# Kit de documentation

## Documentation — HTML Pipeline — EN

# 1 Introduction

This document is a translation in substance. The reference is the original document in French; extensions and changes are always made there.

The site publishes some of the Kit's documents, not all of them: only those declared to the *pipeline* are converted into pages. The others stay in the printed *binder* and appear nowhere online, even after a recent update.

The conversion starts from the Word files (.docx) produced by the *stylesheet* for prose documents, extracts their structure, and renders it as HTML pages with their navigation, their glossary and their *home page*. Each publication produces an archive ready for transfer.

The table below shows what applies in full to every project and what depends on the project, to be filled in at set-up.

| Scope | Section | Project value |
|---|---|---|
| **INVARIANT** | §2.1 Selection criteria for published documents | — |
| **INVARIANT** | §2.3 Delivery rules — tools, delta logic | — |
| **INVARIANT** | §3 *Registry.js* — structure, usage, update rule | — |
| **INVARIANT** | §4 Delta ZIP deployment — logic, naming, lastDeploy | — |
| **INVARIANT** | §5 HTML site structure | — |
| **INVARIANT** | §6 PDF *generation* | — |
| **INVARIANT** | §7 Robustness rules | — |
| **PROJECT** | §2.2 Sections and published documents | List of the project's documents, HTML slugs, PDF activation. Define in *Registry.js*, section documents. |
| **PROJECT** | §4.3 ZIP naming | Replace the Kit's domain with that of the project's sub-site. |

# 2 Publication scope

Not every Kit document is published on the site. The selection is deliberately limited to documents meant to be read by an active Kit user. Internal configuration files — prompts, stylesheets, *PCL* constants — are excluded from direct publication: they remain listed for information.

## 2.1 Selection criteria

- The document is worth reading on its own for a Kit user.

- The document does not expose the Kit's internal configuration.
- The document is stable — neither a draft nor a working document.

## 2.2 Sections and published documents

The site is organised into numbered sections. For the Kit, the *paper binder* takes up the same sections, with the same titles and the same documents — see §2.5. The stylesheets do not appear there: their documents are found in the downloadable archive.

| # | Title | Documents | HTML file | PDF |
|---|---|---|---|---|
| **01** | Before you begin | Kit — Documentation — Comment documenter ses projets | comment-documenter-LANG.html | yes |
| | | Kit — Documentation — Reading Guide | reading-guide-LANG.html | yes |
| **02** | Working with **Claude AI** | **Claude AI** — Documentation — Manuel | claude-manuel-LANG.html | yes |
| | | **Claude AI** — Documentation — Guide Pratique | claude-guide-pratique-LANG.html | yes |
| | | **Claude AI** — Documentation — Environnement | claude-environnement-LANG.html | yes |
| **03** | Working with *Grok AI* | *Grok AI* — Documentation — Manuel | grok-manuel-LANG.html | yes |
| | | *Grok AI* — Documentation — Guide Pratique | grok-guide-pratique-LANG.html | yes |
| | | *Grok AI* — Documentation — Environnement | grok-environnement-LANG.html | yes |

| # | Title | Documents | HTML file | PDF |
|---|---|---|---|---|
| | | *Grok AI* — Documentation — Prompts de dialogue | grok-prompts-de-dialogue-LANG.html | yes |
| **04** | Working with **ChatGPT AI** | **ChatGPT AI** — Documentation — Guide Pratique | chatgpt-guide-pratique-LANG.html | yes |
| | | **ChatGPT AI** — Documentation — Environnement | chatgpt-environnement-LANG.html | yes |
| **05** | Starting and maintaining a project | Kit — Projet — Guide d'initialisation | guide-initialisation-LANG.html | yes |
| | | Kit — Projet — Guide de mise à jour | guide-maj-LANG.html | yes |
| | | Kit — Projet — Convention de nommage | convention-nommage.html | yes |
| **06** | Steering the *AI* and adapting the Kit to your needs | Kit — Documentation — Prompts de dialogue | prompts-de-dialogue-LANG.html | yes |
| | | Kit — Documentation — Étendre le Kit | etendre-le-kit-LANG.html | yes |
| **07** | Files and structure | Kit — Documentation — Environnement d'exécution | environnement-execution-LANG.html | yes |
| | | Kit — Projet — *Prompt* | kit-prompt-LANG.html | yes |
| | | Kit — Projet — Structure commune | structure-commune-LANG.html | yes |

| # | Title | Documents | HTML file | PDF |
|---|---|---|---|---|
| | | Kit — Documentation — *Pipeline* HTML | pipeline-html-LANG.html | yes |
| | | Kit — Documentation — Quality Control | quality-control-LANG.html | yes |
| | | Kit — Glossaire — Termes | glossaire-LANG.html | yes |
| **08** | Publishing and sharing a project | Kit — Documentation — Publication et reprise | publication-reprise-LANG.html | yes |

**Note:** *LANG stands for the language suffix: comment-documenter-fr.html, comment-documenter-de.html, comment-documenter-en.html. An untranslated document has a slug without a suffix.*

## 2.3 Delivery rules

Each *generation* produces either a full pass — first deployment or lastDeploy null — or a *delta pass* for every later deployment. The logic is driven by the *Registry*. See §4 for details.

| File produced | Tool | Full pass | Delta |
|---|---|---|---|
| **html/*.html — document pages** | **Node.js**, *pandoc AST*, HTML *stylesheet* | All | Only the .docx changed since lastDeploy |
| **glossaire.html** | Kit helper kit_gen_glossaire_html.js from the terms module, never via the *AST* | Always | Always |
| **index.html and index-{langue}.html — loaders** | **Node.js**, pageChargeur() | Always | Always |
| **assets/kit-style.css** | **Node.js**, getCSS() | Always | If the HTML *stylesheet* has changed since lastDeploy |

| File produced | Tool | Full pass | Delta |
|---|---|---|---|
| **assets/pdf/*.pdf** | *LibreOffice* CLI from the binary .docx | If PDF is enabled | With the matching .html |
| **assets/pieces/* — appendix pieces** | Direct copy from the project root, declared by the piece key | If declared | The sheet produces its own PDF, like any document |
| **assets/png/{slug}/*.png** | Extracted by *pandoc* | With the .html | With the matching .html |
| **.htaccess** | Not produced — deposited by the parent site | Out of scope | Out of scope |
| **robots.txt** | Not produced — deposited by the parent site | Out of scope | Out of scope |

**Note:** *The generator locates each source file (.docx) by a partial match on the file name. The pattern must be unambiguous. Apostrophes in file names are always straight apostrophes, never typographic ones. Detailed rules: HTML Reference §14.10.*

A document published in several languages produces one page per variant, each with its suffixed slug and its own entry in Registry.documents. The delta compares slug by slug, so each language is republished independently of the others. Where the language selector appears follows the mode declared in the *Registry* under rendering.languageSelector — Common Structure §15.

## 2.4 Home page — introductory block

The *landing page* shows a quotation block above the list of sections. This text is the official source of truth — the index generator copies it verbatim into its dedicated constant.

```
Ce site vous aide à vous servir du kit de documentation pour documenter vos projets. Une fois appliqué, le kit produit des documents au format Word, des PDF et des pages web. Il sert de mode opératoire à un moteur d'intelligence artificielle, celui de votre choix.
```

The text is passed to the landing generator through the configuration field provided for it. If the field is empty or absent, the block is hidden automatically — no orphaned orange bar.

The same rule applies to *consumer projects*: each project's *Pipeline HTML* defines its own text in §2.4, copied verbatim by its site generator.

### 2.5 Paper Cover Sheet and HTML landing page — two independent indexes

The two indexes — §2 of the *Cover Sheet* document and the section constant in the site generator — are declared separately. No script reads one to drive the other.

Whether they match is the project's choice. The Kit keeps them identical: the same sections and the same documents in the same reading order, the *binder* carrying only the base language. A project may also put together a lighter *binder* that takes up only part of what the site publishes.

The HTML *landing page* remains the web index: it enriches each entry with publication metadata, and what is not read online is left out — stylesheets, *Registry*, *Cover Sheet*, *data modules*.

The shared look — section colours, spirit, reading key — is kept in every case.

**Note:** *A divergence between the two indexes is not a drift for a script to correct. Silent synchronisation on the assumption of drift is catalogued as an anti-pattern in Quality Control §6; for the Kit, the match is checked by reading.*

The site generator follows the pattern gen-{projet}-site.js, one per project. The Kit has its own, separate from those of *consumer projects*. It is not a reusable stable Kit *artifact*: by nature it is specific to the project it publishes.

**Note:** *The Kit's site generator exposes no public API. Its internal helpers and constants are tied to the Kit's structure and are not exported. Consumer projects write their own generator from scratch — a direct consequence of the gen- prefix, Naming Convention §3.6.*

## 3 Registry — central source of truth

The *Registry* is the *pipeline*'s single point of configuration. It holds the *stylesheet* versions, the *generation* timestamps of each document, the rendering flags, the compilation notice and the last deployment. Any information that could be derived for consistency is read here.

### 3.1 Structure

The *Registry* exports an object whose sections are described in the table below. The file has no *timestamp* in its name — it is the source of timestamps. The *Registry* of a *consumer project* has a slightly different structure, documented in Common Structure §5: it carries no stylesheets section, since projects leave version tracking to the Kit's *Registry*.

| Section | Keys | Content |
|---|---|---|
| **project** | docLanguage, documentAuthor, webAuthor, documentSiteBase | Identity and language of the Kit. Drives the stylesheets' *L10N* selection and the *Cover Sheet renderer*'s localised constants. Also holds the author of documents and pages, and the root for cross-document references. |
| **requires** | coverSheet, readingGuide, glossary, textHighlights, documentTitles, brands, variableNames, hassEntities, colors | Presence flags for *data modules* and optional files. Read by generators to make require() and *setters* conditional. |
| **stylesheets** | general, glossary, yaml, html | Semantic version, *timestamp* of the .js and .docx *pair*, and for General the minDocumentVersion floor below which a published document is obsolete. |
| **familyDomains** | array of hostnames | Domains whose links stay captured in the *Android* app's *WebView*. Read by setFamilyDomains(). |
| **documents** | one slug per document | *Timestamp* of the last *generation*. One entry per language variant for translated documents. |
| **rendering** | homeHref, languageSelector, coverSheet, toc | Flags driving deliverable *generation*: home icon of the *landing page*, the other pages returning to the index of their language, language selector position, height mode of the *Cover Sheet* sections, *generation* and hiding of the HTML TOC. |

| Section | Keys | Content |
|---|---|---|
| **compiledWith** | object per language or null | Compilation notice in the HTML footer. null or absent: no notice. Never concerns the .docx. |
| **projectZip** | version, filename, updated | Release downloadable from the *landing page*. Version number as per Common Structure §11. |
| **deploy** | siteName, siteInfrastructure, lastDeploy | Name of the published site, which names the archive; origin of the icons, robots.txt and .htaccess; ISO *timestamp* of the last deployment, written only by the site generator. |

```
module.exports = {
  project:       { docLanguage: 'FR' },
  requires:      { coverSheet: true, readingGuide: true, glossary: true,
                   brands: true, variableNames: false, hassEntities: false },
  stylesheets: {
    general:  { version: '…', ts: 'AAAA-MM-JJ - HHhMM',
                minDocumentVersion: '…' },
    glossary: { version: '…', ts: 'AAAA-MM-JJ - HHhMM' },
    yaml:     { version: '…', ts: 'AAAA-MM-JJ - HHhMM' },
    html:     { version: '…', ts: 'AAAA-MM-JJ - HHhMM' },
  },
  familyDomains: ['sliver.lu', 'hexi.lu'],
  documents: {
    'reading-guide': { ts: 'AAAA-MM-JJ - HHhMM' },
    // un slug par document Kit ; un slug par variante de langue
  },
  rendering: {
    coverSheet: { sectionHMode: 'FIXED' },   // FIXED | DYNAMIC
    toc:        { generate: true, hidden: false },
  },
  compiledWith: null,                        // objet par langue, ou null
  projectZip:       { version: '…', filename: '…', updated: 'AAAA-MM-JJ' },
  deploy:       { lastDeploy: null },        // ISO 8601 UTC
};
```

## 3.2 Use in generators

Every generator that uses cross-references loads the *Registry* at start-up and reads the values dynamically.

```
const registry = require('./Kit - Registry.js');

const v    = registry.stylesheets.general.version;   // version active
const ts   = registry.documents['reading-guide'].ts; // horodatage document
```

```
const mode = registry.rendering.coverSheet.sectionHMode;
const last = registry.deploy.lastDeploy;            // null ou ISO
```

**Note:** *Version references in running text use string templates. A version change in the Registry propagates automatically to every generated document.*

## 3.3 Update rule

The *Registry* is updated by hand, except for the fields the site generator writes itself.

- **Stylesheet version changed:** update the *pair*'s version and *timestamp* in the stylesheets section.
- **Document regenerated:** update the *timestamp* of the slug concerned. A translated document has one entry per variant.
- **Rendering flag adjusted:** update the rendering section. Renderers read this flag at *generation* time — a simple switch needs no document to be regenerated.
- **Compilation notice:** update compiledWith. An editorial decision, with no effect on the documents produced.
- **lastDeploy:** updated automatically by the site generator after each ZIP is generated. Never edit by hand.
- **Content dates:** contentTs and contentHash, in the documents section, are written by the site generator. At every pass it takes the *fingerprint* of the document's plain text — headings, paragraphs, list items, cells, notes, tips, code and *prompt* blocks, image captions — without indentation, colour, italics, version *fingerprint* or *timestamp*. Same *fingerprint*: nothing moves. Different or absent: contentTs takes the document's *timestamp*. The history page is dated on contentTs, so a *stylesheet* bump, which regenerates the corpus without changing its text, adds no line to it. Never edit by hand. The write preserves the entry's other keys — the PDF flag, for one — and an entry whose value carries a brace or a bracket is reported by name instead of being rewritten.
- **Icons of the sub-site:** the sub-site's four icons — favicon.svg, favicon.ico, apple-touch-icon.png and index-icon.svg — live at the root of the project, and the pass copies them into the site's resources. The parent site draws them, the project carries them: a sub-site taken over from its archive alone keeps its identity. The pass stops, naming the file that is missing; index-icon.svg is expected only if rendering.indexIcon declares it. deploy.siteInfrastructure now covers robots.txt and .htaccess only. The drawing in index-icon.svg centres itself in its display area, vertically and horizontally, and fills it: the *stylesheet*

centres the 64 by 64 box, never the content of the file, and an off-centre drawing shows exactly as it is on the page.

- **Project archive:** when projectShareable is true, the pass produces the archive of the whole tree and places it under assets/downloads/, under the name declared in projectZip.filename. The publication ZIP carries it, so that the *landing page*'s link and projectZip.downloadUrl resolve the announced version, with no separate deposit. The site folder and the archives are excluded from what is packaged.

**Note:** *The Registry ships with the Kit and is put back into the working tree at every update. No timestamp in the file name — it is the source of timestamps.*

## 4 Delta deployment

The site generator produces a full ZIP or a delta ZIP depending on the state of lastDeploy. A full deployment includes every file. A *delta deployment* includes only the files changed since the last deployment.

### 4.1 Delta logic

The comparison uses the document's local *timestamp*, read from the *Registry*, converted to ISO 8601 UTC and then compared with lastDeploy. A document with a later *timestamp* triggers the regeneration of its page, its PDF and its images. The others are left out of the delta ZIP.

```
// lastDeploy null = passe complète
const lastDeployIso = (mode === 'full')
  ? null
  : (registry.deploy && registry.deploy.lastDeploy);

function kitTsToIsoDateTime(ts) {
  // 'AAAA-MM-JJ - HHhMM' en heure locale Luxembourg vers ISO 8601 UTC.
  // L'offset local (CET +1h, CEST +2h) est recupere via Intl.DateTimeFormat
  // et soustrait pour obtenir l'UTC reel. Sans cette conversion, toutes
  // les comparaisons seraient decalees de une ou deux heures.
  const m = /^(\d{4})-(\d{2})-(\d{2})\s*-\s*(\d{2})h(\d{2})$/.exec(ts || '');
  if (!m) return null;
  const [, y, mo, d, h, mi] = m;
  const naive = new Date(Date.UTC(+y, +mo - 1, +d, +h, +mi, 0));
  const dtf = new Intl.DateTimeFormat('en-US', {
    timeZone: 'Europe/Luxembourg',
    timeZoneName: 'longOffset',
  });
  const tz = dtf.formatToParts(naive).find(p => p.type === 'timeZoneName');
  const m2 = /GMT([+-])(\d{2}):(\d{2})/.exec(tz ? tz.value : '');
  if (!m2) return naive.toISOString();
  const offsetMin = (m2[1] === '+' ? 1 : -1) * (+m2[2] * 60 + +m2[3]);
  return new Date(naive.getTime() - offsetMin * 60 * 1000).toISOString();
}

function isNewerThanLastDeploy(ts, lastDeployIso) {
  if (!lastDeployIso) return true;          // passe complete
  const iso = kitTsToIsoDateTime(ts);
  if (!iso) return true;                       // prudence : inclure si TS
```

```
incompatible
  return iso > lastDeployIso;
}

// Dans la boucle des documents :
const meta = registry.documents && registry.documents[slug];
if (!meta || !meta.ts) continue;
if (mode !== 'full' && !isNewerThanLastDeploy(meta.ts, lastDeployIso)) {
  console.log('- ' + slug + ' (inchange, exclu du delta)');
  continue;
}
```

**Note:** *The delta logic long relied on the file's modification date on disk. In an environment where the file is re-imported at every session, that date is rewritten and does not show the actual generation date: the delta ended up including almost every file. The comparison now uses the Registry timestamp, the canonical source in local time, converted to UTC.*

### 4.2 Special case — the stylesheet

The site's CSS file is regenerated if the HTML *stylesheet* has changed version since the last deployment, by comparing its *timestamp* in the *Registry*. Since a *consumer project*'s *Registry* carries no stylesheets section, the CSS is regenerated there on a full pass only.

### 4.3 ZIP naming convention

Pattern: {domaine} (AAAA-MM-JJ - HHhMM).zip. The ZIP of the Kit's sub-site therefore carries the Kit's domain. *Consumer projects* replace the domain with that of their sub-site.

### 4.4 Automatic update of lastDeploy

After a ZIP is generated successfully, the generator writes the ISO *timestamp* of the current moment into lastDeploy. This operation rewrites the *Registry* on disk.

**Note:** *Never edit lastDeploy by hand. To force a full deployment, set the value to null and regenerate: the generator detects null and switches to a full pass.*

## 5 Site structure

All HTML files for document pages sit in a dedicated subfolder. Only the *landing page* stays at the root of the sub-site. This separation keeps the hierarchy readable and the relative paths uniform from every document.

### 5.1 File hierarchy

| Path | Content |
|---|---|
| **[sous-site]/** | Root of the sub-site |

| Path | Content |
|---|---|
| **index.html, index-{langue}.html** | Loaders for the index pages. The only files the Kit writes at the root; they carry no content. |
| **html/** | Folder for every document page and the per-language index pages |
| **html/[slug].html** | Document page — one file per published document |
| **html/[slug]-[lang].html** | Language variant of a translated document — Naming Convention §2.4 |
| **html/glossaire.html** | HTML glossary — generated by the Kit helper from the terms module, never via the *AST* |
| **assets/** | Shared resources |
| **assets/kit-style.css** | HTML *style sheet*, generated from the *stylesheet* |
| **assets/favicon.svg** | Main icon. Carried by the project — Kit - Projet - *Prompt* §16 |
| **assets/favicon.ico** | Multi-size fallback. Carried by the project — Kit - Projet - *Prompt* §16 |
| **assets/apple-touch-icon.png** | *iOS* home-screen icon. Carried by the project — Kit - Projet - *Prompt* §16 |
| **assets/index-icon.svg** | Illustration of the *landing page*. Carried by the project — Kit - Projet - *Prompt* §16 |
| **assets/downloads/** | Project archive offered for download from the *landing page*. Folder present when projectShareable is true in the *Registry*: the pass produces the archive there, under the name declared in projectZip.filename, and the publication ZIP carries it. |
| **assets/pieces/** | Appendix pieces deposited by the project, under their original name — never the slug, never a produced file. HTML *Pipeline* §6.3 |
| **assets/pdf/** | PDFs generated from the .docx — one per enabled document |
| **assets/png/{slug}/** | Images extracted by *pandoc* — one folder per document |

| Path | Content |
|---|---|
| **robots.txt** | Crawling directives. Deposited by the parent site — Kit - Projet - *Prompt* §16 |
| **.htaccess** | *Apache* configuration of the sub-site. Deposited by the parent site — Kit - Projet - *Prompt* §16 |

## 5.2 Relative paths

All HTML files in the subfolder use relative paths that go up one level: the *landing page*, the *style sheet*, the images and the PDFs are all reached through the parent folder.

**Note:** *Never use a path relative to the current folder to reach the landing page from a document page: it would point to a file that does not exist in the subfolder. Always go up one level.*

## 5.3 Structure of the deployment ZIP

The ZIP mirrors the hierarchy above exactly. Extracting it on the server rebuilds the expected structure with no manual handling.

```
kit.sliver.lu (AAAA-MM-JJ - HHhMM).zip
├── index.html
├── index-fr.html
├── index-de.html
├── html/
│   ├── kit-index-fr.html
│   ├── kit-index-de.html
│   ├── reading-guide.html
│   ├── manuel.html
│   ├── glossaire.html
│   └── ...
└── assets/
    ├── kit-style.css
    ├── pdf/
    │   └── reading-guide.pdf
    └── png/
        └── {slug}/
            └── {slug}
```

A site publishes one index page per language, in html/, named {préfixe}-index-{langue}.html — the prefix comes from deploy.siteName. The base language carries its suffix like the others: none has two possible addresses.

**Note:** *The root belongs to the project. The Kit writes only loaders there: one per published language, index-fr.html and its peers, plus index.html, which serves the language declared in the Registry under project.docLanguage — never the browser's, otherwise a shared link would not lead to the same place for every recipient. These are the names expected by the parent site's.htaccess, whose language block tests and serves absolute paths from the root: that is what keeps it identical from one sub-site to the next. A loader carries no content and uses location.replace: an href or a refresh tag writes a history*

*entry, and a reader going back from the index page would be sent straight back to it.*

**Note:** *The generator refuses to overwrite a root file it did not produce, and stops, naming it. A project whose output is a website puts its own home page there: it cannot disappear without a message. Files written by the Kit carry a marker at the top that distinguishes them from the project's.*

- **Localised content.** Section titles, document labels, subtitle, welcome text and index page title follow the language of the page. The labels come from the titles table — Common Structure §6.8; the section titles are declared in the site generator.
- **Links to the right variant.** The German index page points to the German pages. An untranslated document appears there under its German label and points to the existing variant.
- **Language selector.** Each index page links to the others through the language selector, placed according to rendering.languageSelector — Common Structure §15. The history page has one name per language: historique, verlauf, history.

# 6 PDF generation

A PDF can be generated from a Word document directly in the session with *LibreOffice* on the command line, which is available in the environment. If the binary is provided in the *chat*, the PDF can be produced without any action on the user's side.

## 6.1 Reference command

```
libreoffice --headless --convert-to pdf document.docx --outdir ./

# Exemple avec chemin complet :
libreoffice --headless \
  --convert-to pdf \
  'Kit - Documentation - Pipeline HTML (AAAA-MM-JJ - HHhMM).docx' \
  --outdir /mnt/user-data/outputs/
```

## 6.2 Integration into the site generator

If PDF is enabled for a document in the *Registry*, the generator produces it with *LibreOffice* and includes it in the ZIP under the PDF folder.

**Note:** *LibreOffice is available in the session. Never ask the user to generate the PDFs themselves once the document's binary is provided in the chat. This capability is always available, unconditionally.*

## 6.3 Annexes and manuals — external PDF

Documents in the Annexe and Manuel categories of a *consumer project* carry a source PDF that is never produced by *LibreOffice*. The user deposits the PDF by hand in the *working tree* with the same base name as the paired

document. The generator copies it as is — no conversion is called for these documents.

Base-name matching rule: for each document in these categories, the generator looks for a PDF whose name is identical to the document's minus its *timestamp* segment and extension. Exact match, no tolerance.

- **File pair:** the document carries a *timestamp*, the PDF never does — it is a frozen source file, deposited as is.
- **HTML slug:** prefixed with the category to avoid any collision if two documents with the same name exist in two different categories. The matching PDF follows the same rule.
- **Pipeline behaviour:** direct copy of the PDF to the PDF folder; the document follows the normal chain of HTML *generation* and validation.
- **Loud failure when missing:** if the expected PDF is missing from the *working tree* at *generation* time, the generator stops and names the missing file. No silent fallback to a generated PDF, no page without an icon.
- **Landing page sections:** the permanent Annexes and Manuals sections appear at the end of the sub-site's *landing page*, after the numbered sections. Each is hidden if it holds no document. Alphabetical order within each.
- **Genre of the coupled document:** the document accompanying the piece is a document of record — *Kit Prompt* §12. It reports on the piece without copying it, names what is wrong, and closes on the section of points found. §6.3 settles the mechanics; the genre of the text is read in the *Prompt*.
- **The sheet is published, the button serves the piece:** the appendix sheet is published like any other page, and its download button returns the original piece, never the PDF drawn from it. The piece is declared in the *Registry*, through the piece key of the document's entry, and lives at the root of the project; the pass copies it under assets/pieces/, under its original name and never under the slug, and stops, naming it, if it is missing. The folder separates what is deposited from what is produced: a piece and a sheet's PDF cannot collide.
- **The PDF of the sheet is still produced:** it enters the site ZIP and the project archive, for the *binder*, and no page links to it: the reader is thus never made to choose between two PDFs of which only one holds. The button changes neither icon nor label from page to page — the note at the head of the sheet says what it returns.

**Note:** *The PDF flag applies to an appendix sheet as to any other document: its PDF is produced from its .docx and stays in assets/pdf, with no page linking to it. The piece is served*

*separately, from assets/pieces.*

Reference pattern for the site generator: detection by category, matching by base name, copy of the source PDF.

```
// gen-{prefix}-site.js — extrait pour une fiche d'annexe
const fs   = require('fs');
const path = require('path');

// La piece se declare au Registry, par la cle piece de l'entree du
// document ; elle vit a la racine du projet. Aucun appariement par nom
// de base : le Registry dit le fichier, et lui seul.
function pieceDeclaree(slug, registry, projectDir) {
  const meta = (registry.documents || {})[slug] || {};
  if (!meta.piece) return null;
  const src = path.join(projectDir, meta.piece);
  if (!fs.existsSync(src)) {
    throw new Error('Piece d\'annexe manquante : ' + meta.piece);
  }
  return src;
}

// Dans la boucle de generation de chaque document :
const src = pieceDeclaree(slug, registry, PROJECT_DIR);
if (src) {
  fs.mkdirSync(path.join(OUTPUT_DIR, 'assets/pieces'), { recursive: true });
  fs.copyFileSync(src,         path.join(OUTPUT_DIR,         'assets/pieces',
path.basename(src)));
  // pdfHref de renderDocument pointe la piece, jamais le PDF de la fiche
}
// Le .docx suit la chaine ordinaire : page HTML, et PDF si le drapeau
// du document l'autorise. Ce PDF reste dans assets/pdf, sans lien.
```

# 7 Robustness rules

This section documents the *pipeline*'s most dangerous traps and the mandatory rules that follow from them. Each rule comes from a real error that caused lost content or corrupted rendering.

## 7.1 The doc.paragraphs trap — silent loss of tables

If an update script reads an existing document with a *Python* library and iterates over its paragraphs to rebuild it, every table disappears silently. The document produced is syntactically valid but stripped of its tabular content.

Measured on a real document: 300 paragraphs in the XML, of which only 84 were visible to the *iteration* — 28 per cent. The other 216 lived in the tables, and all 18 tables were lost without the slightest warning.

**Note:** *Never update an existing document by iterating over its paragraphs. Always rewrite the generator script from scratch from the source file. The binary is provided in the chat to rewrite the script, never to edit the XML directly.*

## 7.2 Escaping on paths

The HTML *stylesheet*'s escaping function protects special characters when rendering running text. It must never be applied to a path attribute.

- **Forbidden:** escaping an image source, a link target or a file path — it corrupts the path.
- **Allowed:** escaping cell text or paragraph content — running text only.

**Note:** *Escaping an image path produces a corrupted attribute: the image becomes unreachable with no explicit error message in the HTML.*

## 7.3 Prior reading specific to the HTML pipeline

The general rule requiring the References to be read before any *generation* is set out in *Prompt* §4.1 and §5.1. It applies here without exception. The following readings are specifically required before writing the first line of an *AST* converter or a site generator.

- **HTML Reference:** in full, in particular the function contracts and the *AST* conversion *pipeline*.
- **General Reference:** §1.2 for the basic structure of a document and §13 for the return contracts — the converter produces HTML but reasons on structures taken from the Word *model* (.docx).

Strict stop rule: if any of these documents has not been read in the current session, full stop. Do not ask a question, do not assume, do not continue.

## 7.4 Validating the source document before conversion

A document enters the HTML *pipeline* only after passing the validation chain. The *validator*'s detailed contract — list of checks, usage, return codes — is in *Quality Control* §3.1. The conditions below are specific to the *pipeline* and checked on entry. The *fingerprint* and the version do not ask the same question: the *fingerprint* tells whether the document was produced by the Kit, the floor whether it still renders faithfully.

| Condition | Expected | Consequence if absent |
|---|---|---|
| ***Fingerprint*** **present** | StylesheetVersion and GeneratedAt in the document properties | Absent: the document is rejected at conversion |
| **Version up to date** | StylesheetVersion greater than or equal to minDocumentVersion, the floor declared in the *Registry* | Lower: the document is obsolete and blocked |
| ***Validator*** **passed** | kit_validate_docx.js exit 0 on the source .docx | A document that has not been validated must never enter the *pipeline* |

**Note:** *The site generator refuses any document with no fingerprint or with a version below the*

*floor. A published document has therefore passed through a formatting engine whose rendering is still held to be faithful. Bumping the stylesheet no longer makes anything obsolete: the document set can carry several versions at once, uniform in rendering without being uniform in number. Moving the floor is the deliberate act that forces a general regeneration, and no check catches that judgement.*

## 7.5 The level of a heading comes from the document

The conversion returns a heading block carrying its level, and the generator reads it as it stands. This rests on the style inheritance chain, restored in General v1.94 by the declaration of the Normal style: without it, a converter that resolves inheritance recognised no heading, and everything that followed stayed at the level of the previous one.

The text of a heading decides nothing. A paragraph starting with a number stays a paragraph; a heading whose text starts with a digit stays a heading, which *Kit Prompt* §12 forbids in any case for readability.

**Note:** *A document regenerated with a stylesheet older than General v1.94 does not declare the Normal style: its headings fall back to paragraphs at conversion, and the level of everything that follows falls with them. The corpus is therefore regenerated before its next publication.*

## 7.6 Paragraph with a bold lead — the separator stays

When the *AST* converter meets a paragraph starting with bold text, it detects a bold lead and calls the matching helper with the bold text and the rest of the sentence.

The rest goes through as it stands. The separator that follows the lead — colon, em dash, plain hyphen — belongs to the document: it reads on paper and must read on the page. The HTML *stylesheet* sets none of its own, so nothing is doubled.

Rule: nothing is stripped from the start of the rest, neither separator nor space. A strip costs a character that the text-conservation guard then requires, and publication stops on a loss coming from the generator and not from the document. This rule applies to any *AST* converter written from scratch.

## 7.7 Family and external hyperlinks

The HTML generator must distinguish links to the family's domains and their subdomains, which stay captured in the mobile app's *WebView*, from external links, which must open in the system browser. This distinction is handled by an internal helper of the HTML *stylesheet*, which adds the external-opening attributes to the latter only.

Configuration: the list of family domains is declared in the *Registry*, and every site generator calls the matching *setter* at the top. If the *setter* is never called, every absolute link is treated as external.

Matching mechanics. Relative URLs and anchors are family by definition. Non-http schemes — email, telephone — are family by definition, since the system handles them. For absolute URLs the host name is extracted and compared with the list: exact match or subdomain. Cross-links between family sites therefore stay in the *WebView*.

A CSS class is set on external links as a hook, deliberately declared empty. It lets a project style external links differently without touching the *stylesheet*. The helper is called when URLs are detected in running text and for table cells carrying a link.

**Note:** *Dependency on the mobile app. The external-opening attributes only take effect if the app implements the matching routing in its WebView component. Otherwise they are silently ignored and external links go dead.*

## 7.8 Single-column blocks at conversion

A code or dialogue block may come out of the conversion with its first row in the header of its table. The generator therefore reads every row for these blocks, and takes the first of that set for the note and the tip.

The header and body split now holds for the ordinary table alone. Read from the body alone, the first row of a block was lost, and the conservation guard stopped the pass, naming a fragment it could not find.

**Note:** *Dependency on the mobile app. The external-opening attributes only take effect if the app implements the matching routing in its WebView component. Otherwise they are silently ignored and external links go dead.*

## 7.9 Mandatory setters at the top of the site generator

Every HTML site generator must call a series of *setters* at the top, before building the first element. A missing required *setter* silently disables the matching *pipeline*: the generator runs without error but produces a semantically truncated site.

| # | Setter | Source | Effect if absent |
|---|---|---|---|
| **1** | setLanguage | Registry.project.docLanguage | Default EN — every localised string in English |
| **2** | setFamilyDomains | Registry.familyDomains | Every absolute link treated as external |
| **3** | setGlossaryTerms | project's terms module | No glossary link on any page — semantically truncated site |
| **4** | setBrands | project's brands module | No brand rendered in small caps |

| # | Setter | Source | Effect if absent |
|---|---|---|---|
| **5** | setVariableNames | project's variables module | No variable name rendered in green italics |
| **6** | setHassEntities | project's entities module | No entity rendered in purple italics |
| **7** | setGlossaryHref | glossary slug of the rendered language | Every term link targets the same page — a non-French-speaking reader gets the definitions in the project's language |
| **8** | setDocumentTitles | [Préfixe] - Document Titles (TS).js — titleEntries and categoryLabels | Labels frozen in the project language, no document reference turned into a hyperlink, *cover page* subtitles not composed |
| **9** | setLanguageSelector | Registry.rendering.languageSelector | Default “document”: icons at document level instead of the bar menu |
| **10** | setTextHighlights | project's Text Highlights module | No declared fragment highlighted |

Data *setters* depend on the *Registry* flags — a project with no glossary does not call the matching *setter*, since the module does not exist. setLanguage and setFamilyDomains are universal and always required. setGlossaryHref concerns only projects publishing their glossary in several languages: its default value suits a monolingual project, and it is called once per rendered document, since the target changes with the document's language.

The generator also passes the *stylesheet* the compilation notice read from the *Registry* and, for each translated document, the list of its published languages. This list is declared in the generator, never in the *Registry*: the translation scope is an editorial decision per document, not a stable project flag.

```
// Patron canonique en tete de gen-{prefix}-site.js — ordre recommande
const registry = require('./[Prefixe] - Registry.js');
const style    = require('./Kit - Stylesheet - HTML - Code (TS).js');

// 1. Langue active — depuis Registry
style.setLanguage(registry.project.docLanguage || 'EN');

// 2. Domaines famille — toujours appele, meme si la liste est vide
```

```
style.setFamilyDomains(registry.familyDomains || []);

// 3 a 6. Setters conditionnels selon Registry.requires
if (registry.requires && registry.requires.glossary) {
  const { glossarySearchTerms } = require('./[Prefixe] - Glossary -
Terms.js');
  style.setGlossaryTerms(glossarySearchTerms);
}
if (registry.requires && registry.requires.brands) {
  const { brandEntries } = require('./[Prefixe] - Brands.js');
  style.setBrands(brandEntries);
}
// idem variableNames et hassEntities
```

**Note:** *The linter kit_check_setters.py statically checks that every required setter is actually called. It runs before each site generation. Since a missing setter is silent at run time, this linter is the only automatic safety net — see Quality Control §3.4.*