Kit de documentation

Projekt — Gemeinsame Struktur — DE

1 Einleitung

Dieses Dokument ist eine sinngemäße Übersetzung. Maßgeblich ist das Originaldokument auf Französisch; Erweiterungen und Änderungen werden stets dort eingepflegt.

Dieses Dokument legt die verbindliche Struktur jedes Projekts fest, das das Kit de documentation einsetzt. Es ist maßgeblich für die Dateien, die ein Projekt besitzen muss, für die normative Form jeder Datendatei und für die Lebenszyklusregeln, die deren Entwicklung bestimmen.

Zwei Fälle fallen unter denselben Vertrag: die Einrichtung eines neuen Projekts und die Anpassung eines bestehenden Projekts nach einer Kit-Aktualisierung. In beiden Fällen werden die *Projektdateien* aus den unten stehenden Gerüsten erzeugt oder neu erzeugt.

Adressat ist der aktive Kit-Nutzer, der ein oder mehrere Projekte führt. Die *KI* liest dieses Dokument zu Beginn einer Projektsitzung, um die Konformität vor jeder *Erzeugung* zu prüfen.

2 Verzeichnis der Projektdateien

Normative Tabelle der Dateien, die ein aktives Projekt besitzen muss. Die Spalte Flag nennt den Namen des Schalters in [Präfix] - Registry.requires; die Spalte Standard nennt den Wert bei der Einrichtung eines Projekts. Eine Datei ohne Flag ist ausnahmslos verbindlich.

Projektdatei	Rolle	Flag	Standard
[Präfix] - Projet - *Prompt* (TS).docx	Projektspezifische technische Anweisungen	—	Verbindlich
[Präfix] - *Registry.js*	Wahrheitsquelle des Projekts — Versionen, Flags, *Zeitstempel*	—	Verbindlich
[Präfix] - Settings.js	Projekteigene Einstellungen — außerhalb des Kit-Umfangs	—	Bei Bedarf
[Präfix] - Projet - Todos (TS).docx	Offene Einträge, Schulden und datiertes Verzeichnis der Entscheidungen	—	Verbindlich
[Präfix] - Glossary - Terms (TS).js	Glossarmodul — Quelle des Glossars und der *Pipeline*-Passe 2	glossary	true

Projektdatei	Rolle	Flag	Standard
[Präfix] - Glossaire - Termes - LANG (TS).docx	Projektglossar — Couplet mit Terms.js. Ein Dokument je veröffentlichter Sprache, wenn das Glossar übersetzt ist. Benennung: Namenskonvention §2.1 und §2.4	glossary	true
[Präfix] - Brands (TS).js	Markenmodul — *Pipeline*-Passe 1	brands	false
[Präfix] - Text Highlights (TS).js	Hervorgehobene Textstellen — *Pipeline*, zwischen Marken und Glossar	textHighlights	false
[Präfix] - Variable Friendly Names (TS).js	Variablenmodul — *Pipeline*-Passe 3	variableNames	false
[Präfix] - HASS Entities (TS).js	Modul der HASS-Entitäten — *Pipeline*-Passe 4	hassEntities	false
[Präfix] - Colors (TS).js	Farben des Projekts — ersetzen die des Kits, §6.5	colors	false
[Präfix] - Document Titles (TS).js	Anzeigenamen der Dokumente je Sprache, §6.8	documentTitles	false
[Präfix] - Documentation - *Cover Sheet* (TS).docx	*PCL*-Konstanten — Couplet mit dem PNG	coverSheet	true
[Präfix] - Documentation - *Cover Sheet* (TS).png	*Deckblatt* des Papierordners	coverSheet	true
[Präfix] - Documentation - Reading Guide (TS).docx	Leseleitfaden des Papierordners	readingGuide	true

Projektdatei	Rolle	Flag	Standard
[Präfix] - Documentation - *Pipeline* HTML (TS).docx	Spezifikation der HTML-Veröffentlichung des Projekts	—	Verbindlich

Hinweis: *Eine Datei, deren Flag false ist, fehlt im Projekt. Die **Node.js**-Generatoren lesen Registry.requires beim Start und überspringen die zugehörigen require() und Setter — siehe §6.6. Alle Dateien tragen einen Zeitstempel im Namen, mit der einzigen Ausnahme von [Präfix] - Registry.js, das selbst die Quelle der Zeitstempel ist. Ein in mehreren Sprachen veröffentlichtes Dokument trägt zusätzlich ein Sprachkürzel vor seinem Zeitstempel — Namenskonvention §2.4.*

3 Gemeinsame Kit-Abhängigkeiten

Die Stylesheets des Kits — General, Glossary, YAML, HTML — sind technische Bibliotheken, die alle Projekte teilen. Sie werden nie in ein Projekt kopiert. Die Generatoren eines Projekts verweisen per require() unmittelbar auf ihren Ort im Kit.

Die Datei Kit - *Registry.js* ist die einzige Quelle der aktiven *Stylesheet*-Versionen. Projekte lesen sie, um eine Kit-Aktualisierung zu erkennen und zu wissen, ob ein Projekt-Couplet neu erzeugt werden muss. Kein Projekt speichert eine *Stylesheet*-Version lokal — die Wahrheit steht im Moment der *Erzeugung* stets in Kit - *Registry.js*.

Das Kit besitzt außerdem sein eigenes Glossar Kit - Glossaire.docx, getrennt vom Projektglossar. Ein Projekt trägt seine eigenen Begriffe in seinem Begriffsmodul: Die Generatoren laden das des Projekts, nie das des Kits.

Eine dritte Gruppe kann im Arbeitsverzeichnis auftauchen: die Dateien mit dem Präfix Kit-X. Das sind die vom Kit-Nutzer geschriebenen Erweiterungen — zusätzliche Stylesheets, eine Farbpalette, das *Register* dieser Stylesheets. Sie sind weder Kit-Dateien, die bei jeder Aktualisierung ersetzt werden, noch *Projektdateien*, die einem Projekt eigen sind: sie gehören ihrem Autor und folgen ihm von Projekt zu Projekt.

- **Ein Projekt liest sie, ohne sie zu besitzen:** ein Generator kann ein Kit-X-*Stylesheet* einbinden wie eines des Kits. Das *Registry* des Projekts erklärt deren Versionen nicht — diese stehen in Kit-X - *Registry.js*. Benennung: Namenskonvention §3.3. Anleitung: Das Kit erweitern.

4 Projekt-Prompt

Die Datei [Präfix] - Projet - *Prompt* (TS).docx enthält die projektspezifischen technischen Anweisungen. Sie ergänzt den *Prompt* — sie verdoppelt ihn nie. Die *KI* liest sie zu Beginn jeder Arbeitssitzung am Projekt.

Anzupassende Werte: das Projektpräfix, project.documentAuthor und project.webAuthor (die im Seitenfuß des Dokuments und der Seite angezeigten Namen), project.docLanguage (FR, EN, DE oder LU — Standard EN für Projekte, einzige Wahrheitsquelle im Abschnitt project von [Präfix] - *Registry.js*), die redaktionellen Konventionen des Projektgebiets sowie etwaige eigene Erzählregeln.

Die technischen Regeln, die alle Projekte teilen — Formatierung, Sitzungsprotokolle, Namenskonvention, Prüfkette — stehen im *Prompt* und werden im Projekt-*Prompt* nicht wiederholt. Der Projekt-*Prompt* enthält nur das, was sich von Projekt zu Projekt unterscheidet.

5 Projekt-Registry

Jedes Projekt hat sein eigenes [Präfix] - *Registry.js*. Es ist die Wahrheitsquelle für Identität, Schalter der erforderlichen Dateien, Familiendomänen, *Zeitstempel* der erzeugten Dokumente, Rendering-Schalter, den Kompilierungshinweis im Web und die Metadaten der Website-Veröffentlichung.

5.1 Rolle und Inhalt

Das Projekt-*Registry* führt die folgenden verbindlichen Abschnitte und einen optionalen. Jeder hat seine eigene Aktualisierungsregel — siehe §5.3.

- **project:** Identität und Sprache des Projekts. Das Feld project.docLanguage (FR, EN, DE, LU) ist die einzige Wahrheitsquelle der Sprache; die Stylesheets lesen es für die *L10N*-Auswahl, der Cover-Sheet-*Renderer* für die lokalisierten *PCL*-Zeichenketten. projectShareable, standardmäßig false, sagt, ob das Projekt zur Übernahme angeboten wird: bei true erzeugt der Veröffentlichungsdurchlauf das Projektarchiv, und das Download-Symbol erscheint. allowNonKitTemplates, standardmäßig false, eröffnet das Recht, Modelle außerhalb der Formen des Kits zu schaffen — Das Kit erweitern §9. Drei boolesche Kennzeichen kommen ebenfalls hinzu, standardmäßig false: autoAddGlossary, autoAddTextHighlights und autoAddBrands. Bei false meldet die *KI* einen Kandidaten und wartet auf Bestätigung; bei true fügt sie ihn dem betroffenen Modul hinzu und meldet es am Ende des Abschnitts — *Kit Prompt* §13.2.
- **requires:** Schalter für das Vorhandensein optionaler Dateien — coverSheet, readingGuide, glossary, textHighlights, documentTitles, brands, variableNames, hassEntities, colors. Von den Generatoren gelesen, um require() und *Setter* zu bedingen.
- **familyDomains:** Liste von Hostnamen ohne Schema. Familiendomänen, deren Links — und die ihrer Subdomänen — in der *WebView* der *Android*-App gefangen bleiben. Jede andere Domäne wird an den Systembrowser weitergereicht. Gelesen von style.setFamilyDomains() am Kopf von gen-[projekt]-site.js. Stabiler Abschnitt — nur bei einer Änderung des Familienumfangs angepasst.
- **documents:** Zuordnung von Slug zu *Zeitstempel* für jedes erzeugte Dokument des Projekts. Bei jeder Dokumentlieferung aktualisiert — der *Zeitstempel* gibt die letzte Neuerzeugung wieder. Ein in mehreren Sprachen veröffentlichtes Dokument hat einen Eintrag je Variante, dessen Slug das Sprachkürzel trägt: philosophy-fr, philosophy-de. Die

inkrementelle Veröffentlichung vergleicht nach Slug, sodass jede Sprache unabhängig neu veröffentlicht wird. Der optionale Schlüssel piece nennt die Datei eines Anlagenstücks: Sie liegt im Wurzelverzeichnis des Projekts, der Durchlauf kopiert sie nach assets/pieces/ der Website, unter ihrem ursprünglichen Namen, und die Download-Schaltfläche der Seite liefert sie anstelle des PDF des Dokuments — HTML-*Pipeline* §6.3. Zwei optionale Schlüssel kommen hinzu, vom Website-Generator geschrieben: contentTs, der *Zeitstempel* der letzten Textänderung, und contentHash, die *Signatur* des Textes, der ihn erzeugt hat. Ein *Registry* ohne sie funktioniert; der Generator setzt sie beim ersten Durchlauf.

- **rendering:** Schalter, die die *Erzeugung* von Ergebnissen steuern. Die Unterabschnitte coverSheet.sectionHMode (FIXED oder DYNAMIC), von kit_render_cover_sheet.py gelesen, und toc.generate / toc.hidden, die das seitliche HTML-Inhaltsverzeichnis über renderDocument und getCSS steuern. Der Abschnitt ist freigestellt — fehlt er, greift der *Renderer* auf FIXED zurück. Ihn ausdrücklich zu erklären bleibt empfohlen, damit die Wahl lesbar ist. languageSelector wird hier ebenfalls eingestellt — §15 —, und indexIcon erklärt die Illustration der *Startseite*, das vierte Symbol des Projekts neben favicon.svg, favicon.ico und apple-touch-icon.png, alle vier in seinem Wurzelverzeichnis. homeHref gibt das Ziel des Haus-Symbols allein auf der *Startseite* an, das aus der *Subsite* hinausführt; das Haus einer Dokumentseite führt stets zum Verzeichnis ihrer Sprache, ohne Einstellung —, dazu indexIcon: Ein Projekt, das index-icon.svg in sein Wurzelverzeichnis gelegt hat, erklärt es hier, und die *Startseite* zeigt das Bild links vom Untertitel. Fehlt es, ändert sich nichts.
- **compiledWith:** Kompilierungshinweis im Seitenfuß der *Subsite*-Seiten. Ein nach Sprachcode indiziertes Objekt, vom Site-Generator über config.compiledWith an das HTML-*Stylesheet* übergeben. Fehlender Block, null oder leere Zeichenkette für eine Sprache: es wird kein Hinweis erzeugt. Dieses Feld steuert nur das Web — kein vom Kit erzeugtes.docx trägt einen Kompilierungshinweis, da das General-*Stylesheet* den zugehörigen *L10N*-Schlüssel entfernt hat.
- **deploy:** Metadaten der HTML-*Subsite* — typischerweise lastDeploy. Von gen-[projekt]-site.js bei einer vollständigen oder inkrementellen Veröffentlichung geschrieben.
- **stylesheets:** OPTIONAL. Versionen und *Zeitstempel* der vom Projekt geschriebenen Stylesheets, nach dem Muster des gleichnamigen Abschnitts in Kit - *Registry.js*. Fehlt, wenn das Projekt keines schreibt. Er führt nur die Stylesheets des Projekts: die des Kits bleiben im

Registry des Kits erklärt, der einzigen Quelle ihrer Versionen — §3. Anleitung: Das Kit erweitern.

5.2 Normative Struktur

Gerüst, das bei der Einrichtung eines Projekts zu füllen ist. Die Kommentare erinnern an die Bedeutung jedes Abschnitts. Die Datei trägt keinen *Zeitstempel* im Namen — sie ist selbst die Quelle der *Zeitstempel*.

```
'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
  },

};
```

Hinweis: *Das Gerüst wurde um die Abschnitte project, familyDomains, rendering und compiledWith erweitert — sie fehlten historisch, obwohl die Stylesheets und der Cover-Sheet-Renderer sie lasen. Ein nach dem alten Gerüst gebautes Projekt konnte kit_render_cover_sheet.py mit einem KeyError abstürzen lassen. Der Schutz ist nun doppelt: vollständiges Gerüst hier, und ein Rückfall auf der Renderer-Seite.*

5.3 Lebenszyklus

Der Abschnitt project ist stabil — nur geändert, wenn das Projekt seine Hauptsprache wechselt, ein seltener und ausdrücklicher Fall, beschrieben im Aktualisierungsleitfaden §6.

Der Abschnitt requires ist über die Lebensdauer des Projekts stabil — nur geändert, wenn eine funktionale Abhängigkeit hinzukommt oder wegfällt. Ein Projekt, das beginnt, *Home Assistant* zu dokumentieren, setzt hassEntities von false auf true und legt das zugehörige Modul an.

Der Abschnitt familyDomains ist stabil — nur bei einer Änderung des Familienumfangs angepasst.

Der Abschnitt documents wird bei jeder Dokumentlieferung aktualisiert — der Generator des Dokuments, oder der Nutzer nach Erhalt, trägt den neuen *Zeitstempel* ein.

Der Abschnitt rendering ist stabil — geändert, um beim *Cover Sheet* zwischen FIXED und DYNAMIC zu wechseln, ein seltener Fall, oder um das HTML-Inhaltsverzeichnis umzuschalten. Der Nutzer ändert *Registry.js* nie von Hand: er beauftragt die *KI* im jeweiligen Projekt, die den Wechsel vornimmt.

Der Abschnitt compiledWith ist stabil — er ist eine redaktionelle Entscheidung des Projekts, kein technischer Zustand. Ihn zu ändern erzwingt keine Neuerzeugung eines Dokuments: der Wert wird beim Erzeugen der Website gelesen.

Der Abschnitt deploy.lastDeploy wird ausschließlich von gen-[projekt]-site.js bei einer Veröffentlichung geschrieben. Kein anderes Skript und auch nicht der Nutzer bearbeiten ihn von Hand.

5.4 Projekteigene Einstellungen — Settings.js

Das *Registry* des Projekts trägt nur die Blöcke, die das Kit festlegt: project, requires, stylesheets, familyDomains, documents, rendering, compiledWith, projectZip, deploy. Ein Projekt trägt dort Werte ein, es fügt weder Schlüssel noch Block hinzu. Was ihm gehört, lebt in einer eigenen Datei, [Präfix] - Settings.js, die das Kit weder festlegt noch liest.

- **Eine Dateigrenze, keine Disziplinfrage.** Ein eigener Block im *Registry* zwingt dazu, von Fall zu Fall zu beurteilen, ob sein Ort berechtigt ist. In einer getrennten Datei ist ein dem *Registry* unbekannter Schlüssel schon von der Bauart her falsch, und eine Prüfung kann das sagen, ohne zu entscheiden.

- **Im Projekt-Prompt dokumentiert.** Jede Rubrik von Settings.js trägt dort ihren Namen, ihre Schlüssel, die Wirkung jedes einzelnen und die Wirkung seines Fehlens. Ein nicht dokumentierter Schlüssel ist ein zur nächsten Sitzung verlorener Schlüssel.
- **Mit lautem Abbruch gelesen.** Der Generator, der eine Einstellung liest, bricht unter Nennung des Schlüssels ab, wenn dieser fehlt oder nicht den erwarteten Typ hat, statt auf einen stillen Standardwert zurückzufallen.

Hinweis: *kit_check_registry.py weist jeden Schlüssel erster Ebene zurück, der nicht auf der Liste des Kits steht. Es läuft vor jeder Erzeugung — Quality Control §3. Die Trennung gilt nur, wenn etwas sie prüft: ohne Prüfung wäre sie eine zweite Konvention, so freigestellt wie die erste.*

Hinweis: *Die Kit-Aktualisierung rührt weder das Registry des Projekts noch dessen Settings.js an — Aktualisierungsleitfaden. Kein Mechanismus schützt sie: das Verfahren ersetzt sie schlicht nicht, und darum zählt es auf, was es ersetzt.*

Hinweis: *Kit-X-Erweiterungen sind etwas anderes: sie gehören ihrem Autor, folgen ihm von Projekt zu Projekt und erklären ihre Versionen in Kit-X - Registry.js — §3.*

6 Datenmodule

Ein Projekt kann *Datenmodule* besitzen — jedes bedingt durch einen Schalter in Registry.requires aktiviert. Die Strukturregeln jedes Moduls sind normativ und nicht auslegbar. Jede Abweichung zwischen einer vorhandenen Datei und der unten stehenden Struktur ist eine Anomalie, die vor jeder *Erzeugung* zu beheben ist.

Dieser Abschnitt ist der einzige Wohnsitz des Vertrags der *Datenmodule*. Die anderen Dokumente des Kits verweisen darauf, ohne ihn zu wiederholen.

6.1 Glossary Terms

Datei: [Präfix] - Glossary - Terms (TS).js. Flag Registry.requires.glossary, Standard true. Couplet mit dem oder den Glossardokumenten — alle Mitglieder teilen bei jeder Neuerzeugung denselben *Zeitstempel*. Benennung der Dokumente: Namenskonvention §2.1 und §2.4.

Das Modul wird am Kopf jedes Generators geladen und speist Passe 2 der *Verarbeitungskette* — die Begriffe werden im gesamten Fließtext selbsttätig kursiv in Teal gesetzt.

Pflichtfelder je Eintrag: jedes Element von glossaryEntries trägt vier Felder, von denen drei erforderlich sind.

Feld	Typ	Pflicht	Beschreibung
balise	string	Ja	Stabiler HTML-*Anker* — _slugify des Begriffs in seiner Ursprungssprache, in der Datei eindeutig. Sprachunabhängig und nach der Veröffentlichung nie geändert: er bezeichnet den Begriff, nicht das Wort
term	string oder Objekt	Ja	Kanonischer Begriff. Zeichenkette, wenn das Glossar einsprachig ist, sonst ein nach Sprachcode indiziertes Objekt
aliases	Array oder Objekt	Nein	Alternative Formen, die auf dieselbe Marke zeigen. Array, wenn einsprachig, sonst ein nach Sprachcode indiziertes Objekt
definition	string oder Objekt	Ja	Definition in gewöhnlicher Sprache, nie *Markdown*. Zeichenkette, wenn einsprachig, sonst ein nach Sprachcode indiziertes Objekt

Ein Projekt, das sein Glossar in mehreren Sprachen veröffentlicht, indiziert term, aliases und definition nach Sprachcode. Die Marke selbst ändert sich nie: sie wird aus dem Begriff in seiner Ursprungssprache abgeleitet und dient als öffentlicher *Anker*. Das erlaubt der Sprachauswahl, von einer Seite zur anderen zu wechseln und die Position des Lesers zu bewahren — die

deutsche Fassung eines Dokuments zeigt auf denselben *Anker* wie die französische.

Beide Formen bestehen nebeneinander, ohne Migration. Eine Zeichenkette bedeutet einsprachig, ein Objekt bedeutet mehrsprachig. Ein Projekt wechselt Begriff für Begriff, in dem Moment, in dem es übersetzt.

- **Sortierung je Sprache:** die alphabetische Reihenfolge der Einträge richtet sich nach dem Begriff in der gerenderten Sprache. Die Glossare zweier Sprachen haben daher nicht dieselbe Reihenfolge, was für ein Glossar richtig ist.
- **Fehlendes Feld — lauter Abbruch:** hat eine veröffentlichte Sprache keinen Wert für term oder definition, hält der Generator an und nennt die schuldige Marke. Kein stiller Rückfall auf die Sprache des Registrys: ein ohne Warnung verstümmeltes Glossar ist genau der Schaden, den das Kit bekämpft.
- **Übersetzte Aliasse:** die Aliasse einer Sprache speisen die Erkennung der *Verarbeitungskette* in den Dokumenten dieser Sprache. Ohne deutsche Aliasse färbt sich in einem deutschen Dokument kein Begriff teal — die Kette verstummt dort, wo man sie erwartet.
- **Nur im Glossar:** das optionale Feld detection: false hält den Eintrag im Glossar, auf Papier wie auf der Website, ohne dass er die Erkennungsdurchläufe speist — für ein gängiges Wort, das man definieren, aber nicht überall einfärben will. buildSearchTerms lässt ihn aus.
- **Zusammensetzungen mit Bindestrich:** eine Zusammensetzung, deren zweiter Teil mit einem Kleinbuchstaben beginnt — zone-based, Tasmota-flashed —, wird nur erkannt, wenn sie als Alias des Begriffs erklärt ist. Sonst behandelt die Grenze sie wie einen Datei- oder Paketnamen, den sie schützt: python-docx. Ein zweiter Teil mit Großbuchstaben bleibt eine Wortgrenze — HTML-*Stylesheet*.

Pflichtexporte: glossaryEntries, ein Array von Objekten, und buildSearchTerms(LANG), eine Funktion, die die Paare surface und anchor für die angeforderte Sprache longest-first sortiert zurückgibt. Ein einsprachiges Glossar darf den historischen Export glossarySearchTerms behalten.

Gerüst, das bei der Einrichtung oder beim Übernehmen einer Kit-Regel zu füllen ist, die den Vertrag erweitert.

```
'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.** Die Rubriken des Glossars: Nummer, Titel und Anreißer je Sprache. Das Feld rubrique jedes Begriffs verweist darauf. Ein Rubriktitel ist Glossarinhalt, keine Rendering-Entscheidung — ihn in einen Generator zu schreiben brächte ihn außer Reichweite des Autors und ließe ihn zwischen Papier und Web auseinanderlaufen.
- **glossaryDocument.** Die Identität des Dokuments und seine Rahmentexte: Projekttitel, Autor, maßgebliche Sprache, Segmente des Dateinamens je Projektsprache, und je Rendersprache der Anreißer, die Eröffnungsnotiz, die Einleitung, die Schlussnotiz und die Übersetzungsnotiz. Der *Artefakt*, der das Glossar erzeugt, kennt also weder den Projektnamen noch dessen Sprache: er liest sie hier.

Hinweis: *Das Feld nomBase folgt Namenskonvention §2.4: Kategorie und Sujet bleiben in der Projektsprache, nur das Schlusskürzel markiert die Sprache des Inhalts. Ein englischsprachiges Projekt erzeugt “AI - Glossary - Terms - EN”, nicht “AI - Glossaire - Termes - EN”.*

6.2 Brands

Datei: [Präfix] - Brands (TS).js. Flag Registry.requires.brands, Standard false. Datei fehlt, wenn das Projekt keine Marken dokumentiert.

Pflichtexport: brandEntries, ein flaches Array von Markennamen. Über Passe 1 der Kette im gesamten Fließtext in Kapitälchen fett gesetzt. Empfohlene Reihenfolge: longest-first, um Überschneidungen zwischen verschachtelten Marken zu vermeiden.

```
'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

Datei: [Präfix] - Variable Friendly Names (TS).js. Flag Registry.requires.variableNames, Standard false. Datei fehlt, wenn das Projekt keine technischen Variablen dokumentiert.

Pflichtexport: variableNameEntries. Namen von Variablen und technischen Parametern. Über Passe 3 der Kette kursiv in Dunkelgrün VARIABLE_NAME_COLOR gesetzt.

```
'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

Datei: [Präfix] - HASS Entities (TS).js. Flag Registry.requires.hassEntities, Standard false. Datei fehlt, wenn das Projekt *Home Assistant* nicht dokumentiert.

Pflichtexport: hassEntityEntries. Namen von Home-Assistant-Entitäten. Über Passe 4 der Kette kursiv in Violett HASS_ENTITY_COLOR gesetzt.

```
'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 Farben

Datei: [Präfix] - Colors (TS).js, für die Farben dieses Projekts. Eine zweite Ebene besteht außerhalb des Projekts — Kit-X - Colors (TS).js — für jene, die ihrem Autor von Projekt zu Projekt folgen; sie wird von keinem Schalter gesteuert, sie ist da oder nicht. Flag Registry.requires.colors, Standard false. Datei fehlt, wenn das Projekt die Farben des Kits behält.

Pflichtexport: colorEntries, ein Objekt mit zwei Tabellen — docx für die Dokumente, html für die Webseiten. Jede Tabelle trägt nur die ersetzten Schlüssel; ein fehlender Schlüssel behält den Standard des Stylesheets. Die Schlüsselnamen und ihre Standards sind in den Reference-Dokumenten beider Stylesheets tabelliert.

```
'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 };
```

Hinweis: *Drei Ebenen überlagern sich, von der allgemeinsten zur besonderen: die Standards des Stylesheets, dann Kit-X - Colors.js, falls vorhanden, dann die des Projekts. Jede Ebene definiert nur neu, was sie ändern will. Eine in Kit-X abgelegte Palette folgt dem Nutzer von Projekt zu Projekt; eine Projektpalette gilt nur für dieses.*

6.6 Bedingter Aufruf am Kopf des Generators

Jeder Projektgenerator lädt beim Start das *Registry* und zieht jedes *Datenmodul* nur heran, wenn dessen Flag gesetzt ist. Dieses Muster ist das unmittelbare technische Gegenstück zu den Schaltern in Registry.requires — es ersetzt die frühere Regel, nach der die *Setter* ausnahmslos aufgerufen wurden.

```
// ---- 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);
}
```

Hinweis: *Ist ein Flag false, fehlt die zugehörige Datei im Projekt, und für dieses Modul wird weder ein require() noch ein Setter ausgeführt. Die untätigen Passen bleiben stumm — die Textfunktionen arbeiten für die übrigen aktiven Passen fehlerfrei weiter. Die Disziplin wird von kit_check_setters.py geprüft, siehe Quality Control §3.4.*

6.7 Prüfprotokoll nach einer Kit-Einspielung

Meldet der Nutzer eine Kit-Einspielung, wendet die *KI* dieses Protokoll vor jeder anderen Arbeit auf die *Datenmodule* an. Es ergänzt die allgemeine Abweichungsdiagnose des Aktualisierungsleitfadens §4.4, deren modulbezogene Ausprägung es ist.

- **§6 vollständig lesen:** die normative Struktur kann sich mit dem eingespielten Kit geändert haben.
- **Jedes vorhandene Modul prüfen:** das Vorhandensein aller Pflichtfelder und die Vertragstreue der Exporte nach §6.1 bis §6.4 nachweisen.
- **Glossary Terms besonders prüfen:** das Feld Marke bei jedem Eintrag, und das Format surface und anchor bei glossarySearchTerms. Ein aus einem älteren Kit stammendes Projekt kann ein altes Format tragen, in dem die Marke fehlt und glossarySearchTerms termKey statt anchor führt. Die Migration ergänzt jede Eintragung um die Marke über _slugify(term), kollisionssicher, und aktualisiert den Export.
- **Jede Abweichung melden:** in der Form “Abweichung in [Datei] festgestellt: [Beschreibung]. Vorgeschlagene Migration: [Maßnahme].”
- **Die Bestätigung abwarten:** keine Migration wird ohne ausdrückliche Zustimmung angewendet, Migration für Migration.

6.8 Modul der Dokumentbezeichnungen

Der Dateiname ist ein Bezeichner: er ändert sich nicht von Sprache zu Sprache. Die angezeigte Bezeichnung folgt dagegen der Sprache des Lesers — auf der *Startseite* der Website, in einem Verweis von einem Dokument auf ein anderes und im Untertitel eines Deckblatts. Ohne dieses Modul liest ein deutscher Leser französische Titel.

Datei: [Präfix] - Document Titles (TS).js, gesteuert durch den Schalter requires.documentTitles. Sie exportiert die folgenden Tabellen.

- **titleEntries.** Ein Eintrag je Dokument, Schlüssel = Dateiname ohne *Zeitstempel* und ohne Sprachkürzel. Werte: das Sujet in FR, DE, EN und LU; alias, im Korpus verwendete Kurzformen; aliasSection, Einwortformen, die nur vor einem Abschnittsverweis zählen; slug, Bezeichner der HTML-Seite oder null, wenn das Dokument nicht

veröffentlicht wird; multilingue, wahr, wenn der Slug ein Sprachkürzel trägt.

- **categoryLabels.** Die übersetzten Dateinamen-Kategorien — die Tabelle aus Namenskonvention §5.1, ausführbar gemacht. Sie dient dazu, die vollständige Bezeichnung eines Dokuments und den Untertitel seines Deckblatts zu bilden.
- **sectionTitles.** Die Registertitel des Ordners, nach Nummer und Sprache. Das Inhaltsverzeichnis des Deckblatts und der sprachliche Index der Website lesen sie hier. Sie lebten im Site-Generator, außer Reichweite des *Deckblatt*-Renderers, der daher in einem deutschen *Ordner* französische Titel anzeigte.
- **coverHeader.** Die zwei Zeilen des Deckblattbanners, nach Sprache. Sie stammen aus den Rendering-Konstanten des .docx, das nur eine Fassung trägt: ein deutsches Rendering zeigte daher ein französisches Banner.
- **Das Feld lecture von titleEntries.** Der Name eines Dokuments, wie man ihn in einem Satz schreibt — Der Aktualisierungsleitfaden. Von Hand erfasst, Sprache für Sprache: ein Artikel und mitunter eine Präposition lassen sich nicht aus dem Sujet ableiten. Das Inhaltsverzeichnis des Ordners und der Index der Website verwenden es; der Dateiname bleibt die technische Form.

Ein leerer Wert lässt Bezeichnung UND Adresse auf die Projektsprache zurückfallen: eine französische Bezeichnung auszuliefern und dabei auf eine nicht existierende Seite zu zeigen wäre schlimmer als nicht zu übersetzen. Die Füllregel: jedes Dokument der Projektsprache steht darin.

Hinweis: *Der Generator übergibt titleEntries und categoryLabels über setDocumentTitles an das Stylesheet, wie bei Sprache, Glossar und Farben: nur diese beiden braucht es, da es Dokumentbezeichnungen zusammensetzt. sectionTitles und coverHeader dienen allein dem Deckblatt-Renderer und dem Site-Generator, die das Modul unmittelbar lesen. Das Stylesheet bindet keine Datendatei ein.*

6.9 Text Highlights — Hervorhebung

[Präfix] - Text Highlights (TS).js exportiert highlightEntries, eine flache Liste von Textstellen. Jede erklärte Stelle wird an allen ihren Vorkommen kursiv gesetzt, in allen Dokumenten des Projekts, ohne Auszeichnung. Flag textHighlights, Standard false.

- **Was hineingehört.** Eine Einheit, die an jedem Vorkommen eine ist und weder eine Marke noch ein Glossarbegriff: eine Zeitung, eine Organisation, der Name eines Modells künstlicher Intelligenz, eine fremde Software.

– **Vorrang.** Die Passe steht nach den Marken und vor dem Glossar. Eine Stelle, die bereits eine dieser Tabellen trägt, behält ihr Rendering: ein doppelter Eintrag bringt nichts hinzu und nähme einem Begriff seinen Link.

Hinweis: *Die Passe zeichnet alle Vorkommen aus. Eine Betonung, deren Reichweite vom Satz abhängt, hat in dieser Tabelle daher nichts zu suchen: sie wird an der gewünschten Stelle geschrieben — General Reference §5.5 und §5.6.*

7 Projektglossar

Das Glossardokument ist das öffentliche Gesicht des Projektglossars. Es bildet ein Couplet mit [Präfix] - Glossary - Terms (TS).js — beide Dateien teilen denselben *Zeitstempel* und werden stets zusammen neu erzeugt.

Das Glossar wird stets aus Terms.js neu erzeugt, nie umgekehrt. Die .docx-Datei ist eine visuelle Wiedergabe des .js-Moduls — jede inhaltliche Änderung, das Hinzufügen, Entfernen oder Ändern eines Begriffs, geht zuerst durch die .js, und die Neuerzeugung der .docx folgt mechanisch mit einem gemeinsamen frischen *Zeitstempel*.

Ein Projekt, das sein Glossar in mehreren Sprachen veröffentlicht, erzeugt ein Dokument je Sprache. Das Couplet wird dann zu einem Begriffsmodul und n Dokumenten, die alle denselben *Zeitstempel* tragen und zusammen neu erzeugt werden. Einen Begriff hinzuzufügen oder eine einzige Sprache zu korrigieren zwingt daher zur Neuerzeugung des Ganzen. Die Benennung dieser Dokumente regelt Namenskonvention §2.1 und §2.4: Kategorie und Sujet in der Projektsprache, Sprachkürzel nach dem Sujet. Dieses Dokument wiederholt das nicht — eine Regel hat einen einzigen Wohnsitz.

Jede Variante hat ihren eigenen Eintrag in Registry.documents mit ihrem *Zeitstempel*, sodass die HTML-Veröffentlichung nur die tatsächlich geänderten Seiten neu veröffentlicht.

Die Dateien fehlen im Projekt, wenn Registry.requires.glossary false ist. Die Formatierungskette arbeitet ohne sie — kein Begriff wird dann teal kursiv gesetzt.

8 Projekt-Cover-Sheet

Couplet [Präfix] - Documentation - *Cover Sheet* (TS).docx und [Präfix] - Documentation - *Cover Sheet* (TS).png — *PCL*-Konstanten und erzeugtes *Deckblatt*. Flag Registry.requires.coverSheet, Standard true. Beide Dateien teilen denselben *Zeitstempel*.

Das PNG wird vom *Python-Renderer* kit_render_cover_sheet.py erzeugt — zustandslos und deterministisch. Jede visuelle Änderung geht ausschließlich über die *PCL*-Konstanten der .docx, nie über den Code des Renderers. Der vollständige Vertrag des Renderers und das Verzeichnis aller *PCL*-Konstanten stehen im *Cover Sheet*.

8.1 Gerüst der .docx

Die Datei [Präfix] - Documentation - *Cover Sheet* (TS).docx enthält die normativen Abschnitte der Kit-Vorlage. Jedes Projekt kopiert das Kit Cover Sheet.docx und passt nur §2 und §3 an — die übrigen Abschnitte bleiben mit der Kit-Vorlage identisch.

Abschnitt	Inhalt	Projektanpassung
§1 Gegenstand	Beschreibung der Rolle des Cover Sheets — *Deckblatt* des Papierordners des Projekts	Keine — wortgleiche Kopie der Kit-Vorlage
§2 Organisation des Ordners	Tabelle der Projektrubriken — Nummer, Registertitel, zugehörige Dokumente, *Register*	Vollständige Anpassung — siehe §8.2
§3 *PCL* — projektspezifische Konstanten	Tabelle der vom Projekt angepassten *PCL*-Konstanten	Siehe §8.3 — typischerweise nur HEADER_TEXT
§4 Aktualisierung dieses Dokuments	Regeln zur Aktualisierung des Couplets — frischer gemeinsamer *Zeitstempel* für .docx und .png	Keine — wortgleiche Kopie der Kit-Vorlage

8.2 Anpassung §2 — Rubriken des Projekts

Die Tabelle §2 des Projekt-Cover-Sheets führt die Rubriken des Projekts auf, nicht die des Kits. Die Zahl der nummerierten Rubriken schwankt frei von Projekt zu Projekt, begrenzt durch TAB_COUNT, das 12 *Register* umfasst.

TAB_NUMBERED_MAX ist keine einzuhaltende Grenze, sondern ein abgeleiteter Wert: der *Renderer* berechnet ihn selbsttätig, indem er die aktiven Rubriken aus §2 zählt. Die zugehörige *PCL*-Zeile nützt nur, um zusätzliche *Register* ausdrücklich zu reservieren, etwa fünf aktive Rubriken bei acht farbigen Registern, um drei künftige vorzubereiten.

Je Rubrik: Registernummer, Titel, Liste der zugehörigen Dokumente; die Farbe folgt aus der Nummer — §11. Die aufgeführten Dokumente entsprechen den veröffentlichten Dokumenten des Projekts — ein Dokument steht in genau einer Rubrik.

Hinweis: *Die Liste der Rubriken und der zugehörigen Dokumente wird bei der Einrichtung des Projekts im Dialog festgelegt — siehe Einrichtungsleitfaden §5.*

8.3 Anpassung §3 — projektspezifische Rendering-Konstanten

Die Tabelle §3 des Projekt-Cover-Sheets führt allein die vom Projekt angepassten *PCL*-Konstanten auf. Alle übrigen — A4 bei 300 dpi, Farben, Ränder, Schriften, Registerpositionen — werden unverändert und ohne Verdopplung aus Kit *Cover Sheet* §3 geerbt.

In der Praxis werden nur HEADER_TEXT_1 und HEADER_TEXT_2 regelmäßig angepasst: die beiden Zeilen im orangen Banner des Deckblatts. Der Umbruch zwischen beiden wird beim Schreiben entschieden, nicht beim Rendern berechnet — das Sujet in der ersten Zeile, seine Näherbestimmung in der zweiten. Ein kurzer Text darf nur HEADER_TEXT_1 füllen und die zweite leer lassen.

Ihr Wert folgt der typografischen Konvention des Projekts — typischerweise Satzschreibung oder Title Case, etwa Immobilier — patrimoine et acquisitions, Photogear, Roodt-Hass — domotique. Durchgehende Großschreibung ist nicht zu empfehlen, sie wirkt beim Lesen wie ein typografischer Schrei.

Ausnahme Kit: das Kit zeigt in der ersten Zeile “KIT de documentation” und in der zweiten “assistée par intelligence artificielle”. Das Akronym KIT steht bewusst als Signal in Versalien — das Kit ist die kanonische Referenz für alle Projekte, kein Projekt wie jedes andere. Diese Ausnahme überträgt sich nicht auf Projekte.

Braucht das Projekt eine weitere angepasste *PCL*-Konstante, steht der neue Wert in §3 des Projekt-Cover-Sheets mit dem ausdrücklichen Vermerk “override Kit”.

8.4 Anleitung zum Duplizieren

Bei der Einrichtung eines neuen Projekts oder beim Hinzufügen des Cover Sheets zu einem bestehenden Projekt folgt das Verfahren fünf Schritten.

- Das Kit Cover Sheet.docx nach [Präfix] - Documentation - *Cover Sheet* (TS).docx mit frischem *Zeitstempel* kopieren.
- §1 Gegenstand und §4 Aktualisierung unverändert lassen — keine Änderung.
- §2 Organisation des Ordners mit den Rubriken des Projekts nach §8.2 neu schreiben.
- §3 *PCL* auf die angepassten Konstanten reduzieren, typischerweise HEADER_TEXT. Die übrigen Kit-Konstanten werden stillschweigend geerbt.
- Das PNG über kit_render_cover_sheet.py erzeugen, mit der Projekt-.docx und dem Projekt-*Registry* als Argumenten. Das Couplet wird gemeinsam geliefert.

Hinweis: *Jede spätere Änderung am Projekt-Cover-Sheet — neue Rubrik, geänderter*

HEADER_TEXT, überschriebene PCL — erzeugt das gesamte Couplet.docx und.png mit einem neuen gemeinsamen Zeitstempel neu.

8.5 Fall: nicht erforderlich

Ist Registry.requires.coverSheet false, hat das Projekt keinen *Papierordner*. In der Praxis selten — alle bisher dokumentierten Projekte besitzen ihr *Cover Sheet*. Ist das Flag false, fehlt das Cover-Sheet-Couplet im Projekt und der *Renderer* kit_render_cover_sheet.py wird nicht aufgerufen.

8.6 Verhältnis zur HTML-Startseite

Die Struktur des Papier-Cover-Sheets und die der HTML-*Startseite* des Projekts werden getrennt deklariert. §2 der Cover-Sheet-.docx ist die kanonische Quelle der Ordnerorganisation; die Konstante SECTIONS in gen-[präfix]-site.js ist die kanonische Quelle der *Startseite*. Kein Skript liest die eine, um die andere zu steuern. Ihre Übereinstimmung ist eine Entscheidung des Projekts: Das Kit hält sie identisch, ein Projekt kann einen verschlankten *Ordner* zusammenstellen. Dass die beiden Verzeichnisse voneinander abweichen, ist so gewollt und kein zu behebender Fehler: Das Papier trägt einen *Ordner* in der Basissprache, die Website veröffentlicht, was sich online liest, in allen Sprachen. Das Optische — Farben, Geist und Leseschlüssel — bleibt in jedem Fall erhalten. Die vollständige Regel steht in *Pipeline HTML* §2.5. Jede stille Angleichung der beiden Strukturen unter der Annahme einer Abweichung ist in *Quality Control* §6 als Anti-Muster verzeichnet.

9 Projekt-Pipeline HTML

Die Datei [Präfix] - Documentation - *Pipeline* HTML (TS).docx gehört jedem Projekt eigen — sie dokumentiert die Veröffentlichungsentscheidungen der *Subsite* des Projekts. Sie ersetzt nicht *Pipeline HTML*, das die Wahrheitsquelle des Formats und der gemeinsamen technischen Regeln bleibt.

Verbindlicher Mindestinhalt: sieben Veröffentlichungsentscheidungen müssen darin stehen.

- **Verzeichnis der Dokumente:** Liste der Dokumente des Projekts, die für die HTML-Veröffentlichung in Frage kommen — eine Zeile je Dokument.
- **Flag grau oder nicht grau:** je Dokument die Entscheidung, die Zeile grau kursiv anzuzeigen — Dokument aufgeführt, aber auf der *Startseite* nicht verlinkt — oder als anklickbaren Link auf seine HTML-Seite.
- **Flag mit oder ohne PDF:** je veröffentlichtem Dokument die Entscheidung, eine PDF-Fassung zu erzeugen und das zugehörige Symbol in der Leiste des Dokuments und auf der *Startseite* anzuzeigen.
- **Subdomäne des Projekts:** Wurzel-URL der *Subsite* — von renderLandingPage.homeHref und von den Rücklinks jeder Dokumentseite verwendet.

- **Landing quote:** Zitattext auf der *Startseite* der *Subsite*. Wörtliche Festlegung in §2.4 der Projekt-*Pipeline HTML*, an derselben Stelle wie in Kit *Pipeline HTML* §2.4 für die Kit-Website. Der Site-Generator des Projekts übernimmt diesen Text unverändert in seine Konstante LANDING_QUOTE.
- **Anhänge:** Liste der PDF der Kategorie Anhang, im Wurzelverzeichnis des Projekts abgelegt, jedes mit einer kurzen Kit-.docx gekoppelt, die seinen Inhalt beschreibt. Das Stück wird über den Schlüssel piece im *Registry* erklärt; der Generator kopiert es unter seinem ursprünglichen Namen nach assets/pieces/ und erzeugt die .html aus der gekoppelten .docx, deren Schaltfläche das Stück liefert. Siehe *Pipeline HTML* §6.3.
- **Handbücher:** Liste der PDF der Kategorie Handbuch — Originaldokumente des Herstellers oder Erzeugers — jedes mit einer kurzen Kit-.docx gekoppelt, die Gerät und Gebrauch beschreibt. Gleiches Ketten-Verfahren wie bei den Anhängen.

Pipeline HTML liefert das Formatmuster für diese Abschnitte sowie die gemeinsamen technischen Regeln — Aufbau des Ordners html, verbindliches externes CSS, assetsBase, Konventionen des HTML-Glossars. Die Projekt-*Pipeline HTML* ergänzt diese Regeln um die projekteigenen Entscheidungen.

10 Abhängigkeitskette und Lebenszyklus

10.1 Hauptkette

Die Dateien des Projekts hängen in einer strengen Kette zusammen. Sie zu verletzen erzeugt technisch gültige, aber visuell falsche Dokumente.

- **Datenmodule zur Kette:** Brands.js speist setBrands und liefert Kapitälchen fett in Passe 1; Glossary Terms.js speist setGlossaryTerms und liefert Teal-Kursiv in Passe 2; Variable Friendly Names.js speist setVariableNames und liefert Dunkelgrün-Kursiv in Passe 3; HASS Entities.js speist setHassEntities und liefert Violett-Kursiv in Passe 4. Die URL-Erkennung in Passe 5 läuft selbsttätig, ohne Modul.
- **Kit-Stylesheet zum erzeugten Dokument:** jedes .docx wird von einem **Node.js**-Skript erzeugt, das das aktive Kit-*Stylesheet* per require() einbindet.
- **PCL zum PNG:** das visuelle Rendering des Deckblatts hängt ausschließlich von den *PCL*-Konstanten der Cover-Sheet-.docx ab, die der *Python*-*Renderer* auslegt.
- **PCL zur Startseite:** dieselben *PCL*-Konstanten speisen HEADER_TEXT und TAB_COLORS auf der HTML-*Startseite*.
- **HTML-Stylesheet zum CSS:** getCSS() erzeugt die Datei assets/kit-style.css, auf die alle HTML-Seiten der *Subsite* extern verweisen.

10.2 Eine Kit-Aktualisierung übernehmen

Meldet die *KI* eine Kit-Aktualisierung, folgt die Übernahme drei Schritten, nach Abhängigkeiten geordnet. Kein Abkürzen erlaubt.

- Kit - *Registry.js* lesen, um die aktiven Versionen von General, Glossary, YAML und HTML sowie die *Zeitstempel* der Kit-Dokumente zu erhalten.
- Jede aktive Version mit der vergleichen, die bei der letzten *Erzeugung* eines Projekt-Couplets verwendet wurde — maßgeblich ist die als Fingerabdruck in die Projekt-.docx eingetragene StylesheetVersion.
- Für jedes von einer Versionsabweichung betroffene Projekt-Couplet das Couplet mit dem passenden frischen *Zeitstempel* neu erzeugen. Die Neuerzeugung folgt der Kette aus §10.1.

Hinweis: *Projektdateien wurden historisch erst nach den Kit-Dateien aktualisiert. Dieses Modell entfällt. Die geltende Regel lautet: ein Projekt übernimmt eine Kit-Aktualisierung, sobald es beansprucht wird, nach dem obigen Verfahren. Das ausführliche Protokoll der Abweichungsdiagnose steht im Aktualisierungsleitfaden §4.4; seine Ausprägung für die Datenmodule in §6.7.*

11 Steuerung

Die folgenden Regeln bestimmen die Entwicklung jedes Projekts, das das Kit einsetzt. Ohne ausdrückliche *Anweisung* wird keine Ausnahme gewährt.

- **Kit-Präfix reserviert:** Die *KI* erzeugt und ändert keine Datei mit dem Präfix Kit, außer der Nutzer arbeitet ausdrücklich an einer Weiterentwicklung des Kits selbst.
- **Unverletzliche Couplets:** Kit-Stylesheet.js und Reference.docx; Kit Cover Sheet.docx und.png; Projekt-Cover-Sheet.docx und.png; Projekt Glossary Terms.js und Glossaire.docx. Jede Neuerzeugung eines Mitglieds zieht die des anderen mit demselben frischen *Zeitstempel* nach sich. Vollständiges Verzeichnis: *Prompt* §6.
- **Generator von Grund auf:** kein *Ad-hoc-Generator* gen-*.js wird von einer Sitzung zur nächsten wiederverwendet. Jede Sitzung geht vom aktiven Kit-*Stylesheet* und seiner Reference.docx aus, das Skript wird vollständig neu geschrieben. Stabile Artefakte — §12 und *Quality Control* §3 — unterliegen dieser Regel nicht: Sie werden mit dem Kit geliefert, damit ein Projekt von Anfang an alles Nötige hat, und ändern sich nur, wenn sich ihre Logik ändert.
- **Nur PCL-Renderer:** jede visuelle Änderung am *Cover Sheet* geht über die *PCL*-Konstanten der .docx, nie über den Code des Renderers. Der *Renderer* ist zustandslos und deterministisch: gleiche *PCL*-Konstanten ergeben dasselbe PNG.
- **Registry-Flags — einzige Quelle:** der Projekt-*Prompt* darf die in Registry.requires vorhandenen Flags nennen, verdoppelt aber nie deren Wert.

Jedes andere Dokument, das ein Flag erwähnt, muss auf das *Registry* als Referenz verweisen.

- **Infrastruktur der Subsite:** die .htaccess und die robots.txt einer *Subsite* gehören dem Projekt **sliver.lu**. Weder ein Kit- noch ein Projektgenerator erzeugt sie, und sie gehen nie durch ein Veröffentlichungs-ZIP. Verzeichnis der erwarteten und verbotenen Dateien: *Prompt* §16.
- **Lieferung in einem Archiv:** die in einer Sitzung erzeugten Dateien werden in einem einzigen Archiv geliefert, das den vollständigen Zustand des Projekts trägt, nie Datei für Datei. Der Empfänger ersetzt einen *Ordner*, statt Teile zusammenzusetzen, und seine Sicherung kann nicht von der Arbeitsfassung abweichen.
- **Zusammensetzung einer Lieferung:** die folgenden Archive, im selben Austausch übergeben. Das Archiv des vollständigen Projekts, immer — es trägt den ganzen Zustand und ersetzt den vorigen *Ordner*. Das Veröffentlichungsarchiv der Website, sobald ein Durchlauf stattgefunden hat, vollständig oder inkrementell. Ein weiteres, wenn das Projekt eine Kit-X-Erweiterung trägt: es wird getrennt geliefert, denn es gehört nicht an denselben Ort — es folgt seinem Autor, nicht dem Projekt — §3.

Hinweis: *Die Reihenfolge der Ablage wird genannt, wo sie zählt. Das Download-Archiv geht vor dem Website-ZIP nach assets/downloads/: danach abgelegt, führt das Download-Symbol so lange auf eine fehlende Seite, bis das Archiv eintrifft.*

- **Projektarchiv:** das Archiv des gesamten Baums — Werkzeuge, Stylesheets, Module, Dokumente, *Registry* —, unter dem im *Registry* in projectZip.filename erklärten Namen. Es wird vom Veröffentlichungsdurchlauf erzeugt, wenn projectShareable true ist, unter assets/downloads/ der Website abgelegt, und es ist dieses Archiv, das geliefert wird: Ein gesondertes Arbeitsarchiv gibt es nicht mehr.
- **Eigenständiges vollständiges Projekt:** das Archiv enthält alles, was das Projekt braucht, um beim Empfänger zu laufen, ohne etwas aus einer Sitzung holen zu müssen.
- **npm-Module:** eine package.json im Stammverzeichnis erklärt jedes Modul und seine Mindestversion. Jede Installation erfolgt neu, mit *npm* install, nie durch Kopieren eines node_modules.
- **Versionsnummer:** sie steigt, wenn sich die Struktur des Projekts oder die Anweisungen ändern, die sein Verhalten regeln: Organisation der Dateien und Rubriken, Namenskonventionen, *Prompt*, Prüfregeln, Stylesheets. Die zweite Ziffer folgt diesen Änderungen; die erste steigt nur, wenn ein Projekt, das das Kit verwendet, migriert werden muss, um ihm zu folgen. Eine Textkorrektur, eine Übersetzung oder eine erneute Veröffentlichung der Website lassen sie nicht steigen. Jede Erhöhung wird mit ihrem Grund im Changelog des *Registry* vermerkt. Das Datum projectZip.updated ändert sich mit ihr. Jede Lieferung,

die eine Datei ändert, erhöht die Nummer, Textkorrekturen eingeschlossen: Zwei Archive mit derselben Nummer und unterschiedlichem Inhalt machen die Unterprojekte unlesbar.

- **Neuerzeugung der Website:** ein Erzeugungsdurchlauf der Website wird ebenfalls in einem Archiv geliefert, ob der Durchlauf vollständig oder inkrementell ist. Die Unterscheidung bestimmt, was das Archiv enthält, nicht die Form der Übergabe.
- **Benennung der Archive:** ein geliefertes Archiv trägt einen frischen *Zeitstempel*, und sein Name folgt dem Muster seiner Familie — Namenskonvention §4.5, die die Fälle auflistet. Das Projektarchiv bildet die Ausnahme: Sein Name stammt aus Registry.projectZip.filename und trägt eine Versionsnummer, weil er eine veröffentlichte Adresse ist — der Link der Website ändert sich nur mit dieser Nummer.
- **Gleiche Verzeichnisstruktur:** die Dokumentation eines Projekts zeigt sich wie die des Kits — Verzeichnis nach Rubriken, jedes Dokument in HTML und PDF, Sprachsymbole bei Dokumenten mit Varianten, Download-Symbol des Projektarchivs. Die einzige zulässige Abweichung betrifft die Farben des Themas — Leiste, Titel, Links — über das Farbmodul. Die sechs Registerfarben, im Kreis aus der Rubriknummer abgeleitet, gehören dem Papier: Dort unterscheiden sie die Trennblätter des Ordners. Die *Startseite* trägt sie nicht — die Nummer einer Rubrik steht dort in COVER_NUM_COLOR, einer einzigen Farbe für die gesamte Website, je Projekt einstellbar wie die der Leiste.
- **Projektjournal:** jedes Projekt führt ein Journal der offenen Einträge, der Schulden und der datierten Entscheidungen. Aufbau und Inhalt in §14. Es ist das einzige Dokument des Projekts, das eine Geschichte tragen darf.

12 Stabile ausführbare Projektartefakte

Ein Projekt kann eigene stabile ausführbare Artefakte hervorbringen: Skripte, die Sitzung für Sitzung wiederverwendet werden und im *Arbeitsbaum* des Projekts leben, im Unterschied zu den *ad-hoc-Generatoren*, die jede Sitzung neu erzeugt. Typisches Beispiel: trading_generate_history_report.py.

12.1 Namenskonvention

Unix-Nomenklatur in snake_case. Projektpräfix in Kleinbuchstaben, gefolgt von einem Unterstrich, Name in snake_case, kein *Zeitstempel* im Namen. Formelle Ausnahme zu Namenskonvention §2, in demselben Dokument §3.3 festgehalten. Projektbeispiele: trading_generate_history_report.py, hexi_validate_recipe.py. Anwendungskriterium: stabiles ausführbares *Artefakt* — *Python*, **Node.js**-CLI, Shell. Jede andere Datei folgt dem Hauptmuster.

12.2 Rolle — Prüfer oder Renderer

Zwei mögliche Kategorien. Ein Prüfer gibt Exit-Code 0 zurück, wenn alles konform ist, sonst ungleich null, und blockiert im Fehlerfall die Lieferung. Ein

Renderer erzeugt ein deterministisches Ergebnis: gleiche Eingabe und gleicher *Kontext* ergeben dieselbe Ausgabe. Fällt der Projektbedarf in keine der beiden Kategorien, handelt es sich wahrscheinlich um einen *ad-hoc-Generator* und nicht um ein *stabiles Artefakt* — dann das Präfix gen- und Namenskonvention §3.6 verwenden. Entspricht der Kit-Einteilung aus *Quality Control* §3.

12.3 Lebenszyklus

Das stabile Projektartefakt lebt dauerhaft im *Arbeitsbaum* des Projekts, nicht in dem des Kits. Es wird nicht in jeder Sitzung neu erzeugt — nur wenn sich seine Logik ändert. Seine Entwicklung wird in einem Changelog-Block am Kopf der Datei selbst festgehalten, wie bei den Kit-Stylesheets. Kein externes Nachweisdokument ist vorgeschrieben — es gilt die übliche Unix-Konvention. Das *Artefakt* trägt keinen *Zeitstempel* im Namen.

12.4 Anleitung zur Erstellung

Erkennt ein Projekt einen wiederkehrenden Bedarf, der ein *stabiles Artefakt* verdient, folgt die erstellende Sitzung sieben Schritten.

- Mit dem Nutzer bestätigen, dass der Bedarf wirklich wiederkehrend und nicht einmalig ist.
- Die Datei nach §12.1 benennen.
- Das eigenständige Skript schreiben und externe Abhängigkeiten möglichst gering halten.
- Die Abhängigkeiten im Kopf der Datei dokumentieren.
- Den Nutzungsvertrag im Docstring dokumentieren: Eingaben, Ausgaben, Rückgabecodes, Aufrufbeispiele.
- Den Changelog-Block am Kopf der Datei anlegen.
- Über present_files liefern, damit der Nutzer die Datei in den *Arbeitsbaum* des Projekts legt.

12.5 Spätere Weiterentwicklung

Muss ein *stabiles Artefakt* weiterentwickelt werden, sei es wegen eines Fehlers oder einer Erweiterung, ist die Sitzung ihm gewidmet und wird nicht mit anderen Lieferungen vermischt. Die Quelle wird unmittelbar aus dem *Arbeitsbaum* gelesen — die .py- und.js-Dateien werden dort von der Plattform vollständig aufbewahrt, ein erneutes Hochladen ist nicht nötig, anders als bei.docx, die die Quelldatei verlangen. Die Änderung erhöht die Version im Changelog-Block am Kopf der Datei samt Beschreibung. Danach die übliche *sequenzielle Lieferung*: Prüfung, Kopie nach outputs, present_files.

Hinweis: *Ein stabiles Artefakt gehört zur Lieferung des vollständigen Projekts, ebenso wie die Dokumente und die Stylesheets. Ein Archiv, das es ausließe, erlaubte keinen Neubeginn bei null, und der Mangel zeigte sich erst beim Wiederherstellen.*

13 Kit-Fehlerbericht

Ein Projekt kann auf ein Kit-Verhalten stoßen, das eine Meldung verdient: unerwartetes Rendering einer *Stylesheet*-Funktion, Abweichung zwischen dem .js-Code und seiner Reference.docx, ein noch nicht verzeichnetes Anti-Muster, ein falsch positiver Prüferbefund. Der strukturierte Meldeweg ist ein kurzes eigenes Dokument, vom Projekt erzeugt und im *Chat* des Kit-Projekts zur Analyse und Korrektur abgegeben. Es ersetzt die informellen Briefings, die historisch gedient haben — gleicher Inhalt, einheitliche Form.

13.1 Gegenstand und Auslöser

Der Kit-Fehlerbericht entsteht, wenn das Projekt ein Kit-Verhalten feststellt, das sich lokal im Projekt nicht lösen lässt. Es gilt die Regel *Prompt* §5.3: eine lokale Umgehung ersetzt nie eine Korrektur an der Quelle. Liegt die Ursache im Kit, muss das Kit korrigieren.

Drei typische Auslöser.

- **Stylesheet-Fehler:** eine *Stylesheet*-Funktion rendert nicht wie in ihrer Reference.docx beschrieben — Abweichung zwischen Laufzeitverhalten und Vertrag. Beispiel: der schlecht benannte Parameter bgColor von sectionBanner, auf Meldung des Projekts Sorso hin in accentColor korrigiert.
- **Ketten-Fehler:** ein Schritt der Kette liefert ein widersprüchliches oder stillschweigend verstümmeltes Ergebnis. Beispiel: gen-kit-site.js rief die Daten-*Setter* des HTML-Stylesheets nicht auf — die Kit-Website hatte historisch weder Markenauszeichnung noch Glossarlinks.
- **Nicht verzeichnetes Anti-Muster:** ein im Projekt entdeckter wiederkehrender Fehler, der einen Eintrag in *Quality Control* §6 und im Idealfall eine automatische Prüfung in *Schicht 2* verdiente.

13.2 Benennung

Der Fehlerbericht folgt dem Standardmuster aus Namenskonvention §2.1: [Präfix] - Project - Kit *bug report* (TS).docx. Kategorie Project gemäß §7, beiden Präfixen offen. Das Sujet “Kit *bug report*” bleibt in allen Sprachen englisch — stabiler technischer Name im Sinne von §5.2 der Konvention.

Die Kategorie Project wird nach project.docLanguage des meldenden Projekts übersetzt.

Die Datei wird vom Projekt über ein von Grund auf geschriebenes Skript gen-kit-bug-report.js erzeugt, wie jedes Projektdokument — Formatierung mit dem zum Zeitpunkt der Abfassung aktiven General-*Stylesheet* des Kits, Prüfung mit kit_validate_docx am Ausgang.

13.3 Aufbau des Dokuments

Verbindliche Abschnitte, in dieser Reihenfolge. Von Bauart kurz — der Fehlerbericht ist keine Abhandlung: er beschreibt, reproduziert und schlägt vor. Die ausführliche Prüfung und die Korrektur folgen in der Kit-Sitzung.

Der Fehlerbericht ist ein Feststellungsdokument — *Kit Prompt* §12 —, dessen Aufbau hier festgelegt ist: Er berichtet über einen vorhandenen Fehler, glättet nichts, und sein letzter Abschnitt trägt, was noch zu entscheiden ist.

- **§1 Identifikation:** Projektpräfix, Datum der Feststellung, project.docLanguage, die zum Zeitpunkt des Fehlers aktiven Kit-*Stylesheet*-Versionen — aus Kit - *Registry.js* gelesen, da das Projekt-*Registry* sie nicht führt, und unverändert übernommen — sowie die betroffenen Dokumente oder Ketten.
- **§2 Beobachtung:** beobachtetes und erwartetes Verhalten, in zwei kurzen Absätzen. Beschreibt sachlich, was geschah, ohne Deutung.
- **§3 Reproduktion:** minimaler Auszug aus der gen-*.js oder der Befehl, der den Fehler auslöst, als rawBlock, dazu die genau ausgeführte Abfolge. Präzise genug, dass die *KI* auf Kit-Seite ihn ohne Rückfragen in einer Sitzung nachstellen kann.
- **§4 Auswirkung und vorgeschlagene Diagnose:** welche Dokumente oder Ketten betroffen sind und wie schwer. Vermutung des Projekts zur Grundursache, nicht bindend: die *KI* auf Kit-Seite führt die vollständige Analyse erneut. Angeben, ob eine lokale Umgehung besteht oder nicht.

13.4 Meldeweg

Ist der Fehlerbericht erzeugt und projektseitig geprüft, erfolgt die Meldung durch gleichzeitiges Ablegen mehrerer Quelldateien im *Chat* des Kit-Projekts. Das Kit-Projekt hat nur Zugriff auf das, was im *Chat* angehängt ist — nicht auf den *Arbeitsbaum* der Projekte.

- **Der Fehlerbericht .docx:** die Hauptdatei, Quelldatei im Faden. Ein ausgelesener Text genügt für den kanonischen Inhalt einer .docx nicht.
- **Die betroffene gen-*.js:** der Projektgenerator, der den Fehler auslöst, oder ein minimaler Auszug — als Text im *Chat* oder als angehängte.js-Datei.
- **Die Ergebnis-.docx:** erzeugt der Fehler eine schlecht gerenderte.docx, die Quelldatei zur Untersuchung anhängen — Zahl der Tabellen, w:tbl, *custom properties*.
- **Etwaige Bildschirmfotos:** Aufnahmen aus Word, *LibreOffice* oder dem Browser, wenn der Fehler visuell ist — als Bilder angehängt.

Hinweis: *Der Fehlerbericht wird zu Beginn der Kit-Sitzung wie ein Briefing gelesen. Die KI auf Kit-Seite hält ihn gegen die geltenden Regeln, bestimmt die betroffene Schicht, schlägt einen Korrekturplan vor und wartet vor jeder Erzeugung auf das ausdrückliche*

Go. Danach der übliche Zyklus der sequenziellen Lieferung.

13.5 Keine Dauerhaftigkeit

Der Fehlerbericht ist eine flüchtige Datei: erzeugt, geliefert, nach der Lösung archiviert. Er erscheint nicht im Verzeichnis aus §2 — dieses führt die dauerhaften *Projektdateien*, die zwischen Sitzungen überleben, was auf den Fehlerbericht nicht zutrifft. Die Spur der Lösung lebt anderswo: ein Eintrag in den Todos auf Kit-Seite, gegebenenfalls die Aktualisierung eines Kit-Dokuments.

Hinweis: *Aufbewahrung auf Projektseite: freie Wahl des Nutzers. Keine verpflichtende Ablage im Arbeitsbaum — der Fehlerbericht hat seine Arbeit getan, sobald die Kit-Korrektur geliefert ist und das Projekt die neue Fassung des betroffenen Couplets nach §10.2 übernommen hat.*

14 Projektjournal

Jedes Projekt führt ein Journal. Es trägt, was noch zu tun ist, und die datierte Spur dessen, was entschieden wurde — das erlaubt es, Monate später wiederzufinden, warum eine Sache so und nicht anders gemacht wurde.

Es folgt dem Muster von *Kit - Projet - Todos*, das dafür das Referenzexemplar ist. Seine Abschnitte, in dieser Reihenfolge.

- **§1 Offene Fehler.** Was kaputt und nicht repariert ist. Ein Eintrag je Fehler, mit dem, was er erzeugt, und dem, was er verhindert. Der Abschnitt nennt die Zahl der offenen Einträge und sein Datum, damit ein “keine Einträge” als datierte Feststellung gelesen wird.
- **§2 Baustellen.** Was eine Entscheidung verlangt, bevor es getan wird, oder eine Arbeit, deren Umfang eine Sitzung übersteigt. Ein Eintrag nennt, was blockiert, nicht, was man sich wünscht.
- **§3 Technische Schulden.** Was funktioniert und von dem man weiß, dass es unvollkommen ist. Eine Schuld, die heute niemanden stört, bleibt eine Schuld: sie wird mit dem Grund eingetragen, sie jetzt nicht zu tilgen.
- **§4 Verzeichnis der Lösungen.** Eine Tabelle mit vier Spalten — Nummer, Titel, Datum, Inhalt — in fortlaufender Nummerierung, die nie wiederverwendet wird. Jeder Eintrag sagt, was entschieden wurde und warum, samt der gelieferten Versionen.

Hinweis: *Was eingetragen wird: die strukturellen Entscheidungen. Eine Formänderung, eine gesetzte oder zurückgenommene Regel, ein behobener Fehler, ein entschiedener Streitpunkt. Was nicht eingetragen wird: Erzeugungsdurchläufe, Veröffentlichungen, Datenaktualisierungen — sie hinterlassen ihre Spur in den Zeitstempeln, nicht im Journal.*

Hinweis: *Das Journal ist das einzige Dokument des Projekts, das eine Geschichte tragen darf. Die anderen beschreiben einen Zustand; trügen sie auch ihre eigene Vergangenheit, wüsste ein Leser nicht mehr, welches von beiden maßgeblich ist.*

15 Sprachauswahl

Eine durchgehend mehrsprachige Website und eine Website, von der nur einige Dokumente übersetzt sind, haben nicht denselben Bedarf. Das Kit bietet beide Mechanismen; das Projekt erklärt unter rendering.languageSelector, welchen es verwendet.

Modus	Was er voraussetzt	Wo die Auswahl erscheint
site	Jede veröffentlichte Seite besteht in jeder erklärten Sprache	Menü am Ende der Leiste, auf allen Seiten gleich
document	Nichts; nur einige Dokumente sind übersetzt	Symbole auf Dokumentebene, dort wo Varianten bestehen

Hinweis: *Der Standard ist document: er setzt nichts über das Projekt voraus. Ihn zu verfehlen zerbricht nichts und zeigt sich auf der ersten geöffneten Seite — der Generator sagt den Modus überdies bei jedem Durchlauf an. Ein lauter Abbruch bleibt den Fehlern vorbehalten, die sich nicht zeigen.*

Hinweis: *Im Modus site ließe ein nicht übersetztes Dokument das Menü von Seite zu Seite erscheinen und verschwinden, und es wäre kein Anhaltspunkt mehr. Ein Projekt, das einsprachige Dokumente veröffentlicht, verwendet den Modus document.*

- **Was das Projekt liefert.** Die Liste der Varianten jeder Seite — Sprachcode und Adresse. Das Kit liefert die beiden Mechanismen, ihre lokalisierten Bezeichnungen und ihre Flaggen.

16 Basissprache und Übersetzungen

Das Projekt erklärt seine Basissprache unter project.docLanguage. Die in dieser Sprache verfassten Dokumente sind maßgeblich: Eine Regel, eine Korrektur oder eine Entscheidung wird zuerst dort geschrieben, und von ihnen aus wird gearbeitet, von Menschen wie von der *KI*.

Übersetzungen sind ein Dienst am Leser. Sie geben den Sinn wieder und lesen sich wie ein in ihrer Sprache geschriebener Text, nie Wort für Wort. Nichts geht verloren, nichts kommt hinzu; Dateinamen, Code und Abschnittsnummern bleiben unverändert.

Jede Variante trägt am Anfang ihres §1 den festen Hinweis aus *Kit Prompt* §17, mit der Basissprache des Projekts. Eine Übersetzung wird auf Anfrage aktualisiert, aus der Karte des maßgeblichen Dokuments, und ihr Gerüst wird mit kit_check_ossature.py geprüft.

Vollständige Regel: *Kit Prompt* §17.