §7, ouverte aux deux préfixes. Le sujet "Kit *bug report*" reste en anglais dans toutes les langues — nom technique stable au sens §5.2 de la Convention.

La catégorie Project se traduit selon project.docLanguage du projet émetteur.

Le fichier est généré par le *projet consommateur* via un script gen-kit-bug-report.js from scratch, comme tout document projet — moteur de formatage General *stylesheet* Kit actif au moment de la rédaction, *validateur* kit_validate_docx en sortie.

13.3 Structure du document

Sections obligatoires, dans cet ordre. Court par construction — le *bug report* n'est pas une thèse: il décrit, reproduit, et propose. La validation détaillée et la correction se font ensuite en session Kit.

Le rapport de bug est un document de constat — *Kit Prompt* §12 — dont l'ossature est fixée ici: il rend compte d'un défaut existant, ne gomme rien, et sa dernière section porte ce qui reste à trancher.

- **§1 Identification:** préfixe projet, date de constat, project.docLanguage, versions *stylesheet* Kit actives au moment du bug — lues dans Kit - *Registry.js*, le *Registry* projet n'en portant pas, et copiées telles quelles — et documents ou pipelines impliqués.
- **§2 Observation:** comportement observé et comportement attendu, en deux paragraphes courts. Décrit factuellement ce qui s'est passé, sans interprétation.
- **§3 Reproduction:** extrait minimal du gen-*.js ou commande qui déclenche le bug, en rawBlock, plus la séquence exacte exécutée. Suffisamment précis pour que l'*IA* côté Kit puisse reproduire en session sans questions complémentaires.
- **§4 Impact et diagnostic suggéré:** quels documents ou pipelines sont affectés et avec quelle sévérité. Hypothèse projet sur la cause racine, non verrouillante: l'*IA* côté Kit refera l'analyse complète. Préciser si un contournement local existe ou non.

13.4 Workflow de remontée

Une fois le bug report.docx généré et validé côté projet, la remontée se fait par dépôt simultané des fichiers sources dans le fil du projet Kit. Le projet Kit n'a accès qu'à ce qui est joint dans le *chat* — pas à l'arbre des *projets consommateurs*.

- **Le bug report .docx:** fichier principal, fichier source dans le fil. Un extrait de texte est insuffisant pour le contenu canonique d'un .docx.
- **Le gen-*.js incriminé:** script générateur projet qui déclenche le bug, ou extrait minimal — texte direct dans le *chat*, ou fichier .js joint.
- **Le .docx résultat:** si le bug produit un .docx mal rendu, joindre le fichier source pour inspection — compte de tables, w:tbl, *custom properties*.
- **Captures éventuelles:** screenshot Word, *LibreOffice*, ou navigateur si le bug est visuel — joints comme images.

Note: *Le bug report est lu en début de session Kit comme un brief. L'IA côté Kit le rapproche des règles actives, identifie la couche concernée, propose un plan de correction, attend le go explicite avant toute génération. Cycle standard de livraison séquentielle ensuite.*

13.5 Non-permanence

Le *bug report* est un fichier éphémère: produit, livré, archivé après résolution. Il n'apparaît pas dans l'*inventaire* §2 — celui-ci liste les fichiers projet permanents qui survivent entre sessions, ce qui n'est pas le cas du *bug report*. La trace de la résolution vit ailleurs: entrée dans Todos côté Kit, éventuelle mise à jour d'un document Kit.

Note: *Conservation côté projet: libre choix de l'utilisateur. Aucun stockage obligatoire dans l'arbre du projet — le bug report a fait son travail dès que la correction Kit est livrée et que le projet a absorbé la nouvelle version du couplet concerné selon la procédure §10.2.*

14 Journal de projet

Chaque projet tient un journal. Il porte ce qui reste à faire et la trace datée de ce qui a été décidé — c'est ce qui permet, des mois plus tard, de retrouver pourquoi une chose a été faite ainsi plutôt qu'autrement.

Il est calqué sur *Kit - Projet - Todos*, qui en est l'exemplaire de référence. Ses sections, dans cet ordre.

- **§1 Bugs ouverts.** Ce qui est cassé et non réparé. Une entrée par défaut, avec ce qu'il produit et ce qu'il empêche. La section porte le nombre d'entrées ouvertes et sa date, de sorte qu'un "aucune entrée" se lise comme un constat daté.
- **§2 Chantiers.** Ce qui demande une décision avant d'être fait, ou un travail dont la portée dépasse une session. Une entrée nomme ce qui bloque, non ce qu'on souhaite.
- **§3 Dettes techniques.** Ce qui fonctionne et qu'on sait imparfait. Une dette qui ne gêne personne aujourd'hui reste une dette: elle est inscrite avec la raison de ne pas la traiter maintenant.
- **§4 Registre des résolutions.** Un tableau à quatre colonnes — numéro, intitulé, date, contenu — en numérotation continue qui ne se réutilise jamais. Chaque entrée dit ce qui a été décidé et pourquoi, avec les versions livrées.

Note: *Ce qui s'inscrit: les décisions de structure. Un changement de forme, une règle posée ou retirée, un défaut corrigé, un arbitrage tranché. Ce qui ne s'inscrit pas: les passes de génération, les publications, les mises à jour de données — elles laissent leur trace dans les horodatages, pas dans le journal.*

Note: *Le journal est le seul document du projet autorisé à porter un historique. Les autres décrivent un état; s'ils portaient aussi leur propre passé, un lecteur ne saurait plus lequel des deux fait foi.*

15 Sélection de langue

Un site entièrement multilingue et un site dont quelques documents seulement sont traduits n'ont pas le même besoin. Le Kit offre les deux mécanismes; le projet déclare lequel il emploie, sous rendering.languageSelector.

Mode	Ce qu'il suppose	Où le sélecteur apparaît
site	Chaque page publiée existe dans chaque langue déclarée	Menu à la fin de la barre, identique sur toutes les pages
document	Rien; certains documents seulement sont traduits	Icônes au niveau du document, là où des variantes existent

Note: *Le défaut vaut document: il ne suppose rien du projet. Le manquer ne casse rien et se voit à la première page ouverte — le générateur annonce en outre le mode à chaque passe. Un arrêt bruyant se réserve aux défauts qui ne se voient pas.*

Note: *En mode site, un document non traduit ferait apparaître et disparaître le menu d'une page à l'autre, et il cesserait d'être un repère. Un projet qui publie des documents monolingues emploie le mode document.*

- **Ce que le projet fournit.** La liste des variantes de chaque page — code de langue et adresse. Le Kit fournit les deux mécanismes, leurs libellés localisés et leurs drapeaux.

16 Langue de base et traductions

Le projet déclare sa langue de base sous project.docLanguage. Les documents rédigés dans cette langue font référence: une règle, une correction ou une décision s'y écrit d'abord, et c'est depuis eux que l'on travaille, humain comme *IA*.

Les traductions sont un service au lecteur. Elles rendent le sens pour se lire comme un texte écrit dans leur langue, jamais mot à mot. Rien n'y est perdu ni ajouté; noms de fichiers, code et numéros de section restent tels quels.

Chaque variante porte en tête de son §1 la note fixe du *Kit Prompt* §17, avec la langue de base du projet. Une traduction se met à jour sur demande, depuis la carte du document de référence, et son ossature se contrôle par kit_check_ossature.py.

Doctrine complète: *Kit Prompt* §17.