Word Macro to Form and Template

Convert an old Word macro template into an HTML form and a Jinja2 assembler

How to Convert Your Word Macro Templates into a Modern Form + Template System Using Claude

Who This Is For

If you've been using Microsoft Word macro templates to generate documents (wills, contracts, pleadings, intake forms, whatever) and they work, **don't throw them away.** The language in those templates is battle-tested. The logic is proven. You don't need AI to rewrite your documents from scratch every time.

What you *can* use AI for is **modernizing the delivery system**: taking that macro logic and converting it into an HTML intake form and a Jinja2 template engine that assembles your documents cleanly, reliably, and without Word's fragile macro layer.

This guide walks you through exactly how to do that with Claude.

The Core Idea

Your Word macro template has three things baked into it:

1. **The language**: the actual text of your document (clauses, paragraphs, boilerplate)
2. **The fields**: the variables that change per client (names, dates, addresses, specific choices)
3. **The logic**: the conditionals that decide which sections appear, which language to use, and how things get assembled

You're going to preserve all three. You're just moving them out of VBA and into a system that's easier to maintain, version-control, and deploy.

The output is two files: - An **HTML form** that collects the field data from the user - A **Jinja2 template** that takes that data and assembles the final document

Step 1: Prepare Your Word Macro Template

Before you hand anything to Claude, do a quick inventory of your template:

- Open it in Word and enable macros
- Walk through the template as if you were filling it out for a real client
- Note every field (text inputs, dropdowns, checkboxes, yes/no toggles)
- Note every conditional, the places where the document changes based on a selection (e.g., "if married, include spousal provisions")
- If you have a user guide or cheat sheet for the template, gather that too

Save the template as-is. You'll be uploading it to Claude along with your notes.

Step 2: Upload to Claude and Extract the Logic

Upload your `.docm` or `.dotm` file to Claude (or paste the VBA code directly if you can access it via the Visual Basic Editor in Word, `Alt+F11`).

Here's a prompt structure that works well:

> **Prompt:**
> I'm uploading a Word macro template that generates [describe your document type, e.g., "a revocable living trust"]. I want to modernize it into two components:
>
> 1. **An HTML form** that collects all the input fields this template uses
> 2. **A Jinja2 template** that assembles the final document from that form data
>
> Please analyze this template and:
>
> - Identify every input field (text, date, dropdown, checkbox, yes/no)
> - Map out all conditional logic (which sections appear based on which choices)
> - List any repeating sections (e.g., multiple beneficiaries, multiple assets)
>
> **Do not change the document language.** The legal text stays exactly as written in the template. You're only extracting the structure, fields, and logic.
> Start by giving me a complete field inventory and logic map before generating any code.

This two-phase approach is important. You want Claude to show you the extracted logic *first* so you can verify it before any code gets written. Check the field list and conditionals against your own knowledge of the template. Catch errors here, not after the code is built.

Step 3: Generate the HTML Form

Once you've verified the logic map, ask Claude to build the form:

Prompt:
That logic map looks correct. Now build an HTML form that:

- Collects every field you identified
- Groups related fields into logical sections (e.g., "Client Information," "Spouse Information," "Beneficiary Designations")
- Uses appropriate input types (text, date, select dropdowns, checkboxes, radio buttons)
- Handles repeating sections with an "Add Another" button (e.g., for multiple beneficiaries)
- Shows/hides conditional sections based on user selections (e.g., hide "Spouse" section if marital status is "Single")
- Uses clean, readable CSS, nothing fancy, just functional
- Outputs form data as a JSON object when submitted

Keep it in a single HTML file with embedded CSS and JavaScript.

The form should output JSON because that's what your Jinja2 template will consume. The JSON structure becomes your data contract between the form and the template.

Step 4: Generate the Jinja2 Template

Now for the document assembly side:

Prompt:
Now build a Jinja2 template that:

- Takes the JSON output from the HTML form as its input data
- Reproduces the exact document language from the original Word template
- Implements all the conditional logic you identified (use Jinja2 `{% if %}`, `{% for %}`, `{% macro %}` blocks as appropriate)
- Handles repeating sections (e.g., loop through beneficiaries)
- Preserves the document's structure and formatting intent

Critical: Do not rephrase, rewrite, or "improve" any of the original document language. Use the exact text from the template.
Include comments in the Jinja2 template explaining each conditional block so it's maintainable.

Step 5: Review and Test

This is where you put it through its paces:

1. **Compare the Jinja2 template against your original Word template.** Walk through both side-by-side and verify every clause, every conditional, every field reference.
2. **Test with real scenarios.** Feed in sample data that exercises different conditional paths (married vs. single, one beneficiary vs. five, with trust vs. without).
3. **Check edge cases.** What happens with blank fields? What if someone selects contradictory options?

Ask Claude to help you test:

> **Prompt:**
> Generate three sample JSON test datasets that exercise different conditional paths in this template: 1. A simple, straightforward case (single person, one beneficiary) 2. A complex case (married, multiple beneficiaries, all optional sections triggered) 3. An edge case (minimum required fields only)
> Then show me the rendered output for each so I can verify correctness.

Step 6: Set Up a Local Testing Environment

You don't need a production server to test this. Claude can help you set up a simple internal web application that serves your HTML form, accepts the input, runs it through your Jinja2 template, and shows you the assembled document. All of it runs on your own machine.

Here's how to ask for it:

> **Prompt:**
> I want to set up a simple local web application to test my HTML form and Jinja2 template. The app should:
> - Serve the HTML form on localhost
> - Accept form submissions
> - Run the submitted data through the Jinja2 template
> - Display the assembled document in the browser
> - Optionally export to PDF or Word (.docx)
>
> Use Python with Flask. Keep it as simple as possible. Give me: 1. A `requirements.txt` with the needed packages 2. An `app.py` that runs the server 3. Instructions to install and start it

> Assume I'm starting from scratch and may not have Python installed.

Claude will generate a lightweight Flask application. The typical setup looks like this:

Install Python (if you haven't already): - Windows: Download from python.org. Check "Add to PATH" during install. - Mac: `brew install python3` or download from python.org - Linux: Likely already installed; if not, `sudo apt install python3 python3-pip`

Install dependencies and run:

```
pip install -r requirements.txt
python app.py
```

Then open `http://localhost:5000` (or whatever port Claude configures) in your browser. You'll see your form, fill it out, submit, and get your assembled document back.

This runs entirely on your machine. No cloud services, no subscriptions, no data leaving your network.

Tips and Gotchas

Don't let AI rewrite your language. This is the most important rule. Your document text was drafted (and probably revised multiple times) by a human who knew what they were doing. AI is building the plumbing, not writing the content.

Version control your templates. Once you have your Jinja2 templates working, put them in Git. You can track every change, roll back mistakes, and maintain an audit trail of what changed and when.

Start with one template. Don't try to convert your entire template library at once. Pick one, get it working end to end, learn the process, then scale.

Repeating sections are the hardest part. Things like beneficiary lists, asset schedules, or trustee successions require Jinja2 `{% for %}` loops and often nested conditionals. Double-check these carefully.

Test with real data patterns. Don't just test with perfect data. Test with the weird stuff your clients actually give you: missing middle names, international addresses, unusual family structures, contradictory selections.

The HTML form and Jinja2 template are independent. You can update one without touching the other, as long as the JSON data contract stays the same. Need to add a field? Add it to the form, add it to the template, make sure both reference the same key name.

What You End Up With

When you're done, you have:

- A **clean HTML form** that anyone can fill out in a browser, no Word required
- A **Jinja2 template** that deterministically assembles your document (same inputs always produce the same output)
- A **local testing server** to verify everything works before you deploy it anywhere
- **No AI in the loop at runtime**, just pure template logic, not generated text
- **No subscription fees** for document assembly software

Your document language stays yours. The AI just helped you build a better delivery system for it.

More Resources

For more guides, tools, and resources on using AI in legal practice, visit nemolegal.com/tools.

This guide is provided as-is for educational purposes. Adapt it to your own practice, jurisdiction, and document requirements.