Kit de documentation

Project — Update Guide — EN

1 Why the Kit is injected in full

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

This guide is for you, not for the *AI*. It describes how to reinject an updated version of the Kit into an existing project without breaking work in progress.

The Kit evolves as a whole. A new version does not merely bring a corrected *stylesheet* or a reworded document: it carries a full chain of dependencies. A *stylesheet* is always delivered with its associated Reference, at the same *timestamp*. The *Registry* declares the new active version. The technical *prompt* may hold new rules that rely on the new functions.

Absorbing an update piecemeal — replacing a *stylesheet* without its Reference, forgetting the *Registry*, keeping an old *prompt* that does not know the new rules — produces silent inconsistencies. The *AI* then reads documents that contradict each other, applies rules that no longer match the available functions, and the structure of produced documents degrades without warning.

Simple rule: a Kit update is obtained as an archive in a single gesture, from the address declared in the *Registry* in projectZip.downloadUrl. Every file in the batch replaces its *predecessor* in one go. No picking and choosing, no partial replacement.

This rule is deliberately wider than strictly necessary. Deciding file by file what is useful assumes knowledge of the dependencies — precisely what should not be asked of you. A few extra explanatory documents in the *context* cost nothing; a missing file produces an inconsistency that will only show at the next *generation*.

💡 *An update replaces the Kit's files and leaves yours in place. Your work in progress is preserved as long as you have not modified a Kit file yourself.*

2 Rule of non-compatibility

The Kit offers no compatibility, forward or backward. Every new delivery starts from the active *stylesheet* and its Reference, script from scratch. Scripts from previous sessions are neither consulted nor adapted — they do not exist for the current session.

That is also why Kit documents carry no version number in their text: two versions are never in circulation. A document describes the present state, and the history lives in the changelog blocks at the head of the code files.

3 Files concerned

Two kinds of file coexist in your project: the Kit files, replaced in full at every update, and your project's own files, never touched.

- **Kit files — everything, together:** a Kit update is injected in full, as delivered. The complete list is in Initialisation Guide §3 — it does not differ between initialisation and update. No file-by-file examination is needed.
- **Your project's files — untouched:** no file carrying your prefix is touched by a Kit update. They stay exactly as you left them. The one exception: at step 4

below, if a structural change introduced by the update calls for a migration — and only on your explicit confirmation, one migration at a time.

4 Procedure

The update takes five steps. For routine updates — fixes, minor new features — replacing the files does not call for opening a session straight away: version detection happens automatically at the start of the next session.

Exception: if the update brings a major structural change — new data-module contract, new *Registry* flag, reworked export — a validation session is mandatory before resuming documentation work. The §4.4 procedure then applies.

4.1 Step 1 — Fetch the new files

Fetch the updated Kit files from your reference source. Keep the old files on your machine until the procedure is over.

💡 *Always keep the previous version of the stylesheet until the new version has been fully validated in a session.*

4.2 Step 2 — Check the versions

Before replacing, check that the *Registry* is part of the downloaded batch and that the declared version numbers match the actual versions of the new files.

Check	Where to find it
Version of the new *stylesheet* file	First lines of the .js file — the “Version : x.xx” comment.
Version declared in the *Registry*	Kit - *Registry.js*, stylesheets section, version field of each *stylesheet*.
Timestamp* of the *pair	Kit - *Registry.js*, stylesheets section, ts field. Must be identical for the .js and the Reference .docx.

A *stylesheet* file carries its version number in two places: the header comment, readable at a glance, and the exported constant that is actually written into each produced document. A gap between the two is invisible at runtime but falsifies any manual check. The matching audit is described in *Quality Control* §4.2.

💡 *The Registry is the only source of truth for the active versions. A version is never held from memory: it is reread in the Registry at the start of each session.*

4.3 Step 3 — Replace in the project

In your project, delete the old Kit files and upload the new ones, *Registry* included. Order does not matter for this step.

At the start of the next session the *Registry* is read and any gap between declared versions and the actual versions present in the *context* is detected automatically. If the *Registry* was not uploaded together with the stylesheets,

the inconsistency is reported and its deposit requested before any *generation*.

4.4 Step 4 — Adapt the project files

A Kit update may bring changes that affect the structure of the *project files*: a new mandatory field in a *data module*, a new flag, a reworked export. In the first working session after the update, the procedure below applies without exception before any other task.

- **Gap diagnosis:** the structure of the existing *project files* is compared with the normative structure of Common Structure. Each gap is reported as “Gap detected in [file]: [description]. Migration proposed: [action]”, and waits for your explicit confirmation. No silent modification.
- **Data modules:** the module-specific check — mandatory fields, export conformity, multilingual format — follows the procedure in Common Structure §6.7.
- **Missing optional files:** for each optional file described in Common Structure and absent from your project, creation is offered explicitly. Never a silent creation.
- **Existing optional files:** an optional file already present is never deleted, even if its flag is false. Deleting an existing project file happens only on your explicit *instruction*.

Note: *No project file is ever deleted autonomously, for any reason — flag switched off, file judged obsolete, or anything else. Deletion is always your decision, stated in the chat.*

4.5 Step 5 — Check the rendering floor

Most updates do not affect the appearance of documents already produced. Some do: a changed spacing, a changed width, a replaced colour. Your old documents then differ from your recent ones, and the *binder* becomes a mix of two layouts.

The Kit signals these cases with a floor, minDocumentVersion, declared in Kit - *Registry.js* — HTML *Pipeline* §3.1. It carries the last version that changed the rendering. Each produced document remembers the version that produced it.

- **The floor has not moved:** nothing to do. Your documents stay valid, whatever version produced them.
- **The floor has risen:** every document produced by an earlier version is regenerated. The content does not change — it is extracted from the existing binary and rendered by the active version. The site generator refuses to publish a document below the floor, and says so with the file name.

Note: *A regeneration campaign is checked document by document: kit_check_markup.py compares the markup between the source file and the regenerated document. Across a*

whole corpus, that check has already caught a perfectly valid document whose content was another's.

5 Absorption session

After replacing the Kit files in the *working tree*, open a new *conversation* in that project and paste the text below as the first message.

That text triggers the orderly absorption of the update: rereading every source of truth, detecting version gaps between the *Registry* and the actual files, applying the §4.4 procedure to your existing *project files*, and proposing the necessary migrations one at a time. None of your files is modified without your confirmation.

```
ABSORBING A KIT UPDATE
==============================

CONTEXT :
------------------------------

The updated Kit files have been uploaded into the project
context, Registry included. No project-owned file may be
modified without explicit confirmation.

MANDATORY READING — STRICT ORDER :
------------------------------

Before any other action, read in this order :

1. Kit - Registry.js
   — source of truth for the active versions.
2. Kit - Projet - Prompt (TS).docx
   — permanent technical instructions ; may hold new rules.
3. Kit - Projet - Guide de mise à jour (TS).docx
   — this guide, in particular §4.4 adaptation procedure.
4. Kit - Projet - Structure commune (TS).docx
   — normative contract of the project files, reference for
     the gap diagnosis.
5. Kit - Stylesheet - General - Code.js + Reference.docx
   — main pair.
6. The other stylesheet pairs (Glossary, YAML, HTML)
   — Code.js + Reference.docx each time.

VERSION DETECTION — MANDATORY REPORT :
------------------------------

For each stylesheet, compare :
- version declared in Kit - Registry.js
- version read at the head of the .js file
- version constant exported by the .js file
- timestamp of the pair (.js and Reference .docx)

Report expected for each stylesheet :
- Registry version    : x.xx
- .js header version  : x.xx
- exported constant   : x.xx
- pair timestamp      : YYYY-MM-DD - HHhMM
```

```
- consistent : yes / no

GAP DIAGNOSIS ON THE PROJECT FILES :
-----------------------------

After the full Kit reading, compare the structure of the
existing project files with the normative structure of
Kit - Projet - Structure commune. For each gap, report as :

    « Gap detected in [file] : [description].
      Migration proposed : [action]. »

List every gap before applying any migration. Change nothing
until the full report has been delivered.

STOP CONDITIONS :
-----------------------------

If any of the following is met, full stop and explicit
report :

- Kit - Registry.js missing or out of date
- a Kit .js file present without its Reference .docx,
  or the reverse, or with mismatched timestamps
- Registry version and .js file version diverge
- an expected Kit file missing from the working tree
- an existing project file contradicts a Structure commune
  contract in an ambiguous way

AFTER THE FULL REPORT — WAIT FOR THE GO :
-----------------------------

No modification of an existing project file happens without
explicit confirmation, one migration at a time. No project
file is deleted autonomously, for any reason.

For missing optional files, ask explicitly whether they
should be created. Never a silent creation.
```

After the reading report and the gap diagnosis, answer “go” for each migration you approve, or “skip” for those you want to postpone. Only approved migrations are applied.

6 Special case — changing language

If you want to change your project's documentation language, only one action is needed: change the language field in your *Registry*, which is its single source of truth, and upload the updated file.

Every document produced from that session on will use the new language. Existing documents are not modified. Renaming your files to reflect the categories and subjects of the new language is described in Naming Convention §5.4.

Not to be confused with publishing a document in several languages, which is a per-document decision and does not affect the project language — Naming Convention §2.4.

7 What never changes

Some parts of your project are entirely insulated from Kit updates.

Note: *“Kit-X - Registry.js” and “Kit - Registry.js” look alike in a file list, and only the second is replaced. Reread what you are about to overwrite before confirming: too wide a selection takes your extension along with the Kit. Backing up your Kit-X files before each update costs a minute.*

Item	Impact of a Kit update
Your *project files*	None — never touched, modified only on your explicit *instruction*.
Your existing generator scripts	Not reusable — the script is rewritten from scratch each session.
Your content documents already produced	None — they are static files.
Your glossary and *data modules*	None — *project files*, their presence depends on your flags.
Your project-specific conventions	None — they live in the dedicated section of your project *prompt*.
Your Kit-X files	None — the Kit-X prefix is yours. These are your extensions of the Kit: additional stylesheets, colour palette, tools. They apply to all your projects and follow you from one to the next.