Docassemble Guided Interview Tips for Legal Aiders

A guide for non-“techies” interested in creating Docassemble guided interviews using the Suffolk LIT Lab’s Document Assembly Line

By Kris Surette, Legal Services Vermont
December 2025

Table of Contents

Introduction

This guide was created by Legal Services Vermont as part of a Legal Services Corporation TIG grant project. It provides some tips and tricks for members of the legal aid community and others who want to learn to create Docassemble guided interviews.

Legal Services Vermont (LSV) used Suffolk LIT Lab's Document Assembly Line to build guided interviews. These tips reflect our experience in turning fillable PDF forms into accessible and user-friendly guided interviews. At the end of the guided interviews, the user can download, print or email the resulting PDFs to deliver to the court.

Our first six VTCourtForms guided interviews were for highly used Vermont court forms. They were based on fillable PDFs from the court. We are now also developing MS Word-based pleadings, such as an answer to an eviction complaint.

In this guide we offer links to the comprehensive Suffolk LIT Lab Document Assembly Line documentation and Docassemble documentation. The links are typically at the bottom of each section.

Tips to get started

1. **Get plugged into the Suffolk LIT Lab Document Assembly Line community**

Suffolk LIT Lab's Document Assembly Line (see https://assemblyline.suffolklitlab.org/) is a toolbox that helps you build Docassemble-based guided interviews. It has:

- tools to make it easier to convert existing PDF and DOCX forms into draft Docassemble interviews
- features that make it easier to use key Docassemble features
- a library of pre-built commonly used questions
- an accessible, mobile-friendly layout that can be customized, and
- an online "playground" where you can develop and test your guided interviews.

In addition, Suffolk LIT Lab gathers legal aid staffers and others from around the country for weekly meetings to discuss everyone's project blockers and successes. They also offer one-on-one help at online office hours, offer live and recorded training sessions, and host Teams channels where people can ask questions and learn from one another. This was incredibly important as we learned the process of creating, customizing and launching a Docassemble guided interview. The camaraderie also helps a lot, too!

2. **Do some planning**

- Choose a very short PDF form to do first so you can learn while doing an actual project.
- Ask the Judiciary (or other owner of the form) if they plan to change that form anytime soon. If they are, choose another.
- Make sure you know how the form needs to be filled out. If you need to, ask what the fields are really asking for.
- Make sure you know how the form will need to be signed and if a notary signature is needed.
- Make sure you know the details of how someone will file the final document(s): in paper form, email or e-filing.
- Plan who the target form user is; what stage are they in their "journey".
- Plan what intro information they will need before starting the guided interview.
- Think about how you might group the questions that need to be asked. About You, About the Other Party, etc.
- Plan what information should be included in a "Next Steps" instruction document that is also produced when the document(s) are produced.
- Figure out how to host your interviews. Suffolk LIT Lab offers centralized Docassemble hosting. Or, hire some techies to set up and maintain your own secure server.
- Documentation: https://assemblyline.suffolklitlab.org/docs/get_started/planning

3. **Dive into some training**

- It will take hours of reading, watching training videos, getting help, attending online trainings, and trial and error to create custom guided interviews.
- To customize interviews, you need to learn coding. It's probably best if you've had some exposure to HTML, other coding or other technology. Even having one or two classes in the past are helpful.
- Read the Document Assembly Line (AL) documentation. Review the Get Started section. Review the Introduction to Docassemble for a readable, step-by-step guide to the core features and syntax of Docassemble. See the official Docassemble documentation for many more details. Both the Assembly Line and Docassemble documentation is searchable by in-site search, and also by Google.
- Try the AL's "Hello, World" exercises.
- Watch the training videos that Suffolk LIT Lab has made and attend some live trainings.
- Contact Suffolk LIT Lab to start attending weekly meetings to hear what people are working on. You will be able to ask questions here including what your next steps might be, and specific coding questions.
- Documentation: https://assemblyline.suffolklitlab.org/docs/get_started

4. **Prepare your PDF form(s)**

- Open the PDF form in your PDF editor. We use Adobe Acrobat. You are going to edit the fields a bit.
- Open the AL online documentation to see the exact field names you should use in your PDF form.
- Edit the form fields to use as many of their standardized variable names as possible. Type the field names exactly as they suggest.
- When you use their standardized fields, during the next step, the AL Weaver will automatically add questions that you can later customize. It gives you a head-start when making the guided interview.
- On your PDFs, don't use the same label for a field twice. To work around this issue, annotate the second or third field with two underscores and any digit, like this: __1 and __2. You must give each field in your PDF file a unique name.
- If you will be creating a package which has more than one PDF, it's safest to make sure all the fields in the package of PDFs have their own unique name.
- Consider keeping track of your common field and variable names in a spreadsheet. It will be a handy reference when working with the code, and when making an interview that involves more than one form. Then you can also keep reusing common names in other interviews.
- You can edit your PDF to:
    - Split one field into two so you can make use of the standardized fields. Example: a town field and a state field even when it's just one field on the form.
    - You will soon be using a tool called The Weaver. The Weaver cannot handle dropdown fields. Change those fields to a text field for now.
    - The Weaver cannot handle radio buttons. Change those to checkboxes for now.
    - Make the fields have the same font and size to look good. I suggest Calibri or Arial 12 point or larger for accessibility, if possible.
    - Make your fields neat and well aligned! This PDF will be used for the final product.
    - Edit the meta data Properties of the PDF. You want a clear document name and author for accessibility and other reasons.
- Tip: **Ask for help early when you get stuck**. It's very frustrating to get stuck and you don't need that frustration when someone can get you un-stuck pretty easily!
- At the end, use Acrobat's Make Accessible button and walk through their suggestions. You want to be sure the text and fields are tagged. You'll want to see if the reading order of the document is correct. Then, outside of Acrobat, try tabbing your way through all the form options with your Tab key and arrow keys. To be accessible, you need to be able to do that. The user will get this PDF all filled out. Make the document as accessible as possible for people who may use a screen reader or other assistive technology. It's very difficult to make them perfectly accessible, but do what you can! Follow the tips on page 81 of this

accessibility toolkit to check on the accessibility problems and fix as many as you can. Why does this matter? Because we'd like all users to be able to preview their completed PDF before they finalize it and give it to the court.
- By default, the PDF files that are created by the guided interviews can be edited by the user. If you want to prevent users from editing the created form, set editable to False.

```
attachment:
  name: my form
  filename: my-form
  pdf template file: my-form-template.pdf
  editable: False
```

- Documentation:
  - https://suffolklitlab.org/docassemble-AssemblyLine-documentation/docs/pdfs
  - https://assemblyline.suffolklitlab.org/docs/authoring/label_variables
  - https://docassemble.org/docs/documents.html#editable

Note: You can also use MS Word DOCX files as templates. We did not do that in our first interviews, but you can learn more about preparing and using DOCX template files instead of PDF template files: https://assemblyline.suffolklitlab.org/docs/authoring/docx

5. **Use the Assembly Line (AL) Weaver**

- The AL Weaver will help you create a draft guided interview package.
- Drag your prepared PDF into the AL Weaver.
- If you have a few forms that are **always** filed together, consider putting them into the Weaver at the same time. Select them all and drag them into the Weaver together.
- If you want individual interviews for multiple forms, put them in the Weaver separately.
- Tip: Leave plenty of time in the day to go through the Weaver process! You cannot stop part way through.
- Have the original forms nearby so you can consult them.
- The Weaver leads you through a bunch of questions about the form fields and about your intended guided interview.
- Remember that later you can change the order of your questions and the automated questions. You can edit questions, add new ones or delete questions. You can customize the automated questions or eliminate them if they aren't working well for your form. You can also add or edit the introductory information at the start of your guided interview.
- At the end, the Weaver will create a "package" for you that you will download to your computer. Then, in your "Playground" in the Document Assembly Line system, you will upload the package.
- The Playground is where you customize the draft interview, test it and fix bugs.

- Documentation:
  - https://assemblyline.suffolklitlab.org/docs/authoring/generating_code
  - https://assemblyline.suffolklitlab.org/docs/authoring/customizing_interview

6. **Get on GitHub so you can save versions of your work**

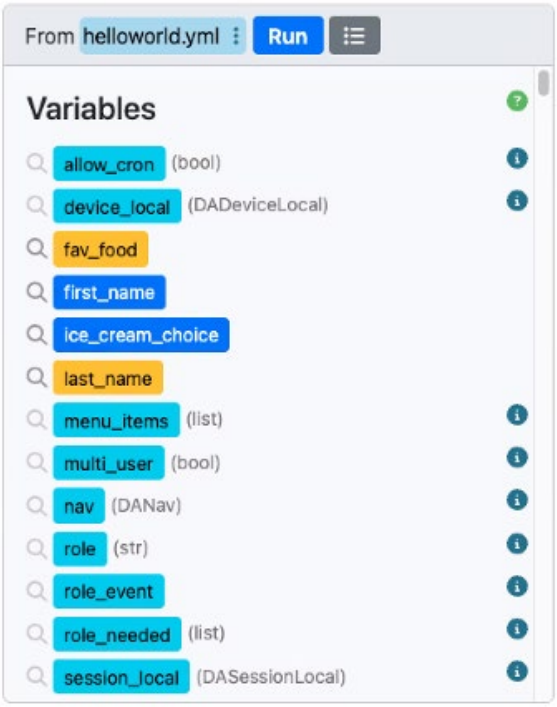
- GitHub is where you keep your official code. During every session where you make improvements to your guided interview, you will "commit" changes to GitHub.
- It's best to commit early and often. That way, when you screw something up, you have a recent copy of code to fall back to!
- Documentation: https://assemblyline.suffolklitlab.org/docs/authoring/github

7. **Try out the draft guided interview in the Playground**

- In the Playground, you will see the main yml file (.yml) for your interview.
- You'll notice that the AL Weaver changed your field names to use periods and things like [0]. users[0] means it's the first user that's defined in the guided interview and on the form. users[1] would mean it's the second user. (Notice that zero is used to mark the first one in a list!)
- Try out the interview by hitting Save and Run button. The interview will open in a new tab. Keep that tab open because you can refresh that screen later after you Save a change in the Playground.
- You may have some initial errors because it was an automated process through the Weaver. You will probably need to ask for help to identify what is wrong.
- Tips for troubleshooting errors:
  - As you click through the draft interview, at the top of the screen you will see red text that tells you where in the code that screen came from (the "id"). It also tells you what variable the program was trying to define by showing that question/screen to you.
  - Also at the top, you will see this < / > That is a button so you can see the source code of that screen. When you click on it, the source code will show up below the questions on that screen.
  - If you scroll down the page, you will also see how the program got to that certain point in the interview.
  - These things help figure out where in the code the question is coming from so you can edit that code as needed.
- Tip: **Ask for help early when you get stuck**. It's very frustrating to get stuck and you don't need that frustration when someone can get you un-stuck pretty easily!
- Documentation: https://assemblyline.suffolklitlab.org/docs/docassemble_intro/basic-troubleshooting

8. **What you see in the Assembly Line Playground (the code editing screen)**

- The main part of the Playground is the editing area where you edit the code in your yml file. Here's a rough outline of the main chunks of code you will see in the yml file in your Playground:
    - list of yml files you need to "include" in your "package" to work with your guided interview (example: assembly_line.yml)
    - "objects" you will gather details about – like people, jobs or things
    - various "metadata"
    - "sections" of your guided interview to be shown in the side navigation
    - "interview order" used to gather the details needed for your form
    - a more general "main interview order" which includes code to show your introductory screen, preview screen, review (editing) screen, signature screen and download screen
    - "question blocks" for all the non-automated questions in your interview (look for question: |)
    - "code blocks" as needed (look for code: |)
    - code for the "review screen", which is where a user can edit their answers if needed (look for review:)
    - overflow fields and "addendum" code, if your user may need an addendum for lengthy answers
    - "attachment" objects and attachment code, which tells the program what values to insert in the PDF attachments (look for attachments:)
- You will also see this rectangle in the Playground, on the right side of screen:
    - dark blue variables are currently defined (used) in the interview
    - yellow variables are currently not defined in the interview
    - red ones are undefined variable names – bad! If you see a lot of red over here, chances are you have a mistake in your code.
- If you scroll down in the Playground, you can also see some example code blocks to use as you get familiar with Docassemble and the Assembly Line.



9. **Edit the code to customize and fix things**

- Carefully edit the code in the Playground. Try doing edits one at a time, then hit Save, or Save and Run, to see if they work.
- Capitalization, spelling and spacing matter! Programming languages are picky!

- Use the Tab key to create indenting that is required in your code.
- If you are trying to fix something, try “commenting out” some code first instead of deleting it or overwriting it. Use a # at the start of the line to comment it out. The program will skip those lines of code. This saves the original code because you may not actually decide to delete it! Example:

```
#id: add motion to dismiss if no lease
#code: |
#  motion_to_dismiss_attachment.enabled = defense_lease_not_attached
```

- At the top of your yml file, you can even add a #TO DO section that is commented out. You can jot down the things you want to do next to your code. It is also helpful to identify main sections of your code by adding comments like:

```
---
########## REVIEW SCREEN ############
---
```

- As you try out your interview, you will see your custom questions as well as AL’s automated questions such as questions about a person’s name, birthdate and address. You can tell which questions are not from your interview because when you look at the source code they say they are “from” another file like from ql_baseline.yml (the AL question library – “ql”!).
- If you want to tweak the way an automated question is asked, you can copy the source code you see under step 7 above. You will then paste that code into your interview code and edit it. Because this code is now directly in your interview, it will override the question library code that was being used.
- You can rearrange the order of the interview by editing the “interview order.” The interview order is the instruction list that the program follows from top to bottom to go and define each variable in the list. The program defines the variables by asking your questions and question library questions. Example:

```
---
########## INTERVIEW ORDER ############
---
comment: |
  Controls order and branching logic for questions specific to this form
id: interview_order_VT_813
code: |
  nav.set_section('intro')
  set_progress(5)
  VT_813A_intro
  nav.set_section('about_you')
  users.gather()
  users[0].address.address
  users[0].daytime_phone_number
  users[0].email
  users[0].birthdate
```

- Tips:
  - All the code blocks in the lower part of your code must have at least one of their fields referenced in the Interview Order. This will make the program go and look for that field, and as it does so, it will read that whole code block.
  - .gather() is code that means the program will go gather information about the person or thing.
  - If you want to collect all information about a person, such as a spouse, make sure there is an “object” for spouse listed near the top of your code. The object should be an “Individual” or “ALIndividual”.
  - When you want to refer to an object generically deeper in the code, you can use x to represent that object. Kind of like algebra! That pattern is called a generic object.
  - Add a unique id to every question block. It can just be a copy of the question text with punctuation removed. This will help you find things as your code gets longer.
- After you get through the interview without an error, be sure to commit the interview code to GitHub. Commit to GitHub often!
- In the draft interview, note how questions are asked and in what order and in what groupings. Think about how you might improve these things for a better experience for the user.
- Documentation:
  - https://assemblyline.suffolklitlab.org/docs/authoring/customizing_interview#how-to-edit-your-interview
  - https://assemblyline.suffolklitlab.org/docs/authoring/customizing_interview#understanding-the-question-library
  - https://assemblyline.suffolklitlab.org/docs/docassemble_intro/controlling-interview-order/

10. Create a “Next Steps” document

- The Assembly Line Weaver makes a simple Next Steps template for you. You can download that Word document and edit it to make it your own. Many Docassemble interviews offer a Next Steps document to the user when they are downloading their form(s). It helps them understand what to do next.
- You can add some code to the Word doc to make use of info that you collected during the guided interview, like offering the contact info for the user’s court.
- Make it an accessible Word document. Use proper headers, 14-point font like Arial, and high-contrast between text and the background. Use the Word accessibility checker.
- Go back to the Playground, click on Folders, then Templates. Upload this Next Steps template. If you are replacing a file, delete the old one and upload using the very same filename.

- Documentation: https://assemblyline.suffolklitlab.org/docs/authoring/docx

11. **Get help**

- Get on the Suffolk Teams group so you can ask about the errors you are getting and ask coding questions. Other users will answer you if they know the answer, or Suffolk experts will answer you.
- If a Suffolk expert is offering virtual office hours, go to them with some specific questions.
- You can also join the Docassemble Slack channel and ask questions there.
- **Don't waste much time trying to figure things out on your own at first.** It can be very frustrating. Let someone point you in the right direction. You'll get better at looking up documentation and coding soon enough!
- Documentation: https://assemblyline.suffolklitlab.org/docs/get_started#join-the-community

Code samples for customizing your interview

Below you can get an idea about what features, code and other things are called in Docassemble. That will help you when you need to go to find helpful documentation. We offer sample code and pertinent links.

Question blocks

Below is a question block for gathering the user's phone and email. The user is called users[0].

```
---
id: phone and email
question: |
  Your phone and email
subquestion: |
  Tell us how the court can reach you.
fields:
  - "Your home or cell phone number": users[0].phone_number
    datatype: al_international_phone
  - "Your email address": users[0].email
    maxlength: 38
    required: false
---
```

Main types of form fields you can use in your questions

Date

```
---
question: |
  When did you receive the complaint?
fields:
  - Date: sued_when_served
    datatype: date
---
```

Checkbox where checking the box = yes

```
  - "Check this box if the defendant gets SSDI": gets_ssdi
    datatype: yesno
```

Radio buttons (round buttons) yes/no choices

```
  - "Do you want custody of this child?": wants_custody
    datatype: yesnoradio
```

Radio buttons yes/no/maybe

```
  - "Does the defendant live in Vermont?": defendant_in_vermont
    datatype: yesnomaybe
```

Radio buttons with yes/no/I don't know or whatever you want for the labels

```
  - Does the defendant own guns or other deadly weapons?: owns_weapons
    input type: radio
    choices:
      - Yes: True
      - No: False
      - I don't know: None
```

Short text field

```
  - "Your email address": users[0].email
    maxlength: 38
```

Big text field

```
  - Tell me about your legal issue: legal_issue_description
    input type: area
    rows: 10
```

or you can set the max number of characters

```
  - Tell me about your legal issue: legal_issue_description
    input type: area
    maxlength: 500
```

and you can add placeholder text in the field box to give a hint of what is expected

```
 - Tell me about your legal issue: legal_issue_description
   input type: area
   maxlength: 500
   hint: My problem started on this date...
```

Checkboxes with multiple choices incl None of Above – where you can choose more than one

If you want to include None of the above in the options:

```
 - "Check all that apply": checkboxes_for_recent_abuse
   datatype: checkboxes
   choices:
     - You were the target of the recent abuse: recent_incident_users1
     - Your children were the target of the recent abuse: recent_incident_children
```

If you don't want to include None of the above:

```
 - "Check all that apply": checkboxes_for_recent_abuse
   datatype: checkboxes
   choices:
     - You were the target of the recent abuse: recent_incident_users1
     - Your children were the target of the recent abuse: recent_incident_children
   none of the above: False
```

You can also change "None of the above" to another phrase, if needed.

Documentation:

- https://docassemble.org/docs/questions.html#question
- https://docassemble.org/docs/fields.html

show if – Show a question only if a previous field was answered a certain way

```
  - "In a word or two, what is your relationship to this family member":
family_member_description
    maxlength: 16
    show if: family_member
```

or

```
show if:
    variable: relationship
    is: Family
```

If you need show if and you are using a calculation or variable from a previous screen:

```
show if:
  code: |
```

```
  longer code goes here
```

- Documentation: https://docassemble.org/docs/fields.html#show%20if

js show if — the js stands for JavaScript

Use this when you have a follow-up question on the **same** screen that depends on an answer above it.

```
fields:
  - Do you have other jobs?: has_jobs
    datatype: yesnoradio
  - How many of these jobs do you have?: jobs.target_number
    datatype: integer
    js show if: |
      val("has_jobs") == true
```

or

```
js show if: |
  val("relationship") == "family_member"
```

- Documentation: https://docassemble.org/docs/fields.html#js%20show%20if

hide if

Just like show if, but it hides the field.

```
 hide if:
   variable: relationship
   is: Family
```

or

```
js hide if: |
  val("relationship") == "acquaintance"
```

- Documentation:
  - https://docassemble.org/docs/fields.html#hide%20if
  - https://docassemble.org/docs/fields.html#js%20hide%20if

Add help text in a collapsible box

This example would have a text box that the user could expand to help them choose which county to pick.

```
---
question: |
  What county is your case in?
fields:
```

```
  - County: user_selected_county
    code: sorted(all_courts.unique_column_values('branch'))
  - note: |
      ${ collapse_template(which_county_to_choose) }
---
template: which_county_to_choose
subject: |
  Not sure which county to pick?
content: |
  If you already have a court case in progress, select the county that is listed on your
court papers.
---
```

- Documentation: https://suffolklitlab.org/docassemble-AssemblyLine-documentation/docs/framework/altoolbox/#collapsible-help-text

To make a field not required

Questions default to being required. If you want to let the user skip a certain question, add required: false.

```
  - "Your email address": users[0].email
    maxlength: 38
    required: false
```

Set the minimum amount of answers/choices that must be made

With multiple choice questions, you can use “minlength” to set the minimum number of choices.

```
fields:
 - "Select all that apply": danger_more_abuse_who
    datatype: checkboxes
    show if: danger_more_abuse
    none of the above: False
    minlength: 1
    choices:
      - "There is danger of more abuse to you": users1
      - "There is danger of more abuse to the child/children": children
```

Another tactic is to add “validation code” in the question block.

```
fields:
  - Phone number: users[0].phone_number
    required: False
  - Work phone number: users1_work_phone_number
    required: False
```

```
  - Email address: users[0].email
    datatype: email
    required: False
validation code: |
  if (not showifdef('users[0].phone_number') and \
      (not showifdef('users[0].email')) and \
      (not showifdef('users1_work_phone_number'))):
    validation_error(word("You need to provide at least one contact method."),
field="users[0].other_contact_method")
```

State dropdown list

This offers a dropdown menu with all the states in the US to choose from.

```
  - State: users[0].mailing_address.state
    code: |
      states_list()
    default: VT
```

Formatting for text you will display on-screen in your interview, or in a PDF document

- Docassemble uses “Mako” for formatting throughout the interview file. Documentation: https://assemblyline.suffolklitlab.org/docs/docassemble_intro/mako
- “Markdown” is used for setting font size, headers, bold, italic and formatted lists in Docassemble. Documentation: https://assemblyline.suffolklitlab.org/docs/docassemble_intro/markdown

Formatting names of other parties

This will print out a neat list of other parties whether there is one or more.

```
${ other_parties.short_list(2) }
```

Formatting dates

This formats a date neatly.

```
${ installments_start_date.format() }
```

When entering a date is optional, use this to print it out:

```
${ format_date_if_defined("other_parties[0].birthdate", format="M/d/yyyy") }
```

Documentation: https://docassemble.org/docs/functions.html#format_date

Auto-computing dates

This will print the date that is 30 days from today.

```
${ today().plus(days=30) }
```

Documentation: https://docassemble.org/docs/fields.html#date

Formatting text to be printed in lower case or upper case

```
${ agrees_to_pay_installments_period.lower() }
${ agrees_to_pay_installments_period.upper() }
```

Add help text in midst of fields with a note

“Note” is a handy tool to allow you to add directions or helpful information where you need it.

```
 - "Please describe the most recent incident with as much detail as possible.":
recent_incident_description
   input type: area
 - note: |
     Check all that apply to this incident
 - You were the target of the recent abuse: recent_incident_users1
   datatype: yesno
```

Documentation: https://docassemble.org/docs/fields.html#note

Separate or group things with a horizontal rule (line)

```
- note: |
   ---
```

Separate or group things with an extra space

```
note: |
 <br/>
```

Advanced formatting: Combining the text of some checkbox choices and a textbox question

This is useful if you are offering some reasons that a user might select from, but you also offer them a text area where they can further explain things in their own words.

```
---
# "!!omap" makes these terms ordered:
# their order here will be their same order when shown to users.
variable name: defenses_terms
data: !!omap
  - do_not_recognize: "I don't recognize this debt."
  - amount_doesnt_match: "The dollar amount the debt collectors are asking of me does
not correspond to the dollar amount I actually owe."
  - already_paid: "I have already paid the debt."
  - bankruptcy: "This debt is discharged due to personal bankruptcy."
  - llc_debt: "This debt is of a Limited Liability Corporation registered in my name."
  - not_orginal_creditor: "The Plaintiff is not the original lender of the debt, and I don't
believe the plaintiff has the right to collect the debt."
  - long_time_since_debt: "It has been six years since I incurred this debt."
---
id: combining individual defenses that were selected and the additional explanation
code: |
  merged_defenses_text = ""
  for item in defenses.true_values():
    merged_defenses_text += defenses_terms[item] + "\n"
  merged_defenses_text += disagrees_additional_explanation
  get_merged_defenses = True
---
```

This code adds pieces of text, called "strings", to each other. "\n" is the new line character in Mako.

On a screen in your interview (like your review screen) you can show a list of the selected checkboxes by using a “for” loop:

```
  % for item in defenses.true_values():
  - **${ defenses_terms[item] }**
  % endfor

${ disagrees_additional_explanation }
```

In your interview order, add get_merged_defenses to do the merging.

In the attachment block for your finished form or document, you can print the whole merged text like this:

```
${ merged_defenses_text }
```

Another advanced option: Explore using the Python join() method.

Documentation: https://docs.python.org/3/library/stdtypes.html#str.join

Adjusting the logic / flow of your interview

Conditional logic — Change the interview order depending on an answer

In the “interview order” section of your code you can use “if” and “else” and “elif” (which means else if). When you do, the next line needs to be indented.

```
recent_incident_date
if not recent_incident_most_serious:
  serious_incident_date
danger_more_abuse
```

Documentation: https://assemblyline.suffolklitlab.org/docs/authoring/customizing_interview#controlling-the-order-of-questions

Conditional logic — Use if and else for displaying info on screen or in document

```
subquestion: |
  % if rental_income:
  You said that your total monthly rental income is
  % if has_schedule_e:
  **${ currency(schedule_e_income/12) }**
  % else:
  **${ currency(monthly_rental_income) }**
  % endif
```

When displaying text like this on the screen or in a document, you need the %.

Use true_values to determine if user selected any checkboxes

```
if len(defenses.true_values()) > 0:
  ask_if_wants_to_file_counterclaim
```

“len” means the number of items, or the length of a list, if you will.

To refer to a particular multiple choice answer (like radio and checkboxes)

If question was “input type: radio” with choices:

```
if when_filing_financial_form == "now":
  start_financial_form
```

But if question was “datatype: checkboxes” with choices:

```
if when_filing_financial_form[“now”]:
  start_financial_form
```

Snapshot code – collect anonymous data about the user sessions

Use snapshot code to collect anonymous info that you can see in the Assembly Line usage report. This could include user zip code, which court they are involved with, etc.

Here's an example from a small claims interview:

```
# Store anonymous data for analytics / statistics
 store_variables_snapshot(
    persistent=True,
    data={
       "zip": showifdef("users[0].address.zip"),
       "county_where_filing": showifdef("trial_court.address.county"),
       "user_has_attorney": showifdef("user_has_attorney"),
      "1st_OP_is_person_or_business": showifdef("other_parties[0].person_type"),
       "claim_principal": showifdef("claim_principal"),
      "claim_interest": showifdef("claim_interest"),
      "claim_court_fee": showifdef("claim_court_fee") if
showifdef("wants_to_claim_court_fee") else 0,
       "reached_interview_end": True,
       "num_forms": len(al_user_bundle.enabled_documents()),
    },
 )
```

Multiple forms in one guided interview

You will probably want to offer two or more forms in a single interview sometime. For example, you want to let the user answer a small claims action, and also fill out the fee waiver form if they have a counterclaim and want to ask for a waiver.

"Include" code block near top of interview

In the small claims example above, you might make a main yml file for the small claims answer and then "include" a yml file for the fee waiver. You will add that fee waiver file to the "include" list at the very top of your interview.

The include code block tells Docassemble that it should also go look for code in other yml files. The order of the list matters. Believe it or not, the bottom item on the list has top priority. So if you have multiple variations of questions that ask about a person's contact information, for example, Docassemble will look at the list and pick the question from the yml file that is lowest on the list.

Here's a sample include code block. It includes four other forms the user may require. The interview also relies on the Assembly Line and phone number validation files, as well as Vermont's file that holds CSS styles, logos and some commonly used text.

```
---
include:
  - docassemble.VTSmallClaimsCertificateOfService:small_claims_ service.yml
  - docassemble.VTFinancialDisclosure:VT_financial_disclosure_to_include.yml
  - docassemble.VTDisclosureOfExemptIncome:disclose_exempt_income_to_include.yml
  - docassemble.VTFeeWaiver:VT_fee_waiver_to_include.yml
  - docassemble.AssemblyLine:assembly_line.yml
  - docassemble.ALToolbox:phone-number-validation.yml
  - docassemble.VTSharedYMLFile:VTSharedYMLFile.yml
---
```

Watch out for using same field names in the PDFs

We've seen that sometimes information gets left out or populated the wrong way in the PDFs when we had multiple forms in one interview. To avoid this: In your PDF template, if you have multiple fields for birthdate, for example, consider making each fieldname unique for each form, such as users1_birthdate__11 or users1_birthdate_feewaiver.

Editing and testing code with multiple yml files

When you have an interview project in your playground and it has multiple yml files, you can use the dropdown menu to toggle between the files. Be sure to save your work before toggling. (You can also open the files in their own browser tabs.) And when you want to see the changes that you made, be sure to go back to the main interview yml file, hit save, and then see the changes you have made.

Objects, lists and their attributes

In coding, "objects" are people or things that have certain characteristics. Instead of holding one piece of information, they can hold several at once.

You will likely make use of "lists" of objects in your interviews — lists of people or things that you are collecting information about.

In this example, we've created a list of users of the interview (typically one user who is filling out the form) as well as children that are shared between the user and their spouse.

```
objects:
  - users: ALPeopleList.using(ask_number=True,target_number=1)
  - shared_children: ALPeopleList.using(complete_attribute='complete')
```

"Attributes" are the characteristics, values or other attributes of an object. Attributes follow the object name with a period.

For example, the user can have users[0].address.city, users[0].name.first, or any custom characteristic you want to add, like users[0].eye_color. Shared_children can have attributes such as shared_children[0].child_support_amount or shared_children[0].birthdate.

See the documentation below about the various ways to gather the information you want for each object in a list.

To print the items that were collected in a list, you can use a “for” loop. This one says that for every “child” (you can use any word there – kid, child, etc.) in the list of shared_children, print their full name and birthdate.

```
    % for child in shared_children:
    * ${ child.name.full() } - Birthdate: ${ child.birthdate }
    % endfor
```

Documentation:

- https://assemblyline.suffolklitlab.org/docs/docassemble_intro/object-oriented-programming
- https://www.nonprofittechy.com/2018/09/12/object-oriented-programming-for-document-assembly-developers/
- https://assemblyline.suffolklitlab.org/docs/docassemble_intro/repeated-information#introduction-to-lists
- https://assemblyline.suffolklitlab.org/docs/components/AssemblyLine/ALGeneral/overview#ALPeopleList
- https://docassemble.org/docs/groups.html#gathering

Checking if an attribute is defined

If you need to check if an attribute of an object has been defined, here’s the code to use: % if item.attribute_defined('birthdate'). In this example, birthdate is the attribute.

```
- Edit: other_parties.revisit
  button: |
    % for item in other_parties:
    **${ item }**

    % if item.attribute_defined('birthdate'):
    Birthdate:
```

```
    ${ item.birthdate }
    % else:
    Unknown
    % endif

    % endfor
```

To consider the number of things in a list

Perhaps you need to know the number of jobs the user has described during the interview. You can use number_gathered().

if jobs.number_gathered() > 0

means if the number of jobs is greater than zero.

Documentation: https://docassemble.org/docs/objects.html#DAList.number_gathered

Another less reliable option is to use len(). When you use len(), you are saying length of, or number of.

if len(jobs) > 0

means if the number of jobs is greater than zero.

Note: The syntax above doesn't work in a DOCX template. You have to do this instead:

{%p if jobs | length > 0 %}

Using len() will trigger the gathering of that list if it hasn't happened already.

number_gathered() is often a better choice because it won't trigger the gathering of that information when it's not needed. number_gathered() will not require that the list be fully gathered before returning a value.

Greater than, less than or equal to

Use > and < and you can also do this:

if len(jobs) >= 1

This means if the number of jobs is greater than or equal to 1.

You will typically use == to ask if something is equal to a value.

Printing out the index number of an item on a list

If you ever have to print the index number of an item on a list (1, 2, 3, etc.). This may come in handy when formatting a numbered list of items.

- use ${loop.index} in Mako for your review screen or PDF template
- use {{ loop.index }} in Jinja for your DOCXtemplate

The “review screen” where a user can edit their answers

There’s a special “review screen” that lets your users edit their answers. Users are not supposed to edit their PDFs! Instead, they should use the Edit Your Answers button or select that choice in the dropdown menu.

Review screen generator

The Weaver will generate a basic review screen for you. You can also try using the Review Screen Generator in your Assembly Line Dashboard.

You should get your interview working really well before working on the review screen. Why? Because any changes you make to the interview code will impact your review screen and create more work for you.

Documentation: https://assemblyline.suffolklitlab.org/docs/authoring/review_screen/

When a review screen Edit block doesn’t appear

The biggest frustration when you are adding to and customizing the review screen is that when Docassemble thinks a variable is not defined, the user cannot edit that block in the Review screen. They won’t even see the Edit block that you made with that variable in it.

The following tips may help you get around this issue.

- If you asked a question in the interview and there’s a “show if” statement in that question block, use this in that particular Edit block:
  showifdef(“variable_name”)
- If you have asked a question in the interview where you have a variable that is a string (a text field), and there are no “show if” statements on that screen, use this in that particular Edit block:
  % if variable_name ! = ' ':
  The exclamation point means “not” so this is saying “if variable_name is not empty”.
- If you want to let someone edit variables from different question blocks in the interview, but have them all in the same Edit block, use this in that particular Edit block before mentioning the variable:
  if defined(“variable_name”):

Financial forms and the review screen

If your interview is financial in nature, consider not putting all the dollar amounts on the review screen. Instead, just list the names of the public benefits, expenses, jobs, vehicles, real estate, etc. It will keep your review screen simple, clean, shorter and a bit easier to code. The user can click Edit to dive into the numbers if they want to edit.

Use headers to break up information on a long review screen

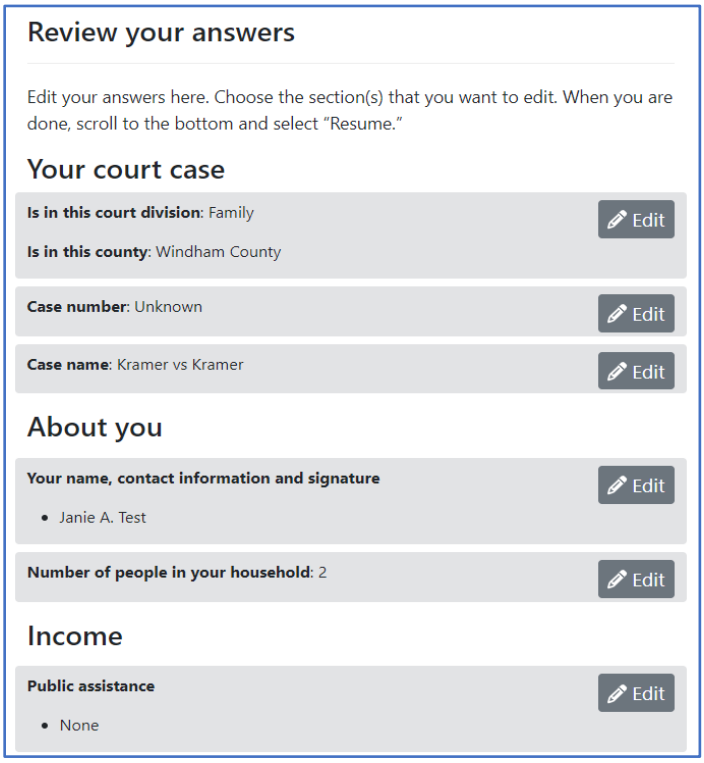
Using headers can make the page easier to look at and use.

Do this by adding notes in your review screen code.

```
- note: |
    ###About you

- Edit: users.revisit
  button: |
    **Your name, contact information
and signature**

    % for item in users:
    * ${ item }
    % endfor
```



recompute

Use recompute in the review screen to recompute certain values or variables in "code blocks" after a user makes a change, and to get it reflected in the review screen and in the attachment PDF or Word doc.

Example: Use recompute to allow the user to change the county their court is in, and get the new court to show up on the PDF.

```
- Edit:
  - user_selected_county
  - recompute:
    - trial_court_index
    - trial_court
```

invalidate

Sometimes you have a question where if you answer A, it will collect extra details. But in the review screen, you want the user to be able to change their answer to B or C and “erase” the old details that were collected.

Invalidate will undefine the variables. But it will also remember the variables as a default if you do have to answer the question again.

The order is important. The invalidate needs to be before any “recompute” in the code block.

Here’s an example where invalidate is used to erase some variable values when a person changes their answer about their payment. Otherwise, those variables may show up on the document when they shouldn’t.

```
- Edit:
  - payment
  - invalidate:
    - agrees_to_pay_installments_amount
    - agrees_to_pay_installments_period
    - agrees_to_pay_installments_start_date
```

confirm: True

Using confirm: True in a review table will ask the user if they really mean to delete something. If they say yes, it will be deleted.

Documentation:

- https://assemblyline.suffolklitlab.org/docs/authoring/review_screen#step-2-understanding-and-editing-the-generated-review-screen
- https://docassemble.org/docs/fields.html#review

Updating a PDF template

If you need to make a small edit to your PDF:

- Edit the PDF in Adobe Acrobat.
- Don’t rename it if you want to just overwrite the file that is in the system.
- In the Playground, click on Folders and choose Templates.
- Upload your updated file.

However, sometimes laws change and/or the courts make changes to a form and you need to **incorporate the new PDF template** in your interview. Luckily, you can still

make use of the work you did on the original form to make the new PDF work with your interview.

- Go to the Assembly Line Dashboard https://apps-dev.suffolklitlab.org/interview?i=docassemble.ALDashboard:data/questions/menu.yml#page1
- Click on PDF tools.
- Drag and drop the PDF that you created with the preferred fields and field names.
- Drag and drop the new PDF form.
- The tool will produce a new PDF that has pasted in your fields and field names. You will just need to clean it up by moving fields that may have moved on the page, deleting fields and/or adding fields.

Documentation: https://assemblyline.suffolklitlab.org/docs/authoring/pdfs

DOCX templates

Microsoft DOCX files are often used as an addendum to print out overflow information (like letting people enter 6 child names when the form only has space for 4). DOCX files are often used instead of PDFs as main templates, too.

You will use special coding called “Jinja2” to print things in a DOCX template. Jinja plays the same role inside Microsoft Word DOCX templates that Mako plays inside the interview file. Jinja statements allow you to insert variables, use conditional text and more in a DOCX file.

Documentation:

- https://assemblyline.suffolklitlab.org/docs/authoring/docx
- https://assemblyline.suffolklitlab.org/docs/docassemble_intro/jinja2
- https://help.dashly.io/article/5146

User testing

User testing is a must. With any interview you create, be sure to have multiple content experts review it. Then be sure to have folks from the general public test it. You can do it in person, or online via Zoom. Every time you watch someone use the tool, you will gain ideas on how to make it clearer or easier to use! We have information on user testing protocols in our Legal Services Vermont TIG22 final report.

Translating an interview

When your interview is **completely finished and few changes are expected**, you can consider translating your guided interview.

Go to the Dashboard. Click on Prepare translation files. Choose the yml file for your desired interview. Then type in the two-letter codes for the languages you want to translate to.

You will get an xlsx spreadsheet for each of your desired languages. You will be able to give the spreadsheet to a translation company to translate many of the words. You will need to explain to them that they should not translate the code, but only the text.

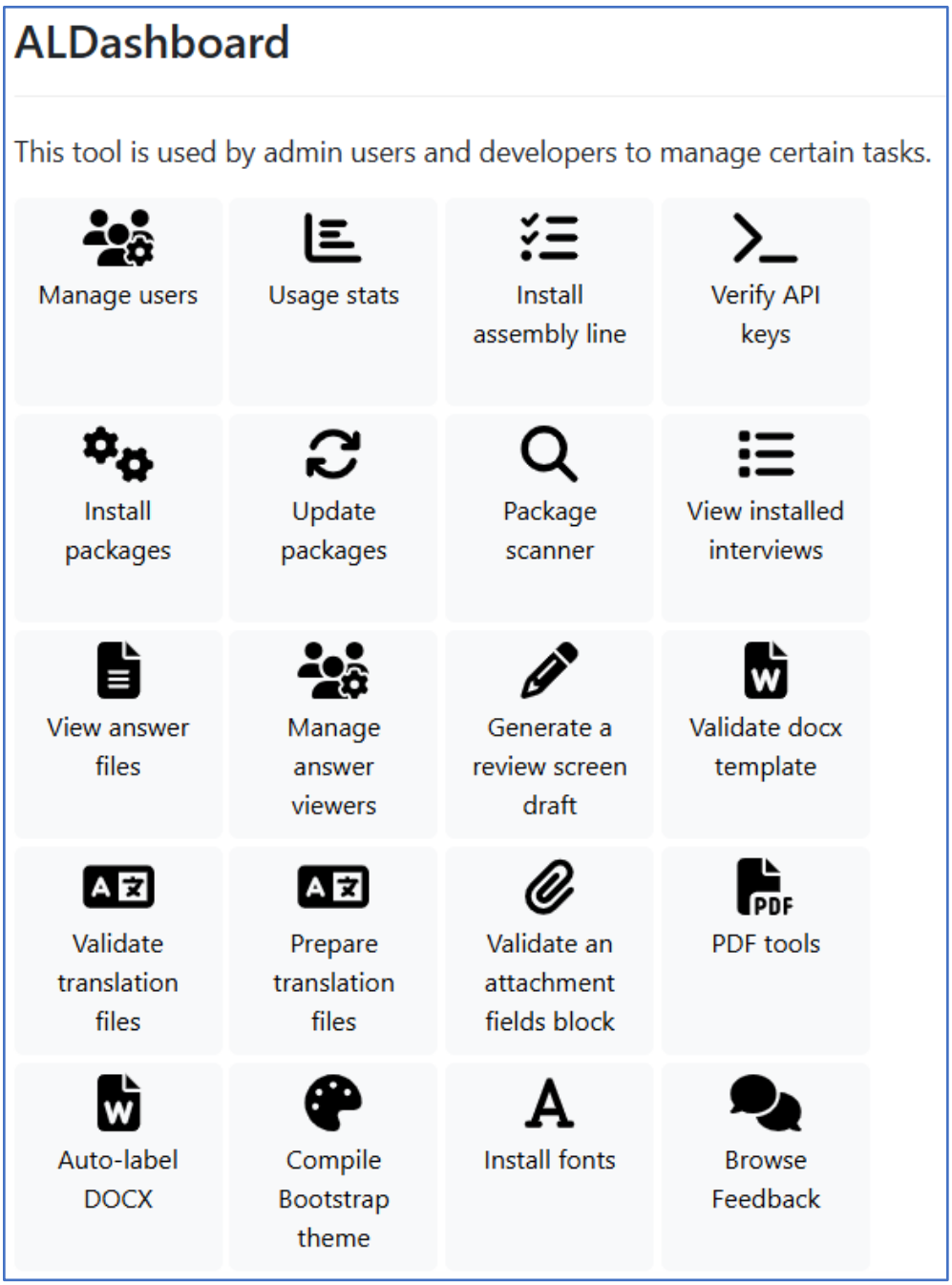
Later you will add the spreadsheet into the “sources” folder and add code to your interview to allow users to choose a language.

Translation works off of an exact match. When you change the original language of a question, the translation will no longer be valid. Even changes to punctuation and white space will cause Docassemble to ignore the translation and show the text in its original language.

Documentation:
https://assemblyline.suffolklitlab.org/docs/components/AssemblyLine/translation

Tools in the ALDashboard

ALDashboard is a collection of tools that an Admin can access when they are in the Playground. Documentation:
https://assemblyline.suffolklitlab.org/docs/components/ALDashboard/overview



Statistics/usage reports

Earlier we talked about snapshot code to collect anonymous user info. To see the statistics, go to the AL Dashboard and “Usage Stats.”

Documentation:
https://assemblyline.suffolklitlab.org/docs/components/InterviewStats/overview

Come up with a consistent style for your interviews

You may need to work with someone who knows CSS or SCSS to update the look of your guided interviews so they are distinct from the drafts that the Assembly Line produces.

Use the same logos, icons and intro page(s) for all of your interviews. This way, your users will get familiar with using them. Be sure the user knows who created the interviews so they can feel comfortable using them.

You can create a CSS file that sets some special styles. You could put that CSS in a shared yml file to be included in all your guided interviews. You can also customize screen parts in your interviews.

Documentation:

- https://assemblyline.suffolklitlab.org/docs/components/ALThemeTemplate/overview
- https://docassemble.org/docs/questions.html#screen%20parts

Go-live checklist for your first Docassemble Assembly Line interview

These are things to do prior to launch – some are optional. We made this list with the help of our Suffolk LIT Lab cohort so other people could refer to it. LSV has **not** done all of these things for VTCourtForms.

Note: You'll notice that some customizations and settings are made in the configuration screen of your Docassemble server (YourURL/config), while others are made by copying some default code, editing it, and placing it in your interview to override some default code.

You should do these:

1. Set up and test the email service for your server (so people can email the forms to themselves or others).
2. Use free tools like WriteClearly and Docassemble's built-in developer stats to test for page readability (aim for 6th grade level), and test your launch page and your interview's color scheme, etc., for accessibility (use WebAIM's Wave tool).
3. Do some user testing.
4. Do thorough testing of the interview, including printing, emailing and downloading the forms. Don't forget to test on smartphones.
5. Make sure the user account sign-up works.
6. Make sure the user account log-in works.
7. Make sure Save answer sets works (if you allow users to do so).

8. Decide if you will allow users to update their passwords.
9. Make sure Exit and delete my answers works. In the configuration screen of your Docassemble server (YourURL/config), you can customize the website that the user is sent to (exitpage: https://google.com, for example).
10. Create data retention policies and publish a terms and conditions page. Link to it from the initial screen of your interview. In your configuration screen, set interview delete days: xxx to the number of days of inactivity before you want interviews deleted.
11. Make sure you are notified when users get "bug" errors in your interviews. In your configuration screen, set error notification email: YourEmailAddress.
12. Set up UptimeRobot (free) – or similar – to alert you if your server is down.
13. Know how to turn off your interview if you need to due to a problem. Have an "under maintenance" message ready.

    - On launch page on your legal help website: Edit the page to say the interview is under maintenance. Also disable the link to the interview.
    - In the code of the interview: Add a mandatory code block that kicks the user to a screen with your under-maintenance message. Note: No one will be able to access the interview – even folks with accounts and stored answers.

14. Make sure your Feedback system works and you get alerted to the Feedback that people submit and you can see it (many admins send the Feedback to a separate, private GitHub repository).
15. Put your Google Analytics account info into the configuration screen of your server so you can track traffic.
16. Make sure your launch page is live (probably on your legal help website) and it links to your interview correctly.
17. Turn off the red debug text at the top of your interviews in your configuration screen (debug: false).

You may want to do these:

18. Use Google address autocomplete.
19. See if you need to customize the Share feature screen. For example, if you aren't offering texting through your server, you need to delete the text that says to enter a phone number.
20. If you want to be able to use a shorter version of your URL for your interview (such as https://YourDomain.org/start/YourShortName/), use the dispatch setting in your configuration screen. For example, if you put the following dispatch text in your configuration, you can use these URLs to run your interview -- https://apps.vtcourtforms.org/start/rfa/ OR https://apps.vtcourtforms.org/run/rfa/

```
dispatch:
  rfa: docassemble.RFApackage:data/questions/RFApackage.yml
```

Note: From your launch page on your legal help website (or wherever it is), there is a difference for the user when you use run or start in your link. If you offer the /start/ URL, it will always start a new session (which may be good for security) when someone clicks on that link. If you offer the /run/ URL, a person using the same computer can go back to the form they were working on (which could be a problem for security.) See https://docassemble.org/docs/helloworld.html#packaging.

21. Decide what kind of anonymous stats you want to collect. If you want to collect basic stats of how many times your interview is used, the user's zip code, and other details from your interview, and then view them on your server, upload this package to your server: https://github.com/SuffolkLITLab/docassemble-InterviewStats. In each interview you want stats for, be sure you have this in your code: store_variables_snapshot. In your configuration screen, set collect statistics: true.
22. Get familiar with the "back-end" of your server and the Assembly Line Dashboard that is installed there so you can see a list of interview users, list of feedback submissions, etc.
23. Create tests to automatically check that your server is up and can load interviews.
24. Set up ALKiln to do automated testing of the interview on your development server – **not** the server that you use for your live interviews. This will take some time. Do it only when your interview is very finished and there will be no changes to names of variables. Using ALKiln will make it easier to test your interview from time to time and especially after you make small changes in the future. (You can commit the new testing files to GitHub and your live server, but you should not run your tests on your live server because it can bog things down and/or cause trouble.)

These would be nice to do some time, but are optional:

25. Optional: SMS integration. If you want people to be able to text someone to Share their answers and progress, you will need an SMS provider set up in your Docassemble configuration.Docassemble supports Twilio. Users can also Share by email, so SMS is not essential.
26. Optional: Apply your branding (organizational logo, colors, etc.) to your interviews. You'd create a special .css file or package and use it with your interview(s).
27. Optional: Also apply your branding to the system-generated pages like:

    - The interview_list page (when logged in, the user can see a list of the interviews they've been doing)
    - Account creation page
    - Account sign-in page
    - The /list page which shows the selection of interviews offered on your server (it's set with the dispatch setting in your configuration screen)

- If you're worried about enforcing your branding, you might want to add "auto color scheme: False" to your configuration screen. That way if a user has dark mode turned on, they will still see your color choices.

28. Optional: Customize the Feedback pages (there are two versions: one for testers and one for general public) by making your own custom version of feedback.yml. Then in your interview add this code: feedback_form = "docassemble.YourFeedbackPackageName:YourFilename.yml"
29. Optional: Customize the bug-catcher page. Assembly Line has a customized error page for interviews. Users see it when your interview fails in some way. You can customize it further. One easy update is to add code to your interview to add specific text and links to show on that error screen – to help the user who may be stuck and needs to download what they've done so far, to get the paper versions of the forms, etc. Make an al_custom_error_options code block. (These errors are rare -- you fixed most bugs during your own testing – and this can always be customized later.)
30. Optional: After ILAO made those config adjustments, they made a new id: custom error action question and template: al_formatted_error_message that aligned with their feedback email. (You can put this in your own organization's framework package, likely in a branded basic_questions.yml. You can see these in the ILAO package here.
31. Optional: Customize the 404 (Page not found) page. Add a link and info to help a wayward user. For example, in your configuration screen, add something like this:

    404 page title: Page Not Found
    404 page heading: Page Not Found
    404 page pre: We are sorry. The URL you entered was not found on the VTCourtForms site. Try hitting the back button, or visit this link to find a VTCourtForms interview.

32. Optional: Consider customizing the About this Form page, which is normally accessed from the footer. Copy template: about_this_interview_version_info and paste it in your interview. Then edit the template. It will override the default text.
33. Optional: Consider customizing and increasing the font size of the emails that are generated from your interview – such as the email that sends attached documents. You would update this. Use HTML in the content area to increase the font size.
34. Optional: Add a favicon for the website that hosts your interviews. The default favicon is the Docassemble cat. Meow.

For other details on the creation of our first interview packages, visit our Legal Services Vermont TIG22 final report, filed in December 2025.

Last updated: January 16, 2025