

Second, there's a community precedent worth knowing: the *pandoc-zotxt* / "*pandoc to Zotero live citations*" approach does essentially this, so the field format is well documented and stable. I'd want to do a round-trip test on a small document with your actual exported CSL-JSON before trusting it on a real paper.

So the architecture your vision implies is entirely buildable here: your defensive Markdown → Lua filter (enforcement + style mapping) → Pandoc with your reference.docx → optional python-docx/lxml post-pass for Zotero fields and any field codes (TOC, cross-references) → a docx that opens in Word with nothing but named styles and live citations.

The one piece I'd need from you to make it concrete is the reference.docx (or a style spec from which to generate one) and a sample CSL-JSON export.

Integration into Microsoft Source Manager

Paul

A pro forma question. No LLM I have known has ever managed to write into Microsoft Source Manager in a useful fashion.

Fable

[Grappling with XML content type configurations for Word documents]

This is exactly the sort of claim I should demonstrate rather than assert. Word's Source Manager is tractable – the reason most attempts fail is that python-docx has no API for it, so people give up at the library level. But under the hood it's just a CustomXML part in the b: (bibliography) namespace plus CITATION fields and a BIBLIOGRAPHY SDT. Let me actually build one.

[Identified technical risks and resolved to verify XML content type handling]

One moment – I want to verify one packaging detail that's a classic failure point before I claim victory.

[Checking whether [Content_Types].xml declares a content type for the customXml item – a common cause of 'unreadable content' errors]

Good – the package is sound. So, to answer the pro forma question properly: the reason no LLM has managed Source Manager "in a useful fashion" is, I suspect, that they attempt it through python-docx's API (which has zero bibliography support) or through Pandoc (which has none either), and stop there.

But Word's native citation system is fully documented OOXML, and it's reachable by exactly the XML-level route we discussed last turn. Specifically, it lives in three places:

1. **A CustomXML part** (customXml/item1.xml) in the b: bibliography namespace – this *is* the document's source list. Anything in here appears in Source Manager's **Current List** when the document opens.
2. **CITATION TagName field codes** in the body, which Word resolves against that part and renders per the selected style (the \l 2057 is the en-GB locale ID, since you're UK-based).

3. **A BIBLIOGRAPHY field inside an SDT** from the Bibliographies gallery, so Insert Bibliography/update behaves natively.

The demo file [supplied] contains all three: one source (Venkatraman 1994 in *Sloan Management Review*, fittingly), a live citation field in the text, and an empty bibliography SDT. Every XML part validates, and the content-type and relationship plumbing is in place. The acid test I cannot perform here is opening it in actual Word, so I'd ask you to do exactly that and check three things:

the document opens without an "unreadable content" repair prompt;

References → Manage Sources shows Venkatraman in the Current List; and

pressing Ctrl+A, F9 (or right-click → Update Field on the bibliography) renders both citation and bibliography in whatever style your Word has selected.

Two honest caveats.

First, Source Manager's **Master List** (Sources.xml in your local AppData) is a file on your machine, so I can't write to it directly – but Word offers "Copy to Master List" in Manage Sources, and I can equally generate a standalone Sources.xml you import via Source Manager's Browse button, which amounts to bulk-loading your master list.

Second, Word's b: schema is markedly poorer than CSL – its source types and fields are a strict subset of what Zotero holds – so a Zotero→Source Manager conversion is lossy by design, which is one reason serious reference managers bypass it entirely. If the demo round-trips cleanly on your machine, the useful production capability is: CSL-JSON or your Markdown footnotes in, native Source Manager citations out, batched across a whole document.

Tell me what Word says when you open it.

[Source manager demo.docx]

Paul again

This works well. I clicked on Bibliography and it appeared. I then added a new reference (to make sure Source Manager was not corrupt) and after another click it appeared too. The file now looks like:

Venkatraman argued that IT-enabled transformation occurs at five levels.
(Venkatraman, 1994) A bibliography follows. (Bacsich, 2026)

References

Bacsich. (2026). First time with Fable. Sheffield: Matic Media. Venkatraman, N. (1994). IT-enabled business transformation: from automation to business scope redefinition. *Sloan Management Review*, 35(2), 73-87.

Fable again

[Validated citation system functionality and mapped production workflow possibilities]

Excellent – and your corruption test was exactly the right one. The fact that Word let you add Bacsich (2026) through the normal UI and re-render the bibliography alongside

my injected source means the CustomXML part isn't just *readable*, it's a fully live peer of anything Word creates itself: Source Manager parsed it, merged it into the Current List, and the BIBLIOGRAPHY field treats both sources identically. There's no second-class "imported" state lurking in the file. (And I note "First time with Fable" – I'm honoured to be a citable event in the Matic Media catalogue.)

So the capability is now established end-to-end, which means the production pipeline we sketched is real rather than speculative: your defensive Markdown with citation markers → Pandoc with your reference.docx for styles → an XML post-pass that writes the b:Sources part, swaps markers for CITATION fields, and appends the bibliography SDT. The output is a document Word regards as entirely its own.

A few things worth settling before scaling this up to a real paper:

Tag and GUID discipline. Each source needs a unique b:Tag (the citation key, effectively) and ideally a stable GUID,¹² so that regenerating a document doesn't spawn duplicates in your Master List if you copy sources across. The natural move is to derive tags deterministically – Word's own convention is first-three-letters-plus-year (Ven94, Bac26), but anything stable works, and if you have existing citekeys (Zotero/BibTeX style) I can carry those through.

The source of truth question. You now have three candidate canonical stores – Zotero, Word's Master List, and the Markdown itself. My instinct, given your workflow, is Zotero as canonical, CSL-JSON export as the interchange, with a converter mapping CSL → b: types. That mapping is lossy (Word's schema has perhaps a third of CSL's expressiveness – no DOI field on many types, crude contributor roles, no original-date), so the converter should warn when it drops data rather than doing so silently. The reverse direction – Word b: back to CSL – is trivially safe since it's an injection into a richer schema.

Citation marker syntax in your Markdown. Something that survives your defensive subset and is unambiguous to a filter – [`@Ven94`] Pandoc-style is the obvious candidate since it's already a recognised token, and a Lua filter can intercept it before citeproc ever gets involved.

If you send me a CSL-JSON export of a handful of real Zotero items – ideally messy ones, with editors, corporate authors, reports, web pages – I'll build and test the converter against the awkward cases first, since that's where lossy mappings bite. Then "Back to Venkatraman" could plausibly be the first full-scale output of the pipeline.

Artwork and general nonlinear layouts in Word

Paul

Thanks, and congratulations! Since one main bastion has fallen, let's look at some others in Word. There are various kinds of text+artwork features in Word such as classic annotated diagrams, standard images (external), flowcharts and box+arrow constructs, side headings (out to left of page etc). Is there a side-effect of your "not image but

¹² For more on GUID see https://en.wikipedia.org/wiki/Universally_unique_identifier