

A Unified Authoring Stack for Jupyter

Description	<i>A Unified Authoring Stack for Jupyter</i>
Tracking issue	https://github.com/jupyter-governance/funding-proposals/issues/83
Proposal owner	QuantStack
Goal(s)	<i>This proposal strengthens plain-text authoring across the Jupyter ecosystem through three complementary efforts:</i> - <i>the standardisation of a strict MyST subset as the official Jupyter markdown format in .ipynb documents,</i> - <i>a JEP-driven specification for text-based notebooks, and a reference implementation - potentially included in the “plainb” project,</i> - <i>a live splitview editor for JupyterLab with synchronized scrolling between source and rendered preview.</i>
Funding	\$80 000
Timeline	<i>One calendar year from the approval of the proposal</i>
Status	Writing ▾

Description of activities

One paragraph explaining the proposal and describing the major activities this funding would cover.

This proposal brings together three complementary efforts to strengthen plain-text authoring across the Jupyter ecosystem:

- the standardization of a strict MyST subset as the official markdown format for Jupyter,

- a specification-driven approach to text-based notebooks, shepherding a long-stalled JEP to consensus with a reference implementation,
- a live split-view editor for JupyterLab that unifies source editing and rendered preview.

Each is independently valuable, but together they form a coherent program to make Jupyter a first-class environment for writing and maintaining the narratives that surround scientific code.

1. The markdown flavor in Jupyter notebooks

The Jupyter notebook format specification requires that a markdown cell holds a string of "Markdown," but it does not specify which *flavor* of Markdown. In practice, every Jupyter interface could ship its own, so the same notebook renders differently across core Jupyter and third-party viewers.

This imprecision is now a liability: it fragments the ecosystem, blocks scientific authoring features that depend on a defined format (cross-references, numbered equations, citations), and forces each tool to re-derive what "Jupyter Markdown" means.

The Jupyter community has, over several years, developed its own markdown dialect, MyST, purpose-built for technical and scientific documents, with first-class support for cross-references, numbered and labeled equations, citations, footnotes, admonitions, and rich tables. A mature TypeScript renderer (MyST-JS, [mystmd](#)) already powers [jupyterlab-myst](#), an extension that renders MyST natively inside JupyterLab cells and document previews. MyST is the substrate of JupyterBook 2 and interoperates with Sphinx. Despite this maturity, MyST has not been adopted as default in JupyterLab, and the notebook format has not committed to any flavor.

This project does two things. First, it adopts the **MyST** TypeScript renderer as the default markdown renderer in core JupyterLab, replacing the current informal "GitHub-Flavored Markdown mode of marked plus ad-hoc extensions" configuration (based on the [marked renderer](#)). Second, it shepherds a Jupyter Enhancement Proposal (JEP) that defines a strict, enumerated subset of MyST as the official format for Jupyter markdown cells — explicitly not the full, open-ended MyST extensibility model.

The reason for limiting scope to a strict subset is deliberate and principled: MyST is highly extensible through directives, which in principle permits features such as executable code embedded in markdown cells. Standardization documents are long-lived; features that enter the format are extraordinarily difficult to remove once notebooks depend on them. We therefore propose to standardize a closed, conservative set of capabilities — the core scientific-authoring features Jupyter users already need — and to leave further additions to future, separate JEPs, each justified on its own merits.

Adopting MyST as the standard markdown format for Jupyter resolves the long-standing imprecision, brings core scientific-authoring features (cross-references, numbered equations, citations) to every Jupyter interface, unifies rendering across the ecosystem, and aligns Jupyter with the broader open-publishing toolchain — all while keeping the standardized surface small, stable, and future-proof.

2. Text-based notebooks

The `.ipynb` JSON format is Jupyter's canonical notebook serialization, but it is poorly suited to the workflows scientists and engineers actually use: it is unfriendly to version control, hard to read and edit as text, and inaccessible to standard text-processing tools. For years the community has worked around this with a proliferation of text representations — Jupyter's percent and Markdown formats, MyST Markdown Notebooks, Sphinx-Gallery, MyST-NB, Quarto, and others — each with its own parser, its own conventions, and no shared specification.

In 2023, the community gathered for a Jupyter community workshop dedicated to the notebook file format. Out of this workshop came a family of Jupyter Enhancement Proposals aimed at giving notebooks a first-class text representation: JEP #102 ("official support for Markdown-based notebooks", which was itself based on Myst) opened the conversation, and its companions — JEP #103, which drafted a concrete serialization, and #138 — attracted significant traction and numerous contributors. What they did not attract was funding: no resources were allocated to carry the proposals through the full JEP process, harden a reference implementation, or coordinate adoption. The effort lost momentum, and the proposals remained open years later.

By 2026, the picture had changed. `plainb` and its JupyterLab extension `jupyter-plainb` — developed by QuantStack — emerged as a working alternative for editing notebooks as plain text, letting authors open and edit `.py/.md` files directly in the JupyterLab notebook editor with saves back to the original text format, and shipping parsers for Jupyter percent, Sphinx-Gallery, classic Markdown, and MyST-NB. Meanwhile, the 2026 Paris-Saclay workshop ("Demystifying MyST Markdown in Education," June 2026) revisited the text-representation discussion.

This project picks up where the 2023 proposals stalled. It shepherds a JEP focused on nbformat text specifications — building on JEP #102 and its companions — and matures `plainb/jupyter-plainb` into the conformant reference parser, in coordination with Jupyter. Because QuantStack is both the author of `plainb` and a core contributor to JupyterLab, the reference implementation and the standardization are developed by the same team that will land and maintain them upstream.

Note on plainb: plainb is a JupyterLab extension that converts plain-text notebook files to Jupyter nbformat 4 JSON. It currently supports the following formats:

- `.py` : Spyder / Jupyter percent format - cells delimited with `# %%`
- `.md` classic : Markdown file where fenced code blocks with a language tag become code cells
- `.md` MyST: MyST Notebook format — `{code-cell}` directives and `+++` cell breaks
- Sphinx Gallery: Sphinx-Gallery `.py` scripts — RST docstring + `# %%` sections

It differs from Jupyter in that it does not make any change to the Jupyter server content API and merely provides the ability to edit these file formats in place with the notebook editor.

3. A live split-view editor for JupyterLab

JupyterLab is the leading interactive computing environment for data science, scientific research, and education, with millions of users worldwide. It already ships a capable text editor and a separate Markdown renderer, but it lacks a consolidated, side-by-side editing experience in which an author edits a

document's source while a live, rendered preview updates in an adjacent pane - with synchronized scrolling that keeps the two views aligned as the author moves through the document. Today, users must manually arrange two tabs and lose their place in the rendered output every time the preview refreshes.

We propose to create a first-class live-preview split editor integrated in core JupyterLab, built on a format-agnostic renderer abstraction so that the same synchronized-editing experience extends beyond Markdown to the full family of plain-text authoring formats used in science and engineering: Markdown, reStructuredText, LaTeX, or Typst.

The split-view editor will immediately be put to use for the markdown format, with a special focus on **MyST**, but with an architecture allowing for other formats to use it in the future.

Impact rationale

Today, Jupyter's markdown story is fragmented:

- The notebook format specifies that markdown cells hold "Markdown" but commits to no flavor, so every interface, core JupyterLab included, ships an ad-hoc rendering stack that re-derives "Jupyter Markdown" independently.
- Meanwhile the Jupyter community has, over several years, built its own purpose-fit markdown dialect, MyST, with first-class cross-references, numbered equations, citations, and rich tables, a mature TypeScript renderer (MyST-JS, [mystmd](#)), and a native JupyterLab extension ([jupyterlab-myst](#)) that already renders MyST inside JupyterLab cells and document previews.

The same gap haunts notebook serialization: the [.ipynb](#) JSON format is hostile to version control and inaccessible to standard text-processing tools, and a family of JEPs (#102, #103, #138) that emerged from the well-attended 2023 community workshop on the notebook file format stalled for years for lack of funded maintenance. These are not niche concerns: the 2023 notebook-file-format workshop and the 2026 Paris-Saclay workshop ("Demystifying MyST Markdown in Education") both drew very strong

attendance, signaling sustained, broad-based demand for a coherent plain-text authoring story across the ecosystem.

This proposal is a unique opportunity to close that gap by dogfooding Jupyter's own technologies inside Jupyter's own flagship product. Adopting the MyST TypeScript renderer as the default markdown renderer in core JupyterLab turns a community-built, Jupyter-native dialect into what every JupyterLab user gets out of the box, replacing the informal format, standardized, conservative MyST subset defined by a JEP. Pairing that with a shepherded text-based-notebook JEP and a reference implementation finally carries the long-stalled 2023 proposals to consensus and adoption, and gives notebooks a first-class text representation with a conformant parser shared across the ecosystem rather than yet another independent format.

The live split-view editor completes the picture, delivering the synchronized side-by-side authoring experience that the JupyterLab text editor and Markdown renderer have never offered together, and built on a format-agnostic abstraction so it can be extended to reStructuredText, LaTeX, or Typst.

Impact Metrics: How do we define success?

Each workstream is tied to a concrete, verifiable artifact deliverable within the one-year funding period.

1. MyST as the standard markdown flavor in Jupyter notebooks

- A JEP defining a strict, enumerated subset of MyST as the official Jupyter markdown format is submitted to a community vote within the funding period.
- The MyST TypeScript renderer is merged as the default markdown renderer in core JupyterLab, replacing the current `marked-plus-extensions` stack — verifiable via the upstream JupyterLab release that ships it.

2. Text-based notebook format

- A JEP specifying the text-based notebook serialization (building on JEP #102 and its companions) is submitted to a community vote.

- A conformant reference implementation (`plainb` / `jupyter-plainb`) round-trips notebooks through the serialization defined by the specification.

3. Live split-view editor for JupyterLab

- A synchronized split-view editor with live rendered preview and synchronized scrolling is merged in core JupyterLab.
- The editor exposes a format-agnostic renderer abstraction, demonstrated for MyST and at least one additional plain-text format.

All three deliverables are achievable on this timeline: QuantStack has already authored and shipped substantial JupyterLab extensions (JupyterCAD, JupyterGIS), giving the team direct, proven experience landing complex, interactive editing surfaces in core JupyterLab.

De-risk the proposal

The central risk is not technical feasibility but governance momentum: standardization documents are long-lived, and features that enter the format are difficult to remove, so a JEP that moves too fast risks the very conservatism this proposal advocates. We mitigate this by keeping the standardized surface small: an enumerated MyST subset rather than the open-ended extensibility model, and by routing every addition through its own future JEP. The JEP process is shepherded, not rushed: the workplan is to carry the long-stalled 2023 proposals (#102 and companions) to consensus, drawing on the community traction they already attracted, rather than introducing a novel specification from scratch.

Technical risk is low and well-covered by prior art. QuantStack's team comprises several core maintainers of Jupyter, as well as the developers of `plainb`, which already edit notebooks as plain text and ship parsers for JupyterText percent, Sphinx-Gallery, classic Markdown, and MyST-NB. The MyST TypeScript renderer (`mystmd`) is mature and already powers `jupyterlab-myst` inside JupyterLab. QuantStack's team members also authored and shipped large, interactive JupyterLab extensions - JupyterCAD and JupyterGIS - demonstrating direct, proven experience building complex editing surfaces in JupyterLab, which is precisely the capability the split-view editor requires.

If a deliverable slips, the three workstreams are deliberately independent: each produces value on its own, so a delay in the JEP schedule does not block the renderer adoption or the split-view editor, and each can ship and be maintained on its own cadence.

Budget

High-level itemized list of approximate costs and explanation of costs. This doesn't need to be exact - focus on outcomes and reasonable expected cost.

DESCRIPTION	AMOUNT	NOTES
1. The markdown flavor in Jupyter notebooks	\$15k	
2. Text-based notebooks	\$40k	
3. A live split-view editor for JupyterLab	\$25k	
TOTAL BUDGET	\$80k	

Endorsements

We strongly encourage authors to seek and include endorsements from Jupyter councils or committees affected by the proposal. Each endorsement should be a brief paragraph about the impact of the proposed work from the council or committee's perspective, and express support for the work and a willingness to receive it.

This proposal was developed in part at the Demystifying MyST in Education workshop, held in Paris in June of this year, as noted in [the overview blog post](#) by event organizer Chiara Marmo and JupyterHub / Jupyter Book community manager Serena Bonaretti. As a follow up to that event, Sylvain engaged with the Jupyter Book community and sought input and engagement from maintainers and community members in the development of this proposal. We are very proud to endorse this proposal and are excited to collaborate to deliver a coherent authoring experience for Jupyter users. – Kirstie Whitaker, maintainer for Jupyter Book

Appendix

Any further optional context for the proposal. We put this at the bottom so that optional context will not clutter the proposal above.