Trellis

User Guide

Version 1.0.0-beta.2 · October 2026

Requirements authoring for life and annuity new business

Table of Contents

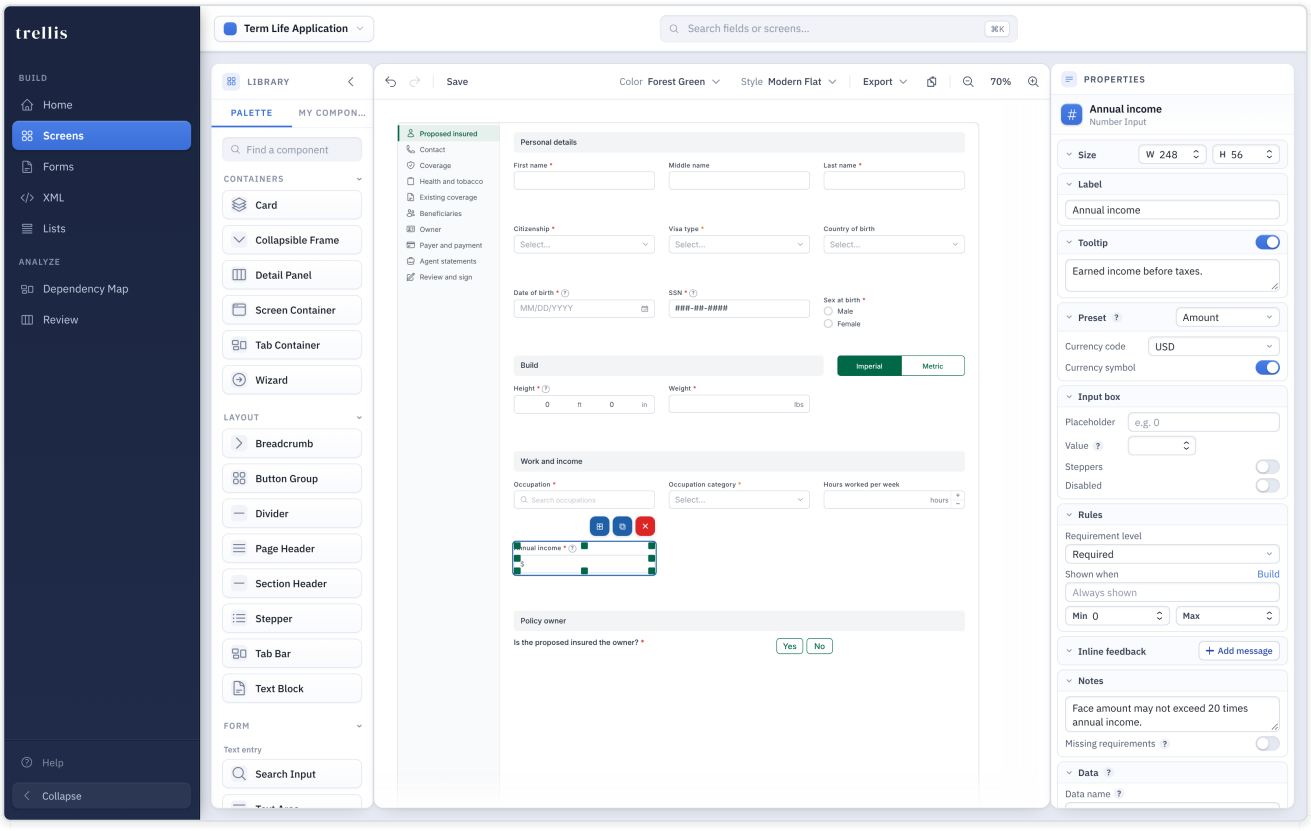
1. Welcome to Trellis
2. Finding Your Way Around
3. Getting Started
4. The Canvas
5. Components
6. Containers and Nesting
7. The Properties Panel
8. Data, Lists, Forms and XML
9. Conditions
10. Themes
11. Saving and Loading
12. Component Packs
13. Review
14. Exports
15. The Dependency Map
16. Keyboard Shortcuts
17. Known Limitations

1. Welcome to Trellis

Trellis is a browser-based requirements authoring tool for life and annuity insurance applications. Business analysts lay out screens from configurable components, set each field's properties and rules, write the conditions that decide when each is required, shown or offered, record the data each field collects and where that data goes (the XML values it is sent as and the forms it prints on), and export structured deliverables for developer handoff, QA and BA documentation.

The screens are high-fidelity mockups, not a working application. What makes them useful is that everything about them lives in one project: move a field, rename it or change its rules, and the mockup, the Rules Matrix, the exports and the Dependency Map all reflect the change.

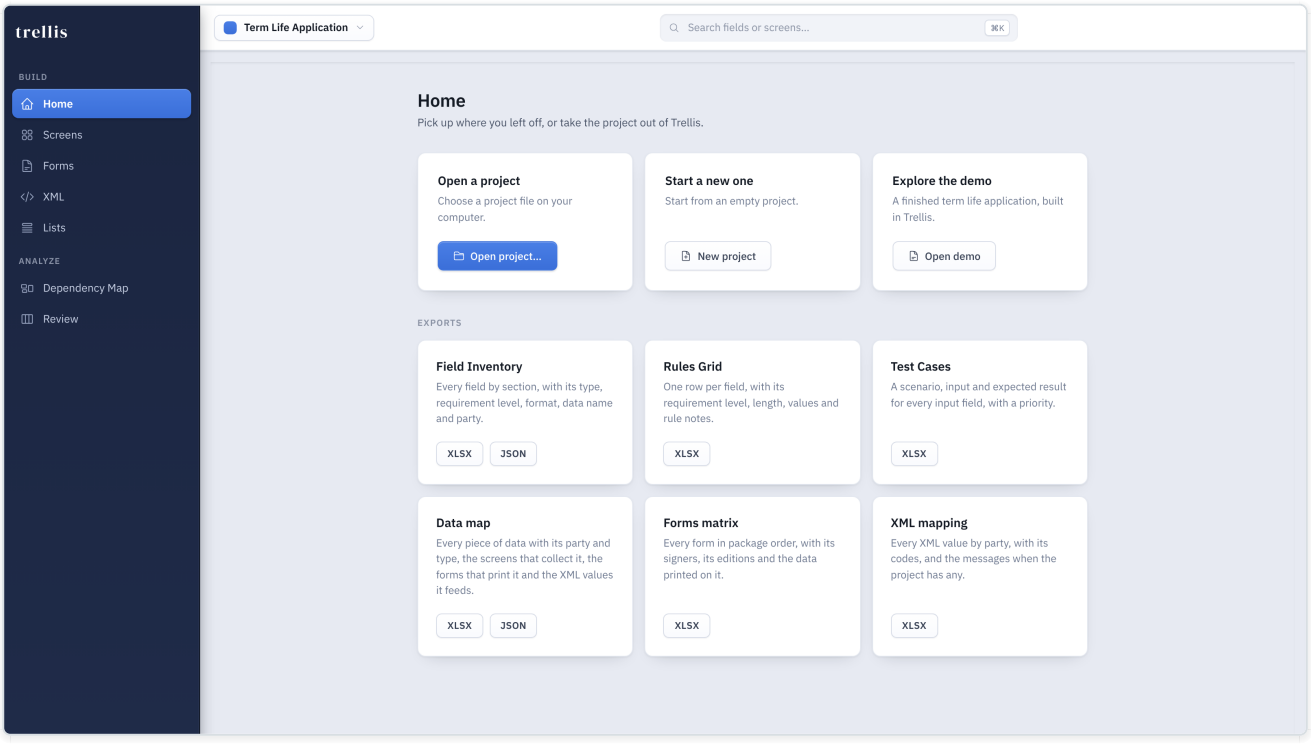
Trellis runs entirely in your browser with no backend, no login, and no external dependencies. Your work saves locally and to `.trellis` files on your computer, and exports to standard formats (PNG, Excel, JSON) that fit into existing workflows.



The Screens view with the demo project open: the library on the left, the screen in the middle, and the Properties Panel for the selected Annual income field on the right.

2. Finding Your Way Around

Trellis is laid out as a rail of views down the left side, a top bar across the top, and the view you are working in.



Home, the project's front page. The rail is on the left and the top bar above.

The Rail

The rail lists the views in two groups:

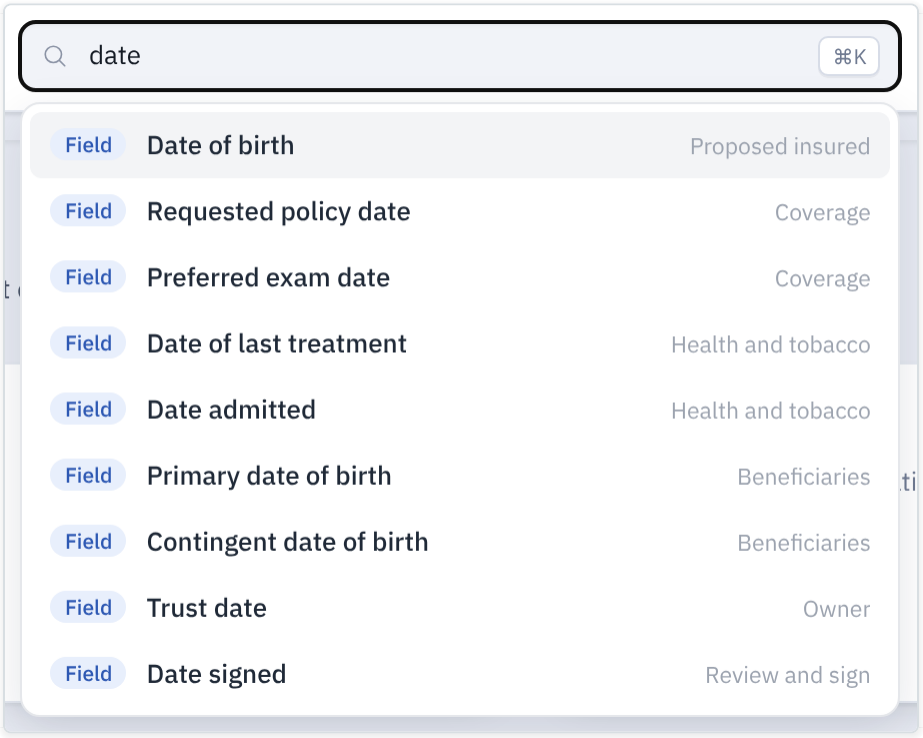
- **Build: Home**, the project's front page; **Screens**, where you design screens on the canvas; **Forms**, the paper and PDF forms the project prints; **XML**, the XML values it sends; and **Lists**, the lists the project picks from.
- **Analyze: Dependency Map**, which shows how everything in the project connects; and **Review**, which puts the Rules Matrix beside the canvas.

At the foot of the rail, **Help** opens About, with the app's version and buttons that open this guide and the Security & Privacy document. **Collapse** narrows the rail to icons (hover over one to see its name), and **Expand** brings the labels back. Trellis remembers the rail's width in this browser.

The Top Bar

The left of the top bar shows the project's name: its file's name without the `.trellis` extension, **Untitled** for a project with no file yet, or **Term Life Application** for the demo. On Firefox and Safari, which open a copy of a file rather than the file itself, an opened project reads Untitled. Click the name to open the project menu: **Open…**, **New** and **Save as…** (section 11).

In the middle is the search. It finds fields by their labels and screens by their names, including tabs, side-navigation screens and wizard steps. Press Cmd+K (Mac) or Ctrl+K (Windows) to jump to it, type part of a name, and pick a result with the mouse or with the arrow keys and Enter. Opening a result switches to Screens, opens whatever was hiding it (a tab, screen or step, a collapsed frame, a parked Detail Panel or a hidden Modal Dialog), and selects the field (or, for a screen, its container). If anything had to open, one undo puts the screen back the way it was.



Searching for "date". Each result says whether it is a field or a screen, and where it is.

Home

Home is the project's front page. Its first row gets you into a project:

- **Open a project: Open project…** opens a `.trellis` file from your computer.
- **Start a new one: New project** starts an empty project and opens it on Screens.
- **Explore the demo: Open demo** opens a finished ten-screen term life application (section 3).

Below them, **Exports** has one card per project export, three to a row, with a button for each format: the Field Inventory (XLSX and JSON), the Rules Grid (XLSX), the Test Cases (XLSX), the Data map (XLSX and JSON), the Forms matrix (XLSX) and the XML mapping (XLSX). Section 14 describes each.

When there is nothing in this browser yet, as on your first visit, Open demo is the highlighted button; once there is a project, Open project… is.

The Views

- **Screens** is the canvas, with the library on its left and the Properties Panel on its right (sections 3 to 7).
- **Forms** and **XML** are the project's forms and XML values, each a table with tabs, edited in place (section 8).
- **Lists** keeps every list the project picks from: its parties, products, form types and messages, its saved conditions, the lists built into Trellis, and what decides a form's products, languages and provinces or states (sections 8 and 9).
- **Dependency Map** shows what uses any screen, field, piece of data, XML value or form, what it feeds and what decides it (section 15).
- **Review** puts the Rules Matrix beside the canvas, for checking a screen's rules with a client (section 13).

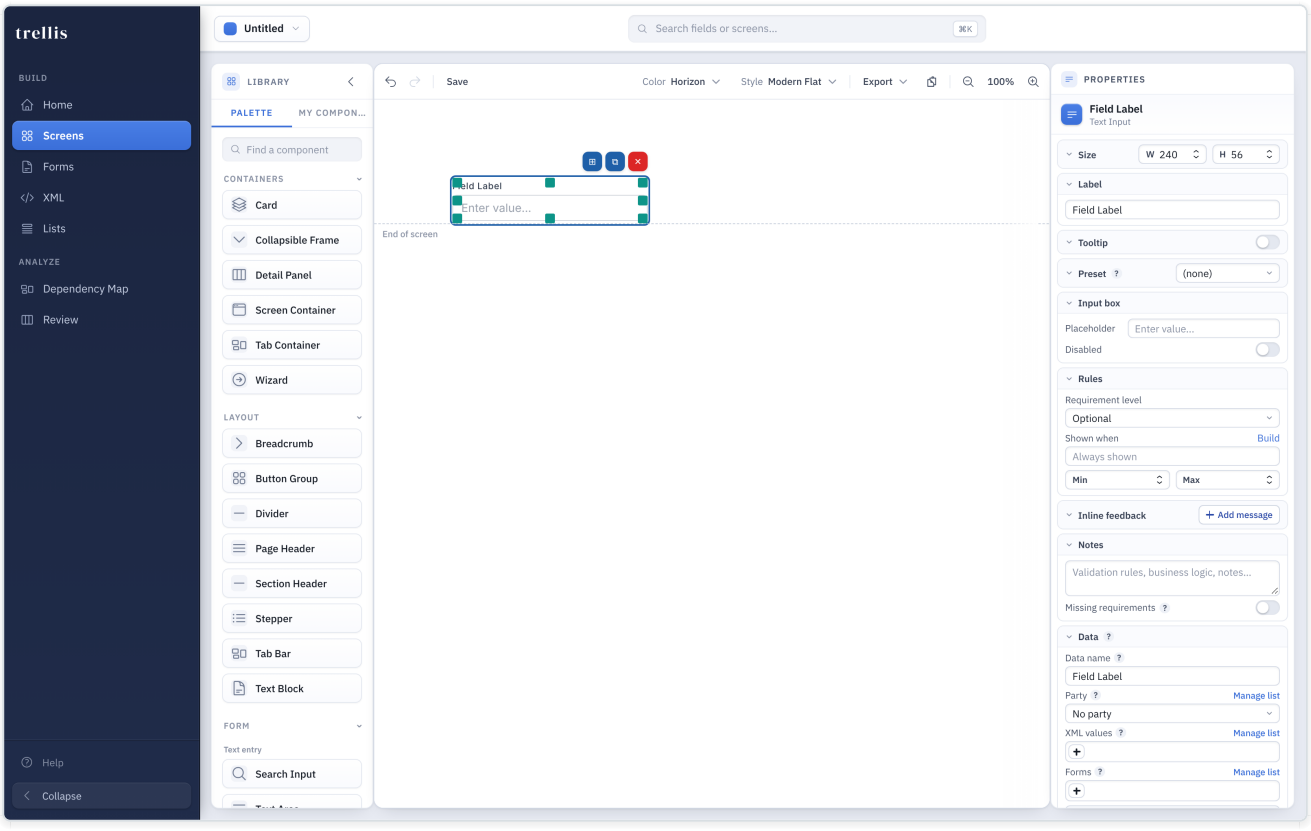
Switching views never changes your project, and undo and redo work on every view.

3. Getting Started

This section gets you from an empty project to your first exported deliverable in a few minutes.

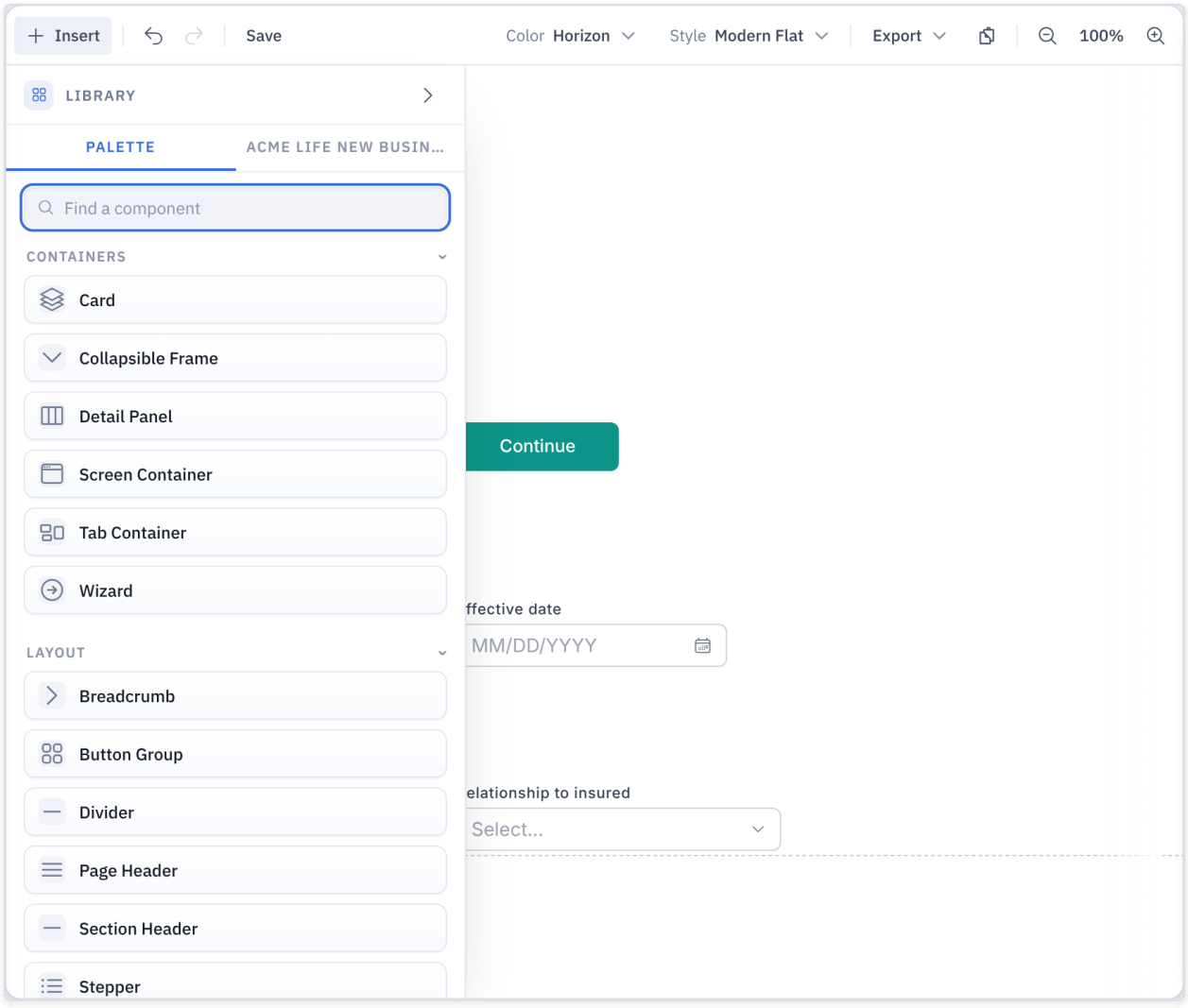
The first time you open Trellis, it opens on Home. **New project** starts a blank project and opens its canvas on Screens, where these steps begin. **Open demo** opens a finished term life application on Screens as an unsaved copy you can explore and change; the first time you save it, Trellis asks where to put the file on Chrome and Edge, and downloads it on Firefox and Safari. The demo is also at trellismade.com/demo. On a laptop-size window it opens with the library tucked away and zoomed to fit its screen's width; your own library setting is left as it was. If the project in this browser has changes that no file holds, Trellis asks before replacing it: **Save first**, **Open demo** or **Cancel**.

Step 1: Drag a component from the library. The library on the left of the canvas lists every component by category. Drag any component tile onto the canvas. To find one by name, type in **Find a component** at the top of the library; click a category's heading to fold it away.



A Text Input dropped on a new project's canvas and selected. The small toolbar above it saves it to a pack, duplicates it or deletes it; the Properties Panel on the right edits it.

The library can be tucked away to give the canvas more room: click the chevron at the right end of its header (Collapse library). The toolbar then starts with **+ Insert**, which opens the library over the canvas; drag a component from it and it closes as you start, so you can drop anywhere, including where it was. Press Escape or click elsewhere to close it, and use the chevron in its header (Dock library) to put it back beside the canvas. Trellis remembers the choice in this browser.



The library tucked away: + Insert on the toolbar opens it over the canvas.

Step 2: Position and configure. Drop the component where you want it. Click it to select it, then use the Properties Panel on the right to set its label, requirement level, options and other properties (section 7). For specialized field types (like SSN, Currency or Gender), place a root component (Text Input, Number Input, Dropdown or Radio Group) and choose a preset in the Properties Panel.

Step 3: Build your screens. Use a Screen Container, or a Tab Container with side navigation, as the outer structure for the application's screens (Proposed insured, Coverage, Beneficiaries and so on). Inside each screen, group related fields with Collapsible Frames and Section Headers.

Step 4: Name the data. Each input field has a Data section in the Properties Panel: name what the field collects, say whose information it is, and pick the XML values it is sent as and the forms it prints on (section 8).

Step 5: Review and export. Review in the rail shows the screen's Rules Matrix beside it (section 13). Home's export cards download the Field Inventory, Rules Grid, Test Cases, Data map, Forms matrix and XML mapping; the Export menu on the canvas toolbar downloads an image of the screen, or the Rules Grid or Test Cases for the whole project or just the screen you are on (section 14).

Step 6: Save. Click **Save** on the canvas toolbar. The first save asks where to put the `.trellis` file (section 11).

> **Tip:** Real eApp screens almost always follow the same outer-to-inner pattern: a Screen Container (or a Tab Container with side navigation) at the top, screens within it, Collapsible Frames and Section Headers within each screen, and fields within each frame. Starting with one of these containers puts you on the right structural path from the first drag.

4. The Canvas

The canvas is your workspace on Screens. This section covers how it behaves and how to work with components once they are placed.

The Canvas Surface

The canvas is 1440 pixels wide, the width of the screen you are designing, and as tall as what is on it. A dashed line captioned "End of screen" marks the bottom of the lowest component, with room below it to place the next; an empty project is a bare sheet. The canvas fills the space between the library and the Properties Panel and scrolls with its own scrollbars, and an edge the screen continues past fades out.

Zoom

The zoom controls sit at the right end of the canvas toolbar: zoom out, the current zoom, and zoom in. 100% is real size. The lowest zoom, **Fit**, shows the screen's full 1440-pixel width across the canvas (45% on a 1440-pixel window with the rail open), and the readout says Fit there. Zoom in and out move in steps of 10%, up to 200%. Click the readout at any zoom to fit the screen again; its tooltip gives the Fit's own percentage. Zooming keeps whatever is in the middle of the view where it is.

Opening or starting a project shows it at real size, at the top left; on a window so wide that the screen's full width fits with room to spare, it starts at Fit instead, which is then larger than real size. Switching tabs, screens or steps scrolls back to the top.

Selecting Components

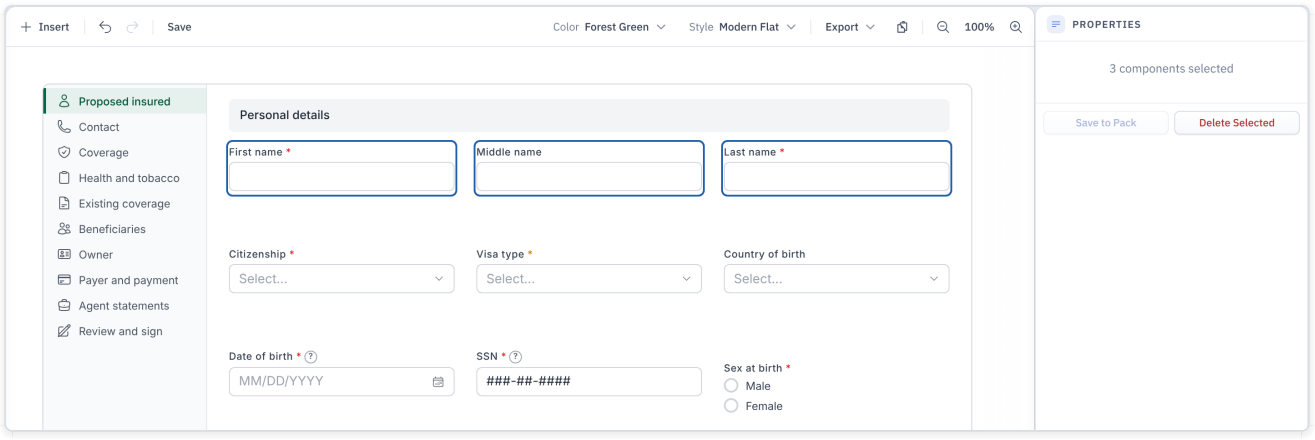
Click a component to select it. The selection shows a blue outline, resize handles, and a small toolbar above the component with three buttons: save to pack (⊞), duplicate (⧉) and delete (✕).

- **Multi-select:** Hold Ctrl (Windows) or Cmd (Mac) and click additional components to add them to the selection, or click a selected one to take it out.

- **Marquee select:** Click and drag on an empty area of the canvas to draw a selection rectangle. Any top-level component that intersects the rectangle is selected. Hold Shift, Ctrl or Cmd as you start to add to the selection instead.
- **Clear selection:** Press Escape or click an empty area of the canvas.

All selected components must share the same parent: Ctrl- or Cmd-clicking a component in another container starts a new selection. Marquee selection works at the canvas root level only.

> **Note:** Selection automatically clears when you switch tabs, screens or steps, or when a Collapsible Frame collapses.



Three name fields selected together. The Properties Panel counts them and offers Delete Selected.

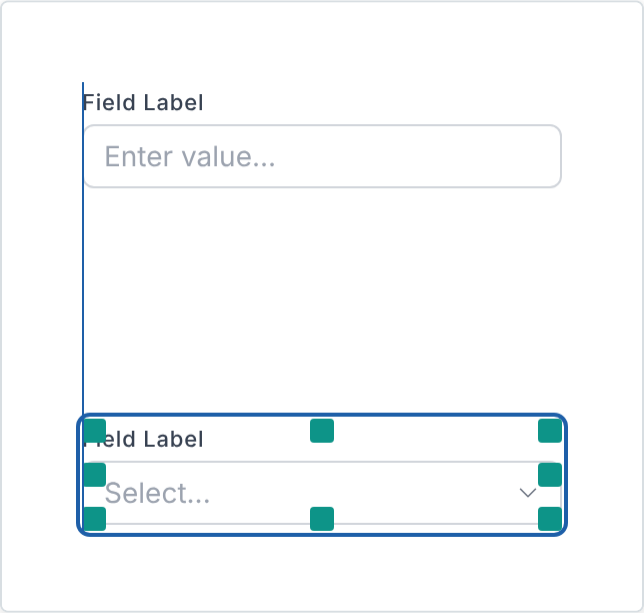
Moving Components

Drag any selected component to reposition it. Positions snap to an 8-pixel grid. When a component's edge comes within 8 pixels of a sibling's edge, it snaps to it and a guide line appears.

When multiple components are selected, dragging any one moves the entire group as a rigid unit, preserving their relative spacing.

Arrow key nudge: Press arrow keys to move selected components by 1 pixel. Hold Shift for 8-pixel steps.

A component stays inside the container it was dropped in. To move it to another container, copy it and paste it there (section 17).



An alignment guide: the Dropdown being dragged lines up with the left edge of the Text Input above it.

Resizing Components

Drag a handle on a selected component's corners or edges to resize it. There are eight, one at each corner and one in the middle of each edge, but an edge too short for three handles (under 48 pixels) shows only its corners: a button shows six, and a small icon only its four corners. Resize snaps to the 8-pixel grid. The handles show for a single selection only, and not on a collapsed Collapsible Frame. A Detail Panel shows one handle, on its left edge, which sets its width.

You can also resize from the Properties Panel by typing exact values into the W and H boxes of its Size row, or by clicking the up/down arrows beside each. This is often smoother than dragging when you want a precise size or fine-grained adjustments.

Containers cannot be shrunk smaller than their contents require. Other components have a minimum size of 40 x 24 pixels.

Duplicating Components

Click the duplicate button in the floating toolbar above a selected component. If the component is a container, all of its children are duplicated with it. Duplicate is available for single-selected components only. A duplicated Detail Panel opens and its original parks, and a duplicated Modal Dialog shows while its original hides.

Deleting Components

Click the delete button (✕) in the floating toolbar, or **Delete Component** at the bottom of the Properties Panel.

Deleting an empty container or any other component happens immediately. Deleting a non-empty container shows a confirmation dialog with the count of child components that will also be removed. With more than one component selected, the button at the bottom of the Properties Panel reads **Delete Selected** and deletes them all in one action: if any of them is a non-empty container, one confirmation dialog asks first, with the count of what they contain, and one undo brings the whole group back, still selected. Deleting a field that a condition reads is never refused: the condition is kept and marked broken, and the toast says how many (section 9).

Copy and Paste

Copy selected components with Cmd+C (Mac) or Ctrl+C (Windows). Paste with Cmd+V or Ctrl+V.

- **Same-project paste:** Components appear 16 pixels below and to the right of the original. Trellis knows a project as the same one once it has been saved; until its first save (a new project, or the demo before you save it), a paste lands centered in your view instead.
- **Cross-project paste:** Components appear centered in your current view. Copy in one project, open the other and paste: a field brings its data with it (section 8).
- **Paste into a container:** If you have a container selected when you paste, components nest inside it on the active tab, screen or step. With nothing selected, a paste goes on the canvas itself, even when what you copied came from inside a container.
- **Nesting rules apply:** If the pasted component type is not allowed in the target location, a message explains why and nothing is pasted.

Copy and paste supports single components, multi-selections, and containers with all their children. What you copy keeps its conditions. In another project, a condition that reads something outside what you copied is left out, and the toast counts them; a saved condition that project lacks arrives as the slot's own copy of its rules (section 9).

Undo and Redo

Undo your last action with Cmd+Z (Mac) or Ctrl+Z (Windows), or the undo button at the left of the canvas toolbar. Redo with Cmd+Shift+Z or Cmd+Y (Ctrl+Shift+Z or Ctrl+Y on Windows), or the redo button. Undo and redo work on every view, not only on the canvas.

Trellis keeps up to 50 undo steps. The history resets when you open a file, start a new project or reload the page. Rapid property edits (like typing in a text field) are coalesced into a single undo step so you do not have to undo character by character.

> **Tip:** Undo works for every change to what the project holds: moving, resizing, property edits, deleting, pasting, theme changes, conditions built in the builder, and edits in the Forms, XML and Lists views. If you accidentally delete a component, Cmd+Z brings it back immediately. The one change it does not take back is choosing a different pack in the library's pack tab.

5. Components

Trellis includes 49 components across seven categories. This section introduces each category and when to reach for it. For the details of any component's settings, select it and look through the Properties Panel (section 7).

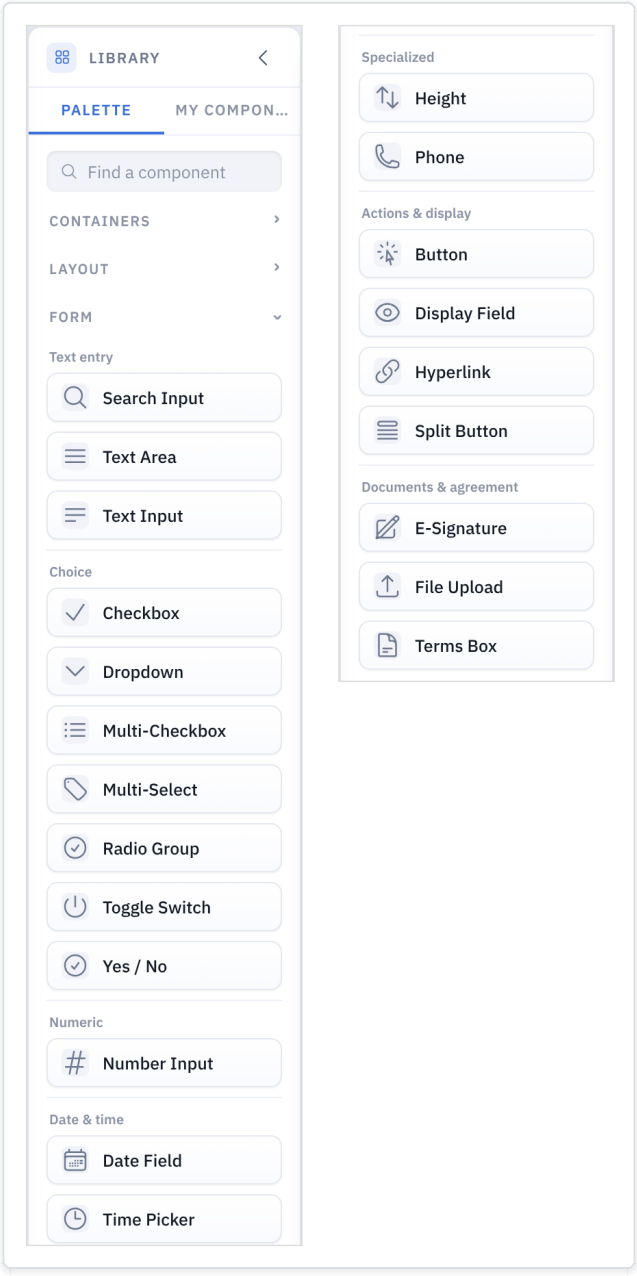
Form (22 components)

Form components are the core of your mockups: the fields someone fills in on a new business application, and the buttons and documents around them. The library organizes them into seven sub-groups:

- **Text entry:** Text Input, Text Area, Search Input. Use for free-text fields like names, addresses, and notes.
- **Choice:** Checkbox, Dropdown, Multi-Checkbox, Multi-Select, Radio Group, Toggle Switch, Yes / No. Use for any field where the user picks from a set of options; Multi-Checkbox lists every option with a checkbox, and Multi-Select is a dropdown that shows the picks as chips. Options are fully editable in the Properties Panel.
- **Numeric:** Number Input. Use for any numeric value (coverage amounts, percentages, ages). Presets configure it for specific financial and measurement formats.
- **Date & time:** Date Field and Time Picker. Date Field takes an optional direction rule (past only, future only); Time Picker takes earliest and latest times, a time format and a minute step.
- **Specialized:** Height and Phone. These two fields have unique multi-zone layouts (Phone has a type selector and extension zone; Height has a metric/imperial toggle with dual inputs) and appear as standalone tiles in the library.
- **Actions & display:** Button, Display Field, Hyperlink, Split Button. Use Button for form actions (submit, cancel, next). Use Display Field for read-only computed or reference values; it can hold several value rows under one label (add, remove, and reorder them in the Properties Panel) and its **Value side** setting places the rows below the label or to its right. Hyperlink is a line of hyperlink text, with a URL and a target that the Field Inventory carries. Split Button combines a primary action with a dropdown of secondary actions (for example, "Submit" with "Save as draft" and "Save and continue" in the menu).

- **Documents & agreement:** E-Signature, File Upload, Terms Box. Use for document collection and consent capture sections.

Many specialized field types (like SSN, Currency, Gender, and Country) are not separate library tiles. Instead, they are configured as presets on the root components listed above. See the Presets subsection below.



The Form category in the library, set in two columns here, showing all seven sub-groups: Text entry, Choice, Numeric, Date & time, Specialized, Actions & display, and Documents & agreement.

Presets

Presets let you turn a generic root component into a specialized field type. When you select a Text Input, Number Input, Dropdown, or Radio Group on the canvas, the Properties Panel has a **Preset** section. Choosing a preset automatically applies the correct label, placeholder or format, industry-standard default options (where applicable), and export metadata. The Products preset is the one that keeps the question's own label.

For example, selecting a Text Input and choosing the "Social Security Number" preset gives you a masked SSN field with the XXX-XX-XXXX placeholder, the correct export type, and format string. Selecting a Dropdown and choosing the "Country" preset populates it with a curated 13-country list.

Switching from one preset to another resets the settings that belong to the old one, and a message says so. To remove a preset and revert to a plain root component, select "(none)" from the Preset dropdown. This clears all preset-specific configuration and restores the root component defaults.

Available presets by root type:

Text Input (9 presets): Email Address, Social Security Number, Social Insurance Number, Tax ID Number, Postal Code, Institution Number, Transit Number, Account Number, Routing Number. These cover contact fields, government IDs, masked inputs, and banking fields.

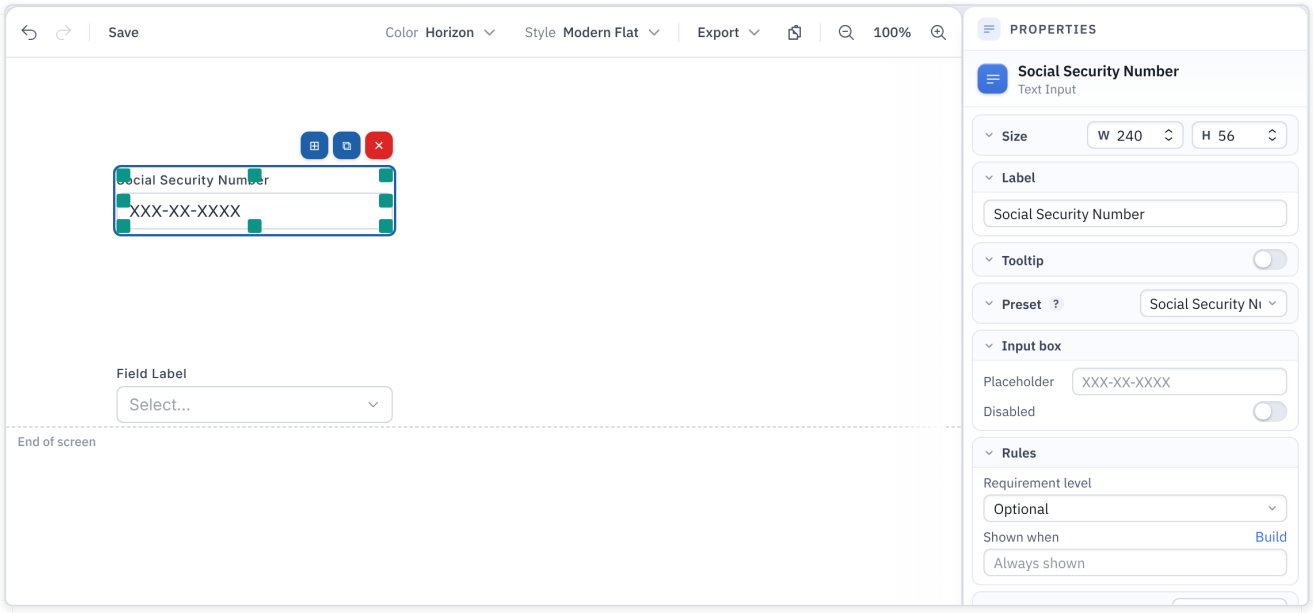
Number Input (5 presets): Amount (currency), Percentage, Age, Duration, Weight. These configure prefix/suffix formatting, unit toggles, and validation bounds appropriate for each field type.

Dropdown (7 presets): Country, State / Province, Payment Frequency, Language, Occupation, Gender, and, in a group of its own, **This project**: Products. The first six pre-load industry-standard option lists. State / Province is country-aware (switching its country resets the option list automatically).

Radio Group (2 presets): Gender, which renders the same standard gender options as the Dropdown preset, but as radio buttons instead of a dropdown; and Products, under This project.

Products shows the project's own Products list, kept on the Lists page (section 8). While the field's options are unedited it follows the list: a product added, renamed or deleted there is added, renamed or deleted on the field too, and an expanded Dropdown grows or shrinks with the list. Editing the options makes the field's own copy, and **Reset to defaults** goes back to the list, keeping the selection only if the list still has it.

> **Tip:** When you apply a semantic preset (Gender, Payment Frequency, Country, State / Province, Language, Occupation), it comes pre-loaded with industry-standard option lists. These are defaults, not constraints. You can edit, add, or remove options to match the actual form you are documenting. A "Reset to defaults" button restores the original list if needed.
> **Note:** Phone and Height are the only specialized field types that remain as standalone library tiles. Their multi-zone layouts (Phone's type selector and extension zone, Height's metric/imperial dual inputs) cannot be expressed as a preset on a single root component.



A Text Input with the Social Security Number preset. The field takes the preset's label and XXX-XX-XXXX mask, and the Preset row names it.

Layout (8 components)

Layout components provide visual structure without holding data.

- **Section Header:** A bold title line that divides a form into logical groups. Use when you want a label without a bordered frame.
- **Divider:** A horizontal rule (solid or dashed) with an optional text label.
- **Page Header:** A full-width banner with title and subtitle. Useful for screen-level identification.
- **Text Block:** Freeform text for instructions, annotations, or conditional logic notes.
- **Breadcrumb:** A horizontal navigation trail showing hierarchical context.
- **Tab Bar:** A cosmetic tab strip for visual navigation mockups. Unlike Tab Container, this does not scope children.

> **Note:** Tab Bar is a cosmetic component. It looks like tabs but does not switch content. If you want functional tabs that scope different children per tab, use Tab Container instead (covered in the next section).

- **Button Group:** A row of segmented toggle buttons.
- **Stepper:** A step progress indicator showing workflow state (complete, current, not started). Each step can take a **Shown when** (section 9), which prints on the Stepper's row: the demo's Stepper on Review and sign shows its Payer step when "The owner isn't the payer (Owner is payer is No)".

Data (3 components)

Data components display structured information.

- **Data Table:** A tabular display with configurable columns, row count, and optional row-level action buttons.
- **Summary Card:** A key-value display card with a title and reorderable field pairs.
- **Status Badge:** A small colored pill indicating a status (success, warning, error, info, neutral).

Feedback (8 components)

Feedback components represent validation messages and contextual help. Each can be linked to a sibling field so it appears associated in exports.

- **Error Box, Warning Box, Info Box, Success Box:** Message boxes with colored backgrounds and icons. Use to mock validation states at the field group or section level.
- **Inline Error, Inline Info, Inline Warning:** Small inline text messages positioned below a field. Use for per-field validation feedback. A field can also carry its own messages, as a setting on the field (Inline feedback, section 7).
- **Tooltip:** A standalone tooltip bubble with configurable position. Use when you need tooltip content visible on the canvas for documentation purposes. A field's own tooltip, the question-mark icon after its label, is a setting on the field instead (section 7).

> **Note:** Linking a feedback component to a field (via the "Linked field" dropdown in the Properties Panel's Source section) affects export ordering. In the Rules Grid, linked feedback appears directly below its associated field.

Overlay (1 component)

- **Modal Dialog:** A modal overlay with title, message, and configurable buttons. Must be placed inside a container, not on the top-level canvas. See the Containers section for details on visibility and chip behavior.

Media (1 component)

- **Icon:** A decorative icon from a library of 138 Heroicons (70 outline, 68 solid) in nine categories. Configurable size (12 to 64px) and color.

Containers (6 components)

Containers hold other components inside them and provide structural organization. They are covered in detail in the next section.

6. Containers and Nesting

Containers are components that hold other components inside them. They provide the structural backbone of your mockups, organizing fields into tabs, screens, sections, and panels. This section covers each container, when to use it, and how nesting works.

Tab Container

A multi-tab interface where each tab shows a different set of child components. Use Tab Container as the top-level organizer for multi-section applications (a Client tab, an Application tab, an Illustration tab).

- Click tab headers to switch between tabs.
- Each tab has a configurable icon and label.
- Minimum one tab required.

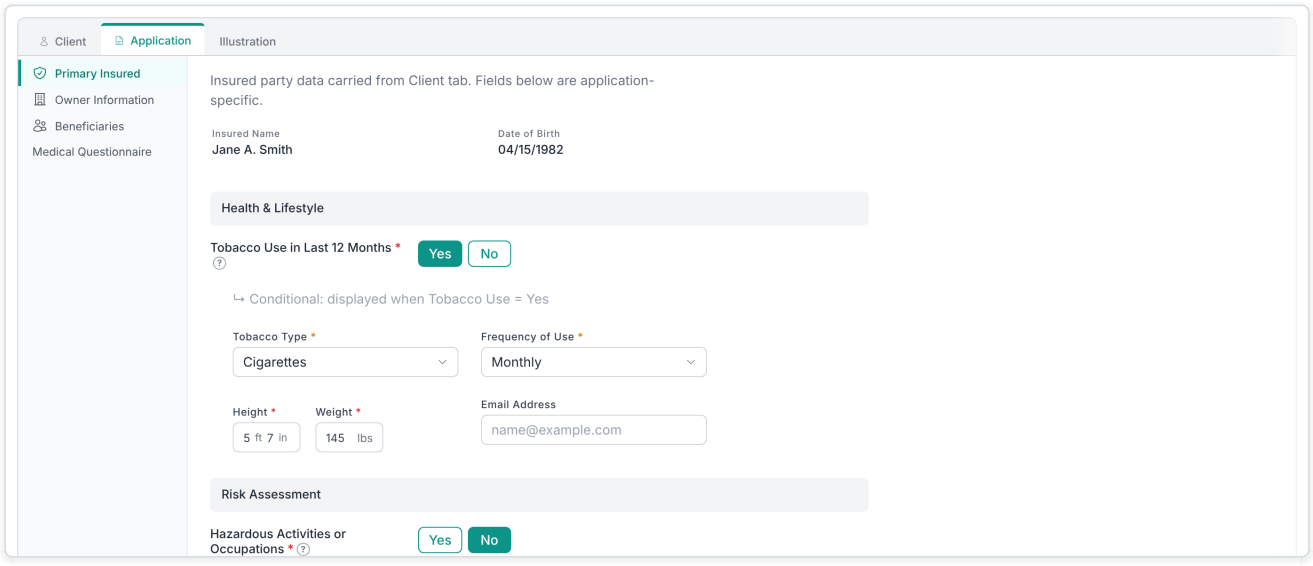
Side navigation: Turn on **Screen nav** in the Properties Panel to give each tab its own set of screens. A sidebar appears showing the screen list. Use this when a single tab needs multiple sub-views (for example, an Application tab with separate Primary Insured, Owner, and Beneficiaries screens). A side-navigation screen takes no **Shown when** of its own; its tab's covers it.

Footer: An optional bar at the bottom with configurable Back, Next, and Submit buttons.

Managing tabs: Add, rename, and remove tabs, give each an icon from the Properties Panel, and a **Shown when** with the funnel on its row (section 9). Tabs keep the order they were added in; unlike screens and steps, they cannot be dragged into a new order. Removing a tab always asks first, and says how many components go with it.

A Tab Container sits on the canvas itself; it cannot be placed inside another container.

> **Tip:** Tab Container with side navigation is a common pattern for multi-section eApp workflows. Each tab represents a major section (Client, Application, Illustration) and screens within a tab represent sub-steps.



A Tab Container with side navigation. The active Application tab has four screens (Primary Insured, Owner Information, Beneficiaries, Medical Questionnaire), with Primary Insured selected.

Screen Container

A multi-screen container with a permanent side navigation panel. Functionally similar to Tab Container with side navigation, but without the tab layer. Use when you need screen-level navigation without the overhead of tabs; the demo project is one Screen Container with ten screens.

- Children are scoped by screen. Click screen names in the sidebar to switch.
- The screen you are on is marked in the project's color theme.
- Same footer options as Tab Container (Back, Next, Submit).
- Add, remove, and reorder screens from the Properties Panel, and give a screen a **Shown when** with the funnel on its row: the demo's Owner screen is shown when "The owner isn't the insured (Insured is owner is No)" (section 9).
- A Screen Container sits on the canvas itself; it cannot be placed inside another container.

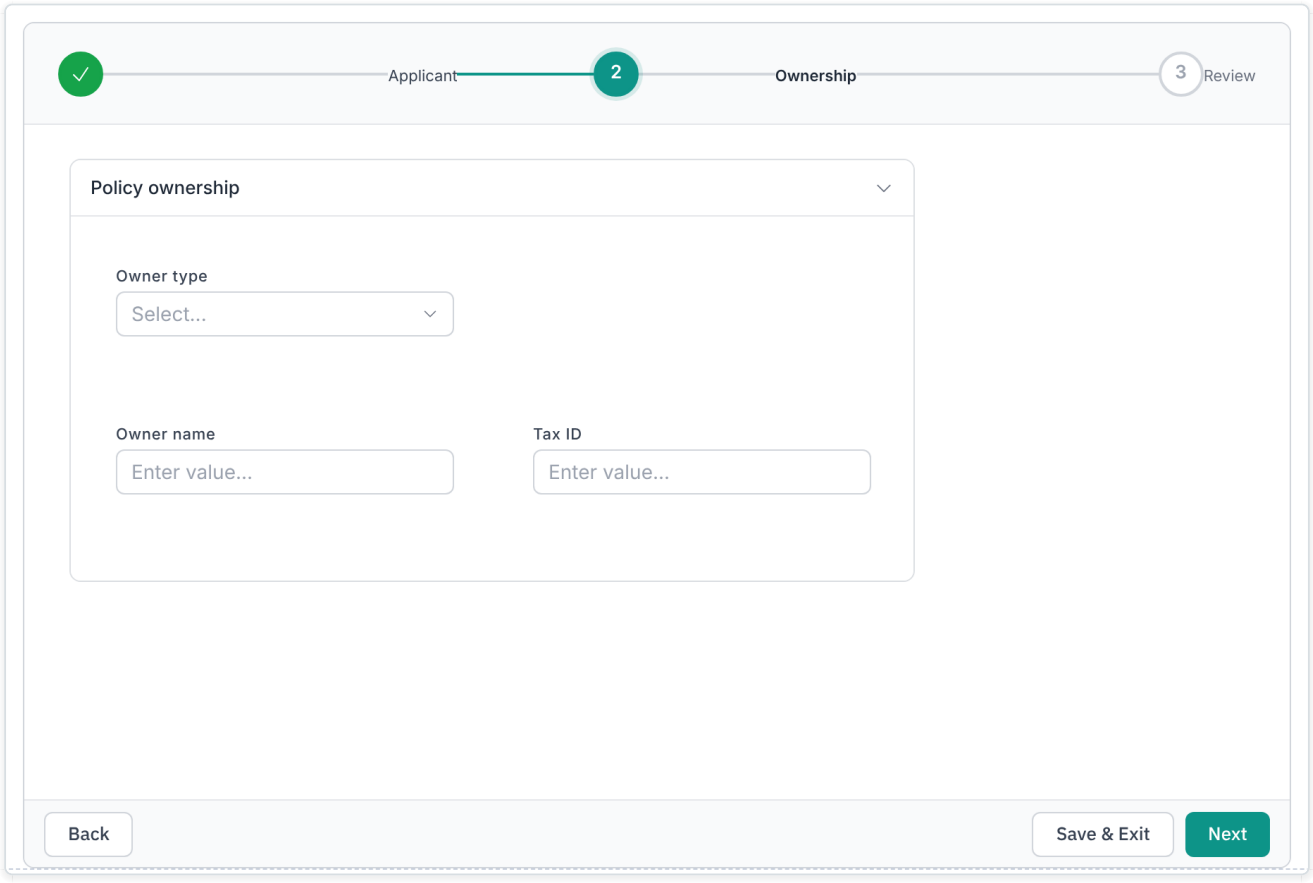
Wizard Container

A multi-step workflow container with a stepper showing progress through a sequence. Each step is marked Not Started, Current or Complete, and a step can take a **Shown when**, under its state on its row (section 9).

- **Layout:** Horizontal (stepper bar at top) or vertical (stepper rail on left).

- Children scoped by step. Click steps to navigate.
- Footer with Back, Next, and Save & Exit buttons.
- Minimum two steps required.
- A Wizard can sit on the canvas or inside a Tab Container.

> **Note:** Wizard Container is less common in real-world eApp patterns, where Tab Container or Screen Container with side navigation is the typical structure. Use Wizard Container when the workflow has strict sequential semantics (like a guided enrollment flow) rather than free navigation.



A Wizard with a horizontal stepper. Applicant is complete and Ownership is the current step.

Collapsible Frame

An expandable/collapsible section with a header bar and chevron toggle. Use Collapsible Frame as the primary section grouping within a screen. Many real application screens are organized as a

series of Collapsible Frames (Personal Information, Contact Information, Employment, etc.). Its **Shown when** covers every field inside it.

- Click the header bar to toggle between expanded and collapsed states.
- Collapsed state shows only the header.
- **Header type:** A label, or a question with Yes/No-style answer buttons in the header itself, which carries its own options, selected value and requirement level, and exports as a Yes / No row.
- **Chevron side:** Left or right (configurable in Properties Panel).
- **Header color:** Customizable background color.

You can nest a Collapsible Frame inside another Collapsible Frame for sub-sections (for example, "Identity Documents" nested inside "Personal Information").

Card

A bounded content card with structured zones. Use Card for self-contained display blocks (a beneficiary summary, a policy overview) rather than for form sections.

- **Header** (optional): Title text with an optional badge or link slot.
- **Field rows** (optional): Read-only key-value display rows, reorderable.
- **Body:** Free-positioning drop zone for child components.
- **Footer** (optional): Primary and secondary buttons with configurable labels.

Cards and Collapsible Frames hold fields and smaller containers, but not the page-level containers: a Tab Container, Screen Container, Wizard or Detail Panel cannot go inside one.

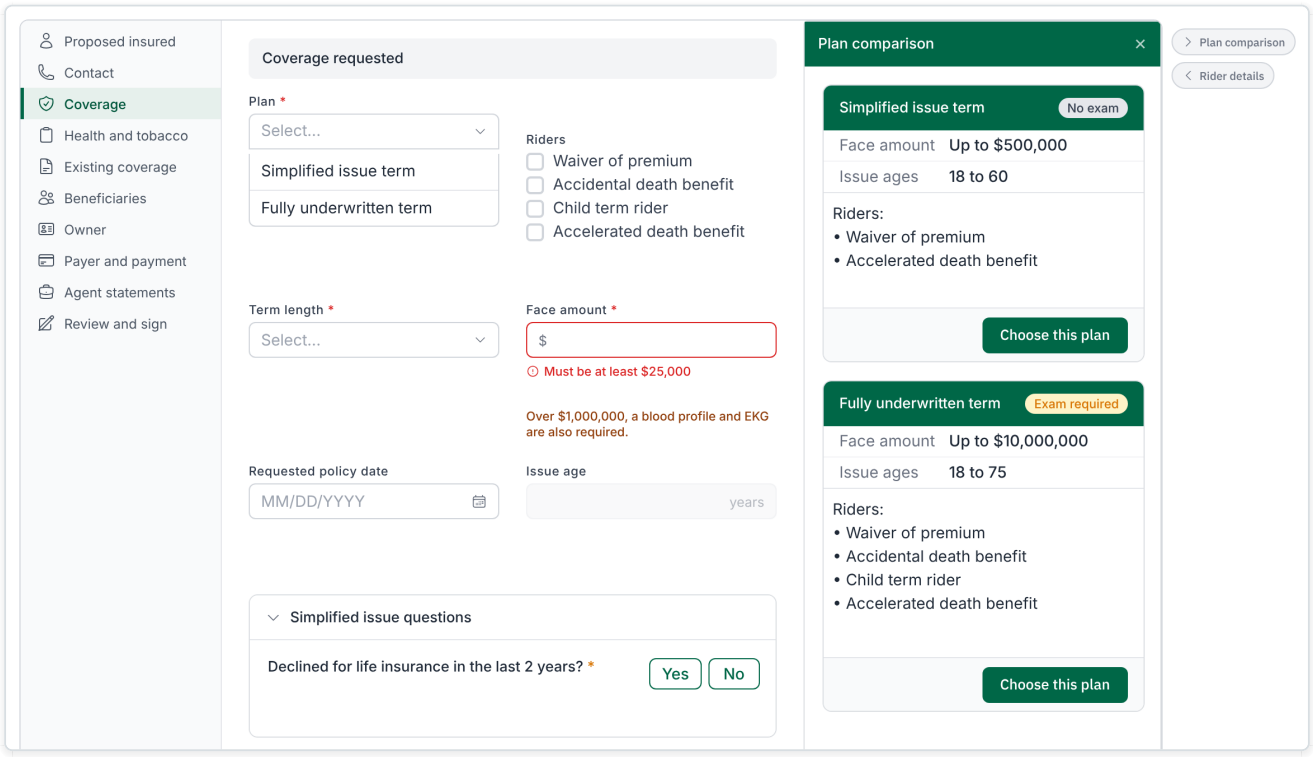
Detail Panel

A side panel for supplementary information. Use Detail Panel for the "click to see details" pattern (for example, selecting a plan or a beneficiary from a list and seeing the full record in the panel).

- Docked to the right edge of its parent container, at the view's full height.
- **Size: Narrow** (320px), **Medium** (480px) or **Wide** (640px) under **Panel width**, or any width in **W**. Its one resize handle, on its left edge, sets the width too. It never draws wider than the container's content, nor narrower than 240px or what it holds.
- Must be placed inside a workflow container (Tab Container, Screen Container, or Wizard).

- Any number of Detail Panels can sit in a view (a screen or step), one open at a time: opening one parks the one that was open. A panel you drop opens.
- **Header:** Optional title text and close button.
- **Open and park:** A Detail Panel's **Open** switch shows it. Switched off, the panel is parked: nothing of it renders inside the screen, and it keeps its chip. **Open** is only what the mockup draws; when the panel opens in the application is its **Opens when**, in its **Rules** (section 9), and neither reads the other.

> **Note:** Chips outside the container's right edge show each Detail Panel's title, open or parked, stacked: the panels first, then any hidden Modal Dialogs. A panel's chip says what a click does, **Open** on a parked panel and **Park** on the open one, and the click selects the panel too. The chips are an authoring aid only and do not appear in image exports.



The list-plus-detail pattern on the demo's Coverage screen: the plan fields on the main area and the open Plan comparison Detail Panel on the right. Outside the screen, Plan comparison's chip, and below it the chip of Rider details, a second panel, parked.

Modal Dialog

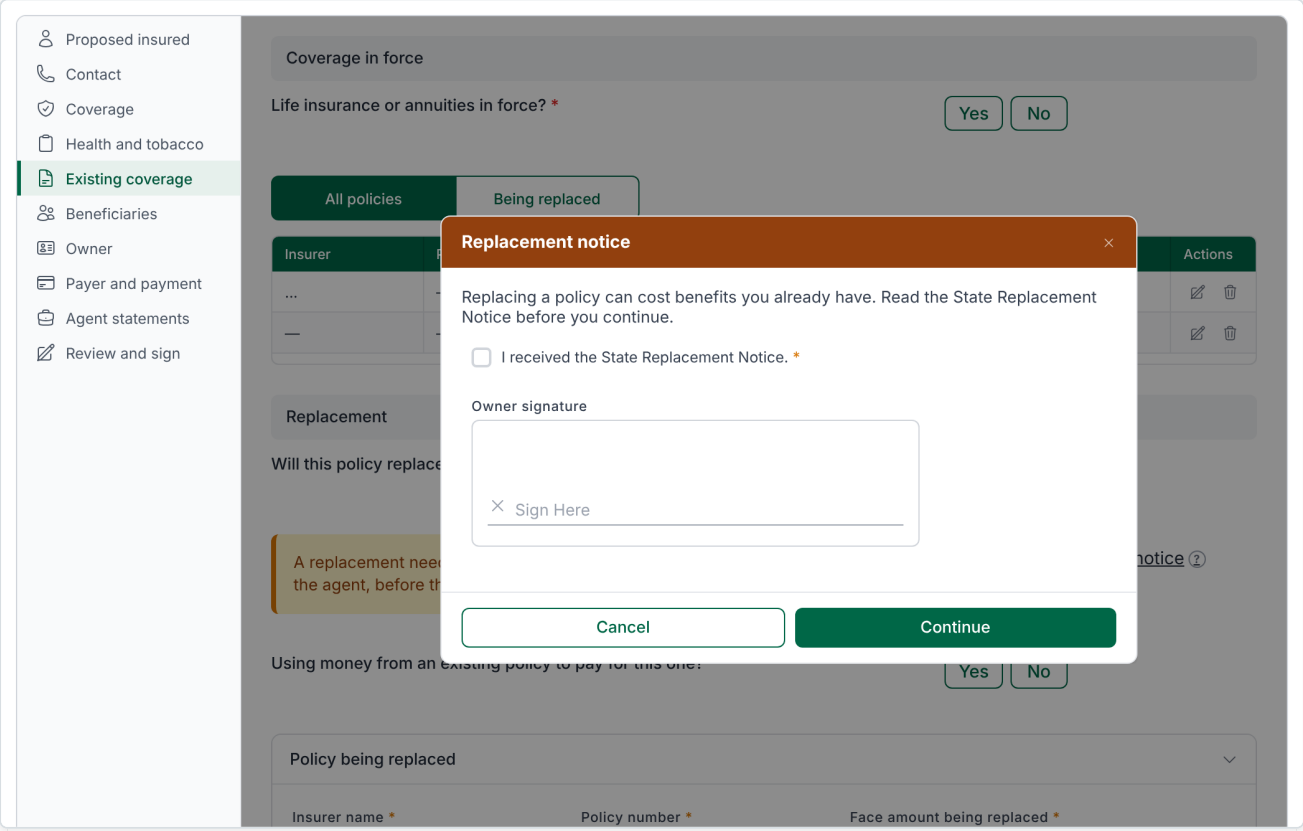
A modal overlay for confirmation dialogs, data entry popups and alert messages. Must be placed inside a container (not on the top-level canvas) and cannot be placed inside a Detail Panel.

- Configure title, message, and button labels in the Properties Panel.
- **Size: Small** (400 × 280), **Medium** (560 × 360) or **Large** (800 × 480) under **Modal size**, or any size in **W** and **H** or with its eight handles. A modal never shrinks below what it holds: no shorter or narrower than its fields need.
- Drop form fields on a visible modal's body to mock a data-entry popup (like "Add Beneficiary"). They draw inside it, where you place them under its message, and nothing of the modal draws while it is hidden, its fields included.

Canvas visibility: Each Modal Dialog has a **Show on the canvas** switch. When it is on, the parent container shows a dim overlay with the modal rendered above it. When it is off, the modal appears as a small chip. The switch is only what the mockup draws; when the dialog opens in the application is its **Opens when**: the demo's Replacement notice opens when "Replacing a policy (Replacement is Yes)" (section 9). Neither reads the other.

> **Note:** When a Modal Dialog is hidden, a pill-shaped chip appears outside its parent container showing the modal title. Click the chip to make it visible (and automatically hide any other visible modal in the same scope). The chips are an authoring aid and never appear in exports. A modal shown on the canvas is drawn in a Download image as you see it, with the dimmed container behind it, so switch it off first for a picture of the screen without it.

Mutual exclusion: Only one Modal Dialog can be visible at a time within the same scope (same container, tab, screen, and step). Showing one automatically hides any visible sibling.



The demo's Replacement notice Modal Dialog, shown from its chip over the dimmed Existing coverage screen, holding its two fields: the "I received the State Replacement Notice." checkbox and the Owner signature.

Nesting Rules

Not every component can go everywhere. Here are the key placement rules:

- **Tab Container and Screen Container** sit on the canvas itself and cannot go inside any container.
- **Wizard** sits on the canvas or inside a Tab Container.
- **Detail Panel and Modal Dialog** cannot be placed directly on the canvas. They must live inside a container.
- **Detail Panel** goes only in a Tab Container, Screen Container, or Wizard.
- **Modal Dialog** takes what a Detail Panel takes and refuses the same five: a Tab Container, Screen Container, Wizard, Detail Panel or Modal Dialog.
- **Card and Collapsible Frame** take any component except a Tab Container, Screen Container, Wizard or Detail Panel.

- **Invalid drops are silent.** Dropping a component onto a surface that rejects it does nothing.

Common Nesting Patterns

These are the patterns you will use most often:

Standard form screen: Screen Container (or Tab Container) > Collapsible Frames > form fields inside each frame. Each Collapsible Frame represents a logical section (Personal Information, Contact, Employment).

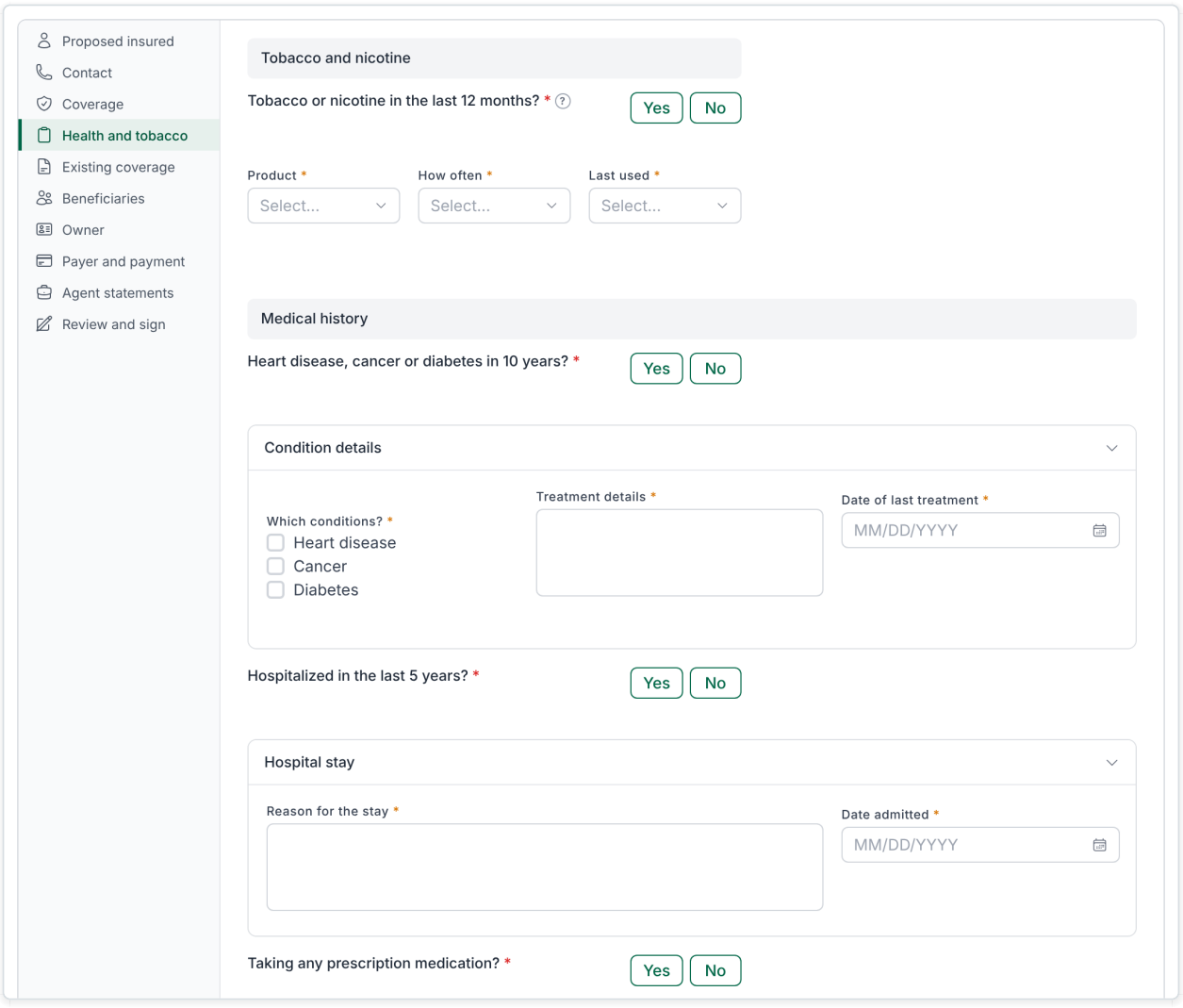
Nested sub-section: Collapsible Frame > Collapsible Frame > fields. Use when a section has a natural sub-group (Personal Information > Identity Documents).

List-plus-detail: A screen with a Data Table or a set of choices on the main area and a Detail Panel on the right showing the selected record's full details.

Data entry modal: A Button labeled "Add [item]" alongside a list, with a Modal Dialog containing the form fields for creating a new entry.

Confirmation modal: A Button that starts an action ("Replace this policy", "Remove beneficiary"), with a Modal Dialog whose title, message and buttons ask before it goes ahead. Shown from its chip, it dims the screen behind it, as the application would.

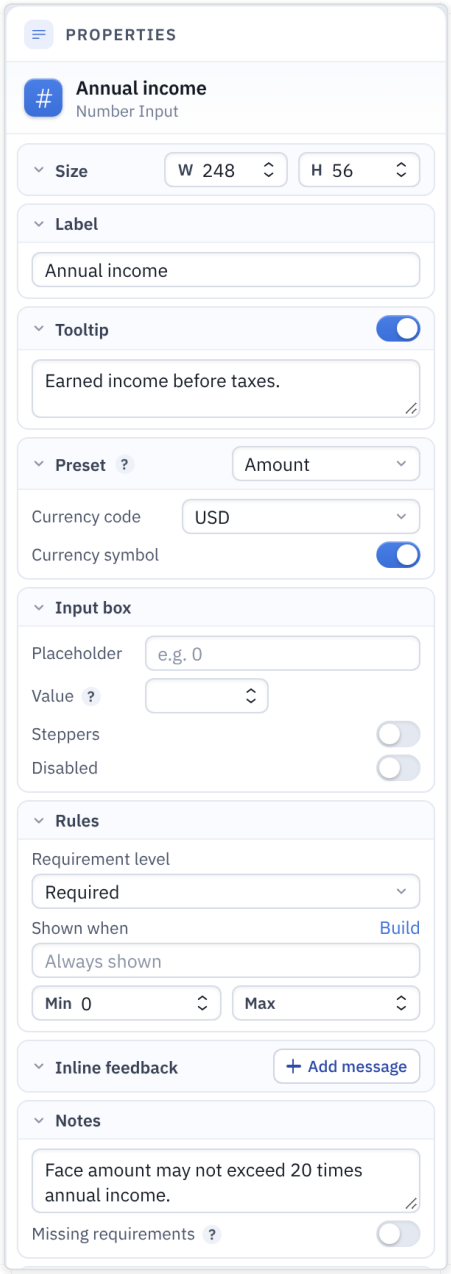
Conditional reveal: A Yes/No question followed by the fields its answer reveals, grouped in a Collapsible Frame. The frame's **Shown when** says when it shows and covers every field inside it; each field that must then be answered is Conditionally Required, with a **Required when** (section 9). In the demo, Condition details is shown when "Major condition is Yes", and Which conditions? is required when "Major condition is Yes". A Text Block on the canvas can say the same thing to anyone reading the mockup.



Conditional reveals on the demo's Health and tobacco screen. Each Yes/No question is followed by the fields its Yes reveals; each frame's Shown when and each field's Required when hold the condition, so the Rules Matrix, the Rules Grid and the Test Cases carry it.

7. The Properties Panel

The Properties Panel on the right edits whatever is selected. Its top names the selection, with its label and its component type. With nothing selected it says so, and with several components selected it counts them and offers **Delete Selected**.



The Properties Panel for the demo's Annual income field.

Sections

A component's settings are grouped into named sections, in the order you reach for them:

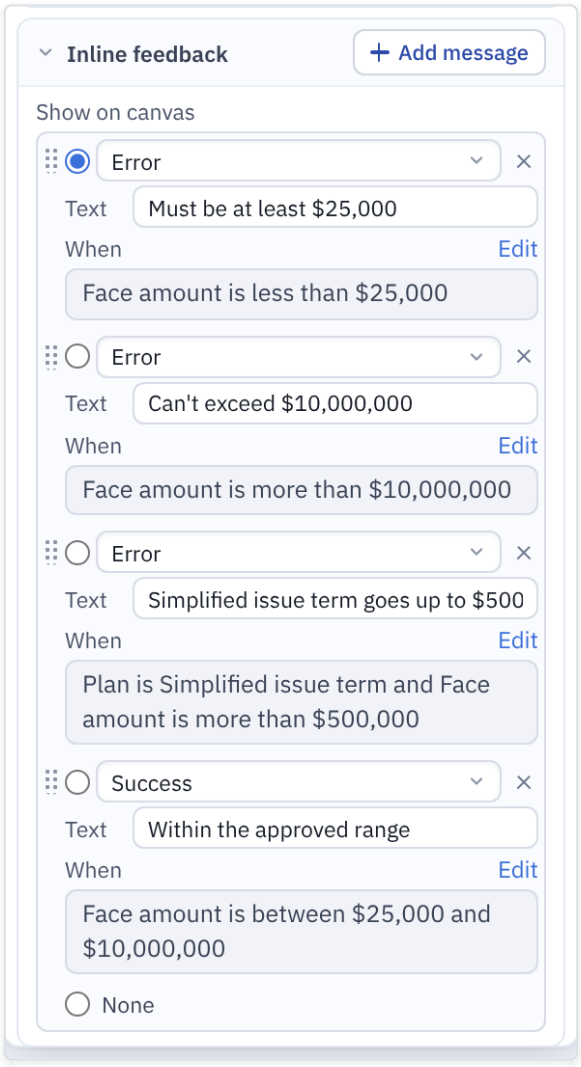
- **Size** first, with width and height on one line, or width alone on a Detail Panel.
- Then the component's own parts, top to bottom: **Label**, **Tooltip**, **Preset** and **Input box** on a field; **Header**, **Body** and **Footer** on a container; **Tabs**, **Steps**, **Columns**, **Buttons** and so on where a component has them.
- Then **Rules**, **Inline feedback** on a field, **Notes**, and **Data**.
- **Text style and color** last, the only section that starts closed.

Click a section's title to open or close it. A closed section still shows its current value in its title row, and a section with a single setting carries it in the title row itself (Open, on a Detail Panel's Body).

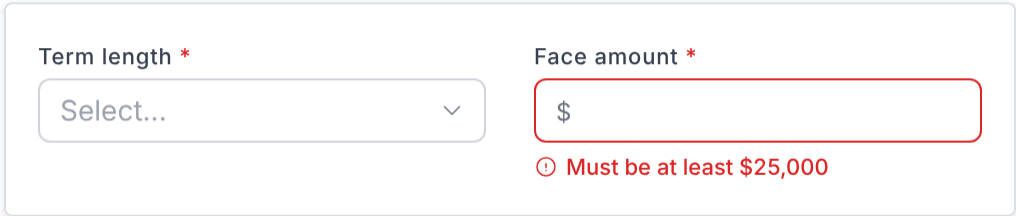
Drag the panel's left edge to widen it, up to 480 pixels; double-click the edge to return it to 300. Trellis remembers its width in this browser.

What the Sections Hold

- **Label:** The field's label.
- **Tooltip:** A switch and its text. Switched on, the field shows a question-mark icon after its label, and the text goes into the Tooltip column of the exports; switched on with no text, the exports say "[Tooltip required]".
- **Inline feedback:** The messages a field gives when what is entered is wrong or worth a word, such as "Must be at least $25,000". **Add message** in the title row adds one, an Error to start. Each message has a type, picked from its menu (Error, Warning, Success or Info), its **Text**, and **When**, when it shows: typed in words, or built with **Build**, as the demo's Face amount builds "Face amount is less than $25,000" (section 9). Drag a message by its handle to reorder the list: the order is precedence, the first message whose condition holds being the one a real form would show. **Show on canvas** picks the one message the canvas shows, or **None**. The shown message hangs under the field at its width, with its type's icon and color, and an Error, Warning or Success recolors the field's border. It takes no room in the layout, so leave a gap under a field that shows one, as you would for an Inline Error. Download image, and the copy button beside Export, take it in. Every message, shown or not, becomes a row under its field in the Rules Grid and in Review, a Feedback row in the Test Cases, or Condition rows where its When is a condition, and a line of its field in the Field Inventory JSON (section 14).



The demo's Face amount: three Errors and a Success, in precedence order, the below-minimum Error shown on the canvas.



The same field on the canvas: the shown Error hangs under it and recolors its border.

- **Preset:** The preset, on the four root components that take one (section 5), with any settings of its own, such as a currency code.
- **Input box:** Placeholder, default value, prefix and suffix, steppers, and other settings for what the field looks like inside its box. **Disabled** is its last row: it draws the field disabled on the canvas

and prints Yes in the Disabled column of the Rules Grid and the Field Inventory. On a component with no input box it sits with the part it draws instead, **List**, **Box**, **Upload** or **Signature**, and under the label on a Button or a Split Button.

- **Options, screens, tabs and steps:** each option of a Dropdown, Radio Group, Yes / No, Multi-Checkbox or Multi-Select, and each screen, tab and step in its container's list, has a funnel before its ×: **Offered when** on an option, **Shown when** on a screen, tab or step (section 9).
- **Rules:** What is required of the component, and when. On a field that takes an answer it opens on the **Requirement level**: Required (a red asterisk on the canvas), Conditionally Required (an amber asterisk), or Optional. While the level is Conditionally Required, **Required when** sits under it: when the field must be answered, typed in words or built with **Build** (section 9). Left empty it reads **Not set**, and the Dependency Map lists the field as a gap, unless the field is a File Upload or an E-Signature (section 15). Then **Shown when**, or **Opens when** on a Detail Panel or a Modal Dialog: when the component shows, or opens, in the application (section 9); empty, it reads **Always shown**, or **Not stated**. On a component that takes no answer, that is the section's one row. A field's limits come last: minimum and maximum length or value, date direction, allowed file types, earliest and latest times. The limits appear in your exports; they do not enforce anything on the canvas.
- **Notes:** Free-text notes for the field's rules and business logic, which become the Rules / Notes column of the Rules Grid after a line for each slot that holds something, and a **Missing requirements** switch that flags the field as still needing requirements. A flagged field's row is pink in the Rules Grid and in Review.
- **Data:** What the field collects and where it goes (section 8).
- **Text style and color:** Font size, weight, style, color and alignment, and color presets.

At the foot of the panel, **Save to Pack** saves the selected component and everything inside it to a component pack (section 12), and **Delete Component** deletes it.

8. Data, Lists, Forms and XML

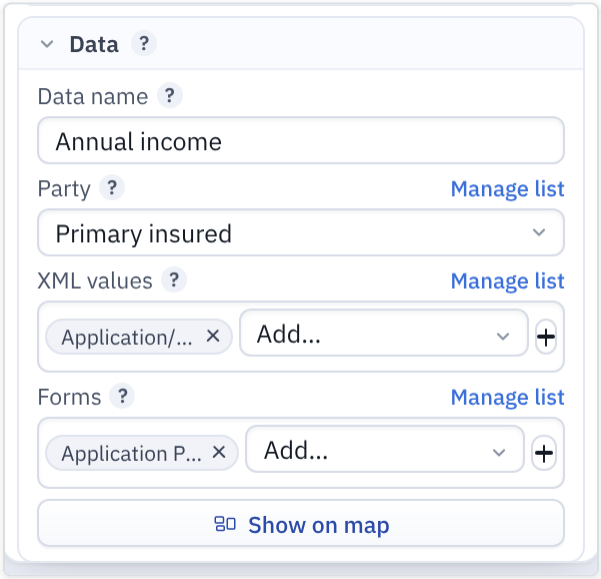
A field's label is what the applicant reads. Its data is what the field collects: the same piece of information can be asked for on more than one screen, sent in the new business XML file under one or more XML values, and printed on one or more forms. Trellis records all of that, so the Dependency Map can show it and the exports can carry it.

Four places work together:

- The **Data** section of each input field's Properties Panel names the data and says where it goes.
- **Forms** in the rail is the project's paper and PDF forms, with their signers, editions and the data printed on each.
- **XML** in the rail is the project's XML values, with what is sent for each party and the codes its options are sent as.
- **Lists** in the rail keeps the lists the rest pick from: parties, products, form types, messages and saved conditions.

The Data Section

Every input field (Text Input, Text Area, Search Input, Number Input, Dropdown, Radio Group, Checkbox, Toggle Switch, Yes / No, Multi-Checkbox, Multi-Select, Date Field, Time Picker, Phone and Height) has a Data section, after Notes.



The Data section for the demo's Annual income field, each row's label over its control.

- **Data name:** What this field collects, named the same way everywhere it appears. It starts out filled in from the field's label, and the data exists once you confirm a name: press Enter or click away. As you type, names already used in the project are suggested, each with the number of other screens it is on.
- **Party:** Whose information it is, picked from the project's parties as the Lists page keeps them, in that order. **Add a new party…**, last in the list, types a new one, which snaps to an existing spelling and joins the list. **Manage list**, under the picker, opens the Lists page.
- **XML values:** The XML values this data is sent as, as chips. Pick from the project's list, or add a new path right there with **+**. Leave it empty if the data is not sent.
- **Forms:** The forms this data is printed on, picked or added the same way.
- **+ on a field with no data yet** adds the new XML value or form to the project's list but does not put it on the field. Confirm the data name first, or pick the new value from the list once it is there.
- **Manage list** under XML values and under Forms opens the XML or Forms view, where the whole list is edited.
- **Show on map** opens the Dependency Map centered on this field (section 15). Until the data has a name, it is grayed out and the section says "Name the data first."

Picking an XML value or a form on a field with no data yet creates the data, under the name the section is showing.

Shared Data

Two fields with the same data name and party are the same data, whatever their labels. Naming a second field "Date of birth" with the party Primary insured joins it to the first: they share their XML values and forms, and the Dependency Map shows the data on both screens. If the two could not collect the same thing (a date field and a text field, say), the section says why under the name instead of joining them.

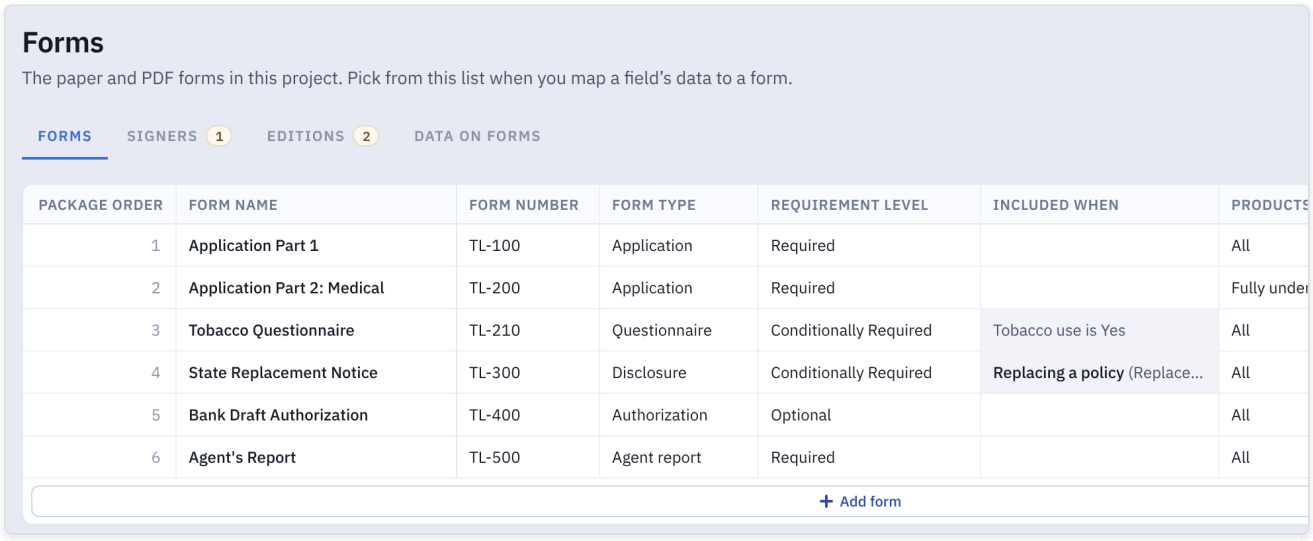
Renaming data that more than one field carries asks first: **All N fields** renames it everywhere, **Just this one** moves only this field to the new name, and **Cancel** leaves everything as it was.

Data that nothing uses any more goes with its last use. When its last field is cleared or deleted (alone or with its screen, tab or step), or its last XML value or form is deleted, and nothing else is left on it, the data is deleted in the same step; one undo brings both back. Data that still has an XML value or a form stays, left behind, until a field names it again, and the Dependency Map lists it as a gap. A condition reads answers, so deleting the last field that collects a piece of data breaks every condition that reads it; the toast says how many. While a form or an XML value still holds the data, a field that collects it again mends them. Data that went with its last field is gone, and the same name makes new data: undo the delete, or pick the new field in each broken rule (section 9).

Copying a field to another project brings its data with it, matched there by name and party.

Forms

Forms in the rail is the paper and PDF forms in the project, in four tabs, each a sheet of the Forms matrix export: **Forms**, **Signers**, **Editions** and **Data on forms**. A tab's count shows how many of its rows are gaps.



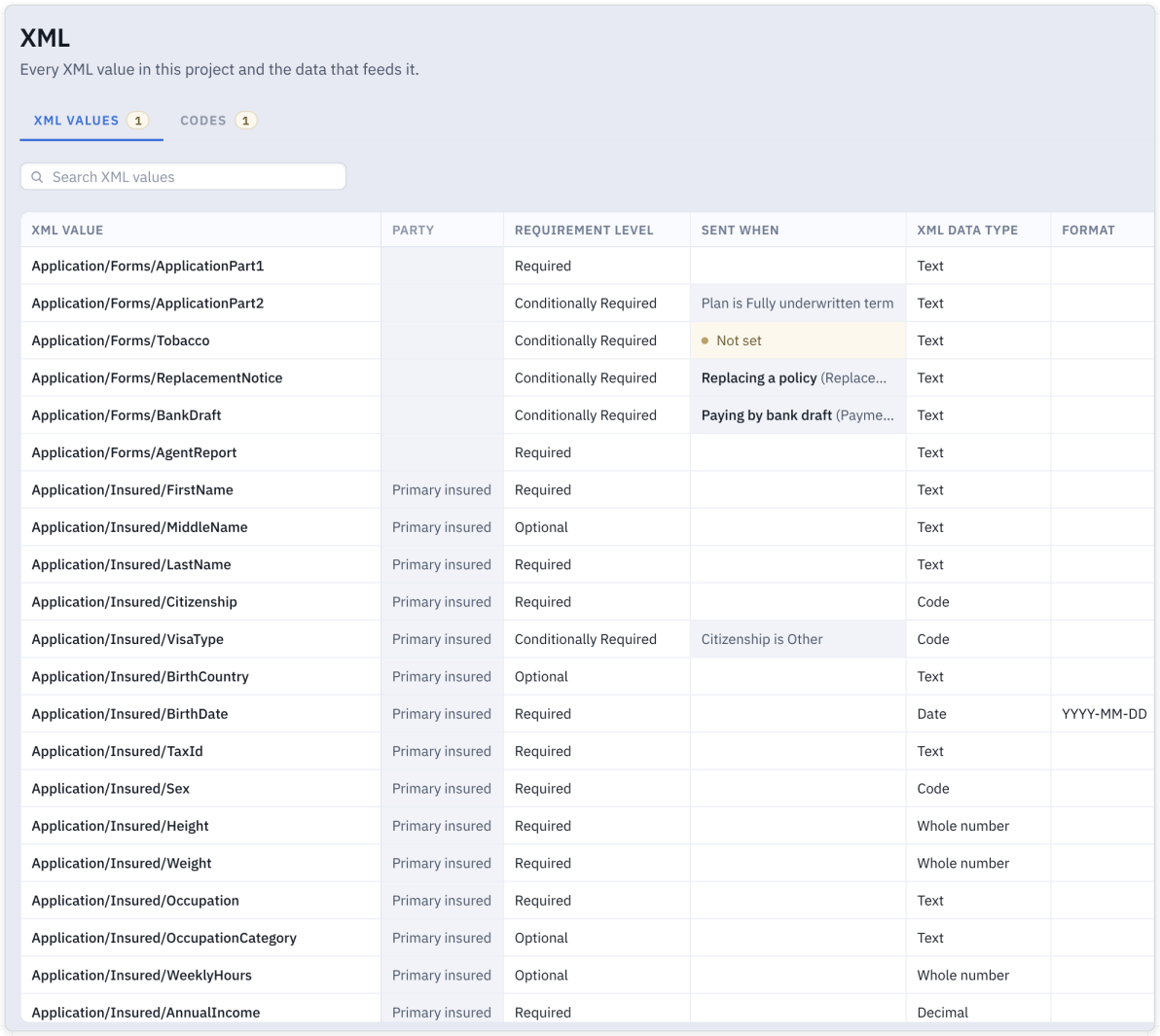
The demo's six forms on the Forms tab, in package order. Package order and Form name stay in view as the table scrolls sideways.

- **The Forms tab** has a row per form, in package order: Package order, Form name, Form number, Form type, Form source, Collected, Requirement level, Included when, Products, Provinces and states, Languages, Repeats per, Signing method, Current edition, Effective, Retires, XML value and Used by. It opens on its main columns; **All columns** shows every one, and **Main columns** goes back. Click a cell to type in it or pick from its menu; Form type's menu ends with **Add a new form type…**. **Included when** says when a Conditionally Required form is included: type words in its cell, or point at it and click **Build** at its right end to build a condition (section 9); a cell that holds one shows its sentence, and a click, Enter or F2 opens it in the builder. Until it has words or a condition it is a gap. **Products**, **Provinces and states** and **Languages** are **All** or the ones you pick; while a pick limits a form and nothing decides that row on the Lists page, the menu offers to set it there and then. Current edition, Effective and Retires are worked out from the form's editions: the one in effect today.
- **Package order** is changed by dragging a row by its number, or with the arrow keys on it; each move is one undo step. **Add form**, at the foot of the table, takes a form name, an optional form number and, optionally, the form's XML value; a new XML value is added from a row's XML value cell, **Add an XML value…**.
- **Signers** lists each form's signers in signing order, under the form: the party (**This copy's party** on a form printed once per party), the requirement level and, while Conditionally Required, **Signs when**, typed or built as Included when is; a gap until it has words or a condition. **Add signer** takes the form and the party; drag a signer within its form to change the order.
- **Editions** lists each form's editions under it: the edition's label, and the dates it is in effect, typed MM/DD/YYYY. Two editions of one form in effect on the same day are flagged.

- **Data on forms** lists the data one form prints, the form picked above the table: each piece of data and its party, the form field name, the format and, for data with options, the mark each option prints as.
- **Used by** says how many pieces of data print on the form ("7 pieces of data", "1 piece of data", "Not used"). Click it to list them, each as "name · party"; click a name to open the Dependency Map on it. **Show on map** opens the Dependency Map centered on the form.
- The **×** at the end of a row deletes it. Deleting a form always asks first, and says what uses the form and any data that would go with it; the data that used it lets go of it, in one undo step.

XML

XML in the rail is every XML value in the project and the data that feeds it, in two tabs, each a sheet of the XML mapping export: **XML values** and **Codes**.



XML VALUE	PARTY	REQUIREMENT LEVEL	SENT WHEN	XML DATA TYPE	FORMAT
Application/Forms/ApplicationPart1		Required		Text	
Application/Forms/ApplicationPart2		Conditionally Required	Plan is Fully underwritten term	Text	
Application/Forms/Tobacco		Conditionally Required	Not set	Text	
Application/Forms/ReplacementNotice		Conditionally Required	**Replacing a policy** (Replace...	Text	
Application/Forms/BankDraft		Conditionally Required	**Paying by bank draft** (Payme...	Text	
Application/Forms/AgentReport		Required		Text	
Application/Insured/FirstName	Primary insured	Required		Text	
Application/Insured/MiddleName	Primary insured	Optional		Text	
Application/Insured/LastName	Primary insured	Required		Text	
Application/Insured/Citizenship	Primary insured	Required		Code	
Application/Insured/VisaType	Primary insured	Conditionally Required	Citizenship is Other	Code	
Application/Insured/BirthCountry	Primary insured	Optional		Text	
Application/Insured/BirthDate	Primary insured	Required		Date	YYYY-MM-DD
Application/Insured/TaxId	Primary insured	Required		Text	
Application/Insured/Sex	Primary insured	Required		Code	
Application/Insured/Height	Primary insured	Required		Whole number	
Application/Insured/Weight	Primary insured	Required		Whole number	
Application/Insured/Occupation	Primary insured	Required		Text	
Application/Insured/OccupationCategory	Primary insured	Optional		Text	
Application/Insured/WeeklyHours	Primary insured	Optional		Whole number	
Application/Insured/AnnualIncome	Primary insured	Required		Decimal	

The demo's XML values. XML value stays in view as the table scrolls sideways.

- **The XML values tab** has one row per XML value and party: a value sent for two parties draws two rows with the same path, and a value nothing uses draws one row with an empty party. It opens on its main columns (XML value, Party, Requirement level, Sent when, XML data type, Format, Value comes from, Data and Used by); **All columns** shows every column, **Messages** among them once the project has a message.
- A path's own settings (Attribute name, XML data type, Format, Max length, Repeats, When there’s no answer, Value comes from, Standard or extension and the rest) are shared by every row of the path; each is there only while what it depends on calls for it, so **Up to** shows while Repeats is Can repeat up to a number. A row's **Requirement level** and its **Sent when** are its own; Sent when is typed or built as the Forms tab's Included when is, and is a gap while it is Conditionally Required and has neither words nor a condition.

- **Party** and **XML role code** are read from the data and the Lists page, not edited here. **Data** and **Fields** name the data the row carries and the fields that collect it.
- **The Codes tab** lists each XML value of type Code, and under it every option that feeds it, from all its parties, with its code typed beside it. An option with no code is a gap.
- **Add XML value**, at the foot of the XML values tab, takes a path. A path already in the project is refused as you type ("Already in the list."), and so is a rename onto one.
- **Search XML values** filters the open tab's rows: on XML values by path, party, data name and the options that feed a row, on Codes by path and option. A clear button sits at its right edge; the line beside it counts the rows shown ("3 of 119").
- **Used by** opens to name the data, then the forms, on that row. **Show on map** and the **×** work as they do in Forms: deleting a value always asks, and the dialog says who uses it.

Lists

Lists in the rail keeps every list the project picks from, each once, so a party, a product or a form type is spelled one way everywhere.



The demo's lists. Each says where it is used.

- **Parties**, **Products**, **Form types** and **Messages** are the project's own: add one with the add at the foot of its table, and click a name to rename it everywhere it is used. **Used by** says what uses each; an item in use cannot be deleted, and the **×** says what holds it; a product a condition reads counts it too ("1 piece of data and 3 conditions"), and cannot be deleted while one does.
- A party's **XML role code** is what its rows send; the XML view reads it from here.
- **Conditions** holds the conditions saved for reuse (section 9), each with its **Condition name**, **Reads as** and **Used by**.

- **Built into Trellis** are the lists every project shares: provinces and states, languages, countries, occupations, payment frequencies and genders. A field can take one as its options, and forms pick provinces, states and languages from them.
- **Applies to** says what decides a form's Products, Languages and Provinces and states: pick the piece of data whose answer it is in **Decided by**. A form limited to some of them applies when the application's answer is one of them.

9. Conditions

A condition says when something applies: when a field is required, when a component, a screen, a tab or a step is shown, when a panel or a dialog opens, when an option is offered, when a message shows, when a form is included, when a signer signs and when an XML value is sent. You build it from rules, each comparing one field's answer with a value, such as "Citizenship is Other". Because Trellis can read a condition, it writes the rule's sentence for you, writes test cases from it, and draws what it connects on the Dependency Map. Every condition is optional, and words you type still work wherever a condition can go.

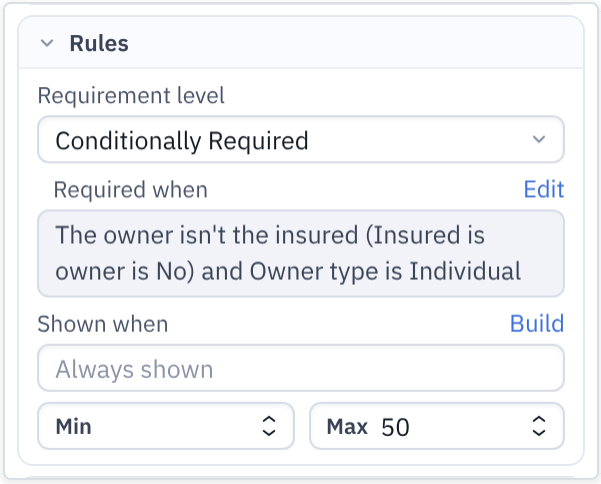
Where a Condition Goes

A condition sits on the thing it governs, in a slot named for what it decides:

Slot	On	Where you find it
Required when	A field whose requirement level is Conditionally Required	The Properties Panel's **Rules**, under Requirement level
Shown when	Any component but a Detail Panel or a Modal Dialog	The Properties Panel's **Rules**
Opens when	A Detail Panel or a Modal Dialog	The Properties Panel's **Rules**
Offered when	An option of a Dropdown, Radio Group, Yes / No, Multi-Checkbox or Multi-Select	Under the option's row, with the funnel before its **×**
Shown when	A screen, a tab or a step	Under its row in its container's list, with the funnel
When	An inline feedback message	**Inline feedback**, on each message
Included when, **Signs when**	A form, a signer	The Forms view, in the cell
Sent when	An XML value's row	The XML view, in the cell

Required when is there only while the field is Conditionally Required: choose another level and it goes, and undo brings it back. Included when, Signs when and Sent when work the same way. Shown when and Opens when say what the application does, not the canvas: a Detail Panel's **Open** switch and a Modal Dialog's **Show on the canvas** still decide only what the mockup draws, and neither reads the other.

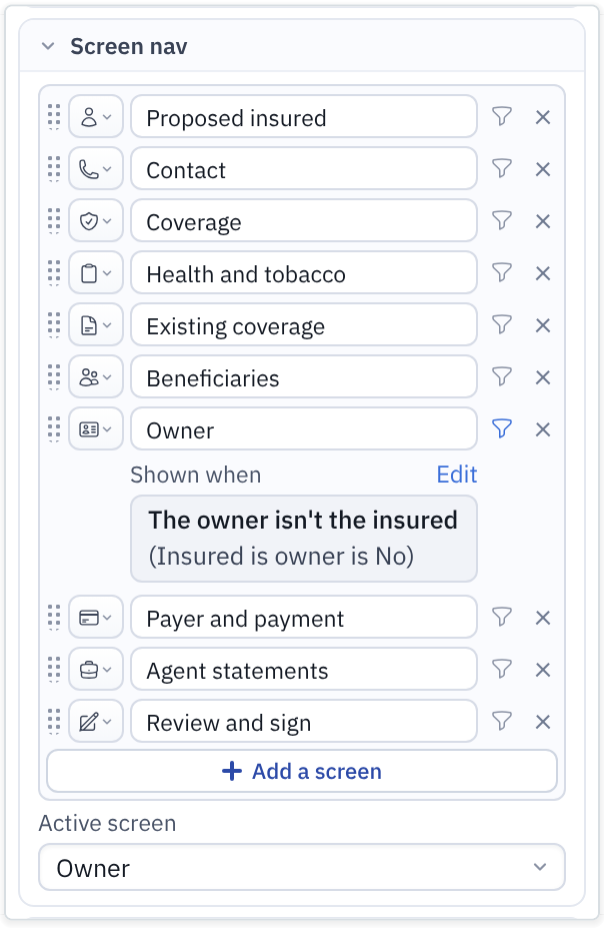
A slot with nothing in it says so: **Not set** in Required when, which the Dependency Map lists as a gap on any field but a File Upload or an E-Signature; **Always shown** in Shown when; **Not stated** in Opens when; and **Always offered** in Offered when.



The demo's Owner first name, required when "The owner isn't the insured (Insured is owner is No) and Owner type is Individual".

Screens, Tabs, Steps and Options

A screen, a tab or a step has a **Shown when** under its row in its container's list, and an option of a choice field an **Offered when** under its own. Click the funnel before the row's **×** to open the slot; a row that holds something stays open, and its funnel is colored. The demo's Owner screen is shown when "The owner isn't the insured (Insured is owner is No)", and its Premium frequency offers Monthly when "Paying by bank draft (Payment method is Bank draft)". A screen's, tab's or step's condition cannot test a field it holds, since the field could never be answered into view.



The demo's Owner screen, shown when the owner isn't the insured.

Building a Condition

Step 1: Open the builder. Click **Build** at the right of the slot. The builder opens over the canvas, named for what the condition is on: the field's name, then its type and its screen.

Step 2: Pick a field. In **Find a field**, type part of the label of the field whose answer you are testing, and pick it. Fields are listed under their screens in the order you read them, each with its data's name where that differs from its label, since the sentence uses that name. The field the condition is on is not offered, nor any other field that collects the same data, since a condition cannot test its own answer; a message's When is the exception and tests its own field.

Step 3: Pick the comparison. Comparison offers only what the field's answer allows:

- A Dropdown or a Radio Group: is, is not, is one of, is not one of.
- A Multi-Checkbox or a Multi-Select: includes any of, includes all of, includes none of.
- A Yes / No: is, with either of its two labels.

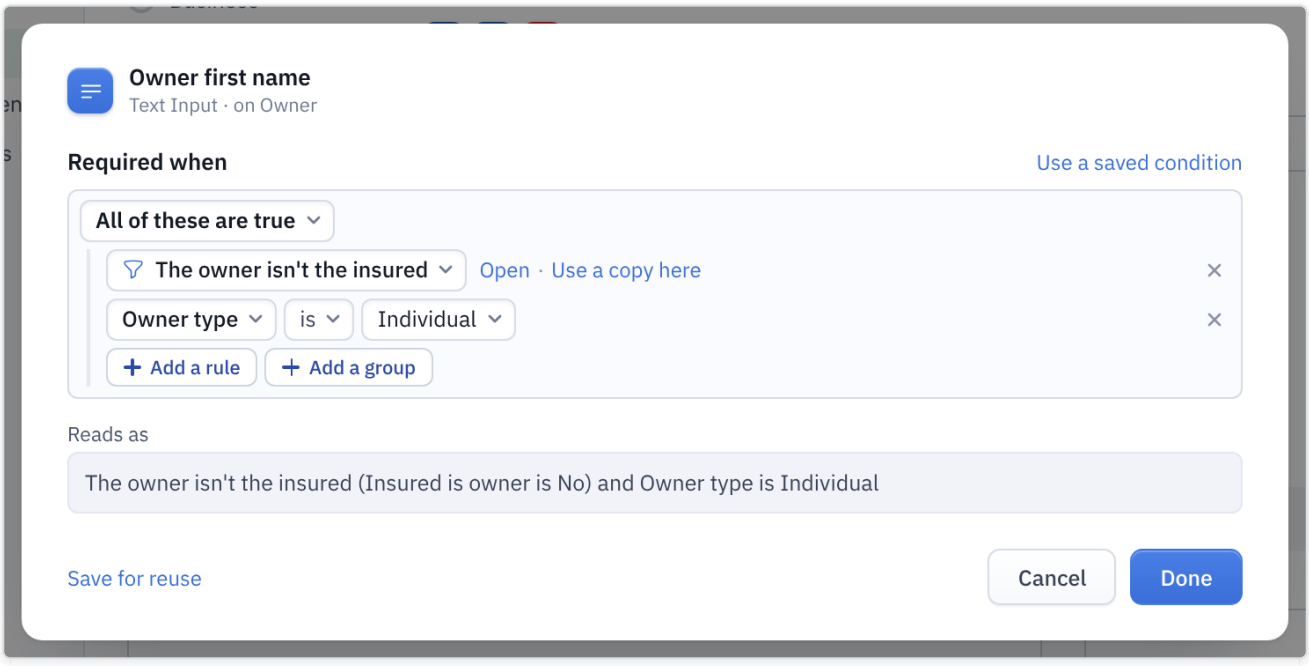
- A Checkbox or a Toggle Switch: is checked, is not checked.
- A Number Input or a Height: is, is not, is more than, is at least, is less than, is at most, is between, is not between. Is between includes both ends.
- A Date Field: is, is before, is on or before, is after, is on or after, is between.
- A Text Input, a Text Area, a Search Input, a Phone or a Time Picker: is, is not.
- Any field: is answered, is not answered.

Step 4: Give the value. Value follows the comparison: one of the field's options, several options ticked in a list, a number, a date (MM/DD/YYYY) or words to type, two for between, or nothing after is checked, is answered and their opposites. On a number or a date rule other than a between, point at the rule and **Use a field** compares with another field's answer instead, times a number on a number and plus or minus a number of days on a date; **Use a number** or **Use a date** goes back.

Step 5: Read it, and click Done. Reads as, under the rules, shows the sentence every export prints, as you build. The sentence names a field by its data name where the field has one. The rule itself shows the field's label, exactly as it appears on the canvas. Click **Done**.

For a condition of more than one rule, **Add a rule** adds one to its group and **Add a group** adds a group inside it; **Remove rule** and **Remove group** take one out. A group is **All of these are true** or **Any of these is true**, picked at its head, and groups sit inside groups as deep as the condition needs. The demo's Face amount shows its third Error when both of its rules hold: "Plan is Simplified issue term and Face amount is more than $500,000". Its Email on Contact is required when either of its rules holds: "Policy documents by email is checked or Signed copy by email is checked".

A slot that holds a condition shows its sentence on a tinted ground, with **Edit**; click either to open it in the builder again. **Cancel**, or Escape, closes the builder and changes nothing, and each Done is one step for undo.



The builder on the demo's Owner first name: a saved condition beside a rule of its own.

Words Still Work

Every slot still takes words. Type in it as before when no rule can say what you mean: the demo's Occupation category is required when "the job title isn't listed", and its Height warns when it is "Outside the standard range". Trellis prints words exactly as you typed them and reads nothing into them.

A slot holds words or a condition, never both. Click **Build** on a slot that holds words and the builder keeps them in view ("Typed words: … Done replaces them.") until you click **Done**; undo brings them back.

Saving a Condition for Reuse

A condition the application needs in several places is written once and saved under a name.

- **Save for reuse**, at the foot of the builder, asks for a **Condition name**, filled in with the sentence to start. Type the name and click **Done**: the condition is saved, and the slot uses it. **Don’t save it**, the cross on the name, keeps the condition on this slot alone.
- **Use a saved condition**, at the right of the slot's name under the builder's head, lists the project's saved conditions, each with its sentence; pick one and it is a rule of the condition, alone or beside others.

- On a saved rule, **Open** opens the saved condition itself and says how many places use it ("Used in 13 places. Changing it changes all of them."), and a change there changes every one of them; **Use a copy here** gives this slot its own copy of the rules instead.

A saved condition prints as its name, then its sentence in parentheses: "The owner isn't the insured (Insured is owner is No)". The demo saves six.

Conditions, on the Lists page after Messages, holds the project's saved conditions (section 8): **Condition name**, typed in place to rename it; **Reads as**, its sentence, with **Edit**, which opens it in the builder; and **Used by**, how many places use it, which opens to name each place as what the condition makes true there ("Owner type is required", "The Owner screen is shown"), each a way to it. A condition is saved from the builder, so the list has no add. The **×** deletes one that nothing uses; one in use says how many places hold it and cannot be deleted.



The demo's saved conditions. Used by opens on every place that uses one.

What a Condition Gives You

- **Rules / Notes**, in the Rules Grid and in Review, prints a line for each slot that holds something, before your notes. The demo's Owner SSN or TIN reads "Required when The owner isn't the insured (Insured is owner is No)", then its note, "9 digits." A field's lines come in the order Required when, Shown when, then one for each option with an Offered when ("Monthly: offered when Paying by bank draft (Payment method is Bank draft)"); a screen's, a tab's or a Wizard step's prints on its own row, and a Stepper step's on the Stepper's row, named for the step; a message's row reads its sentence.
- **Test Cases** gain **Condition** rows, written from the condition: one where it holds, one where it does not, one for each rule that changes the outcome on its own, and one at each number or

date limit. The Scenario is the answers that make the row, and the Expected Result what the screen does. The demo's Visa type writes "Citizenship is Other", "Error: Required field, on an empty submission", and "Citizenship is not Other", "Accepted, on an empty submission". They take the place of the field's Required row, or of a message's Feedback row; a slot of words keeps the row it had.

- **The Field Inventory JSON** gives each field its `requiredWhen`, `shownWhen` and each option's `offeredWhen`, each its sentence with the condition beside it, read by names; **the Data map JSON** lists the saved conditions; and **the Forms matrix** and **the XML mapping** print the sentence in Included when, Signs when and Sent when (section 14).
- **The Dependency Map** draws what a condition connects (section 15). A piece of data, or a field with no data, lists what its answer decides: **Decides screens**, **Decides fields**, **Decides XML values** and **Decides forms**. What a condition decides lists what it reads: **Decided by fields** and **Decided by data**. Three more gaps join the map's: a field with no Required when, a broken condition, and conditions that depend on each other.

When Something a Condition Reads Is Deleted

Deleting a field, an option or a piece of data that a condition reads is never refused. The condition is kept and marked broken, what was deleted reads "(deleted)" in its sentence, and the toast says how many conditions broke: deleting the demo's "Is the proposed insured the owner?" says "Component deleted. 1 condition read this and is now broken. Undo brings it back." In the builder a broken rule names what was deleted; pick a replacement, or remove the rule. A broken condition is a gap on the Dependency Map, and before you delete, the map's strip says how many conditions the delete would break.

Two deletes are refused instead: a saved condition still in use, and a product a condition reads.

Copied components keep their conditions. Inside one project, a duplicate or a paste reads what the original reads. The one exception is a rule on a field with no data that was copied with it: the copy's rule reads the copy's field. A slot that uses a saved condition keeps using it. Pasted into another project, or saved to a component pack, a condition that reads something outside what was copied is left out, and the toast counts them: "1 condition didn't come along. It read something outside what was copied." A saved condition the other project lacks, and every saved condition in a pack, arrives as the slot's own copy of its rules (sections 4 and 12).

Conditions That Depend on Each Other

Two conditions that test each other's fields are allowed, since you may be partway through, and the Dependency Map lists them as a gap, "with conditions that depend on each other", until one of them changes.

What a Condition Can't Say Yet

- Rules over repeating rows, such as "contingent shares must total 100%".
- Calculations and calendars, such as "at least 2 business days after the date signed", or today's date.
- A setting that changes with another answer, such as a minimum that differs by plan. Write the narrower limit as an Error message with a condition, as the demo's Face amount does for Simplified issue term.
- Disabled or read-only when.
- What happens to the answer of a field that becomes hidden.
- Who is filling in the application, at what stage and through which channel.
- A Collapsible Frame's header question, and a Tab Container's side-navigation screens.
- Test cases for forms, signers and XML values.

10. Themes

Trellis has two independent theming layers that control how your mockup looks. You can change either at any time from the canvas toolbar, and any combination is valid. Themes change the screens you are designing and nothing around them: the rail, the top bar and the panels keep their own colors.

Color Theme

The **Color** dropdown on the canvas toolbar controls the primary accent color used by canvas components (tab bars, buttons, headers, active states).

Theme	Accent	Best For
Horizon (default)	Teal	General-purpose, Trellis identity
Enterprise	Navy	Matching legacy L&A platform aesthetics
Sage	Muted green	Healthcare-adjacent, calm tone
Ember	Warm amber	Warm, approachable
Sky Blue	Bright blue	Optimistic, modern
Deep Navy & Gold	Dark navy	Premium, boardroom presentations
Forest Green	Deep green	Institutional
Burgundy	Wine red	Formal, authoritative
Crimson & Slate	Energetic red	High-energy, bold
Custom	Your hex code	Match any brand or carrier palette

Custom color: Select "Custom" and enter any 3- or 6-digit hex code in the box that appears beside it. Dark, light, and border variants are derived automatically.

Style Theme

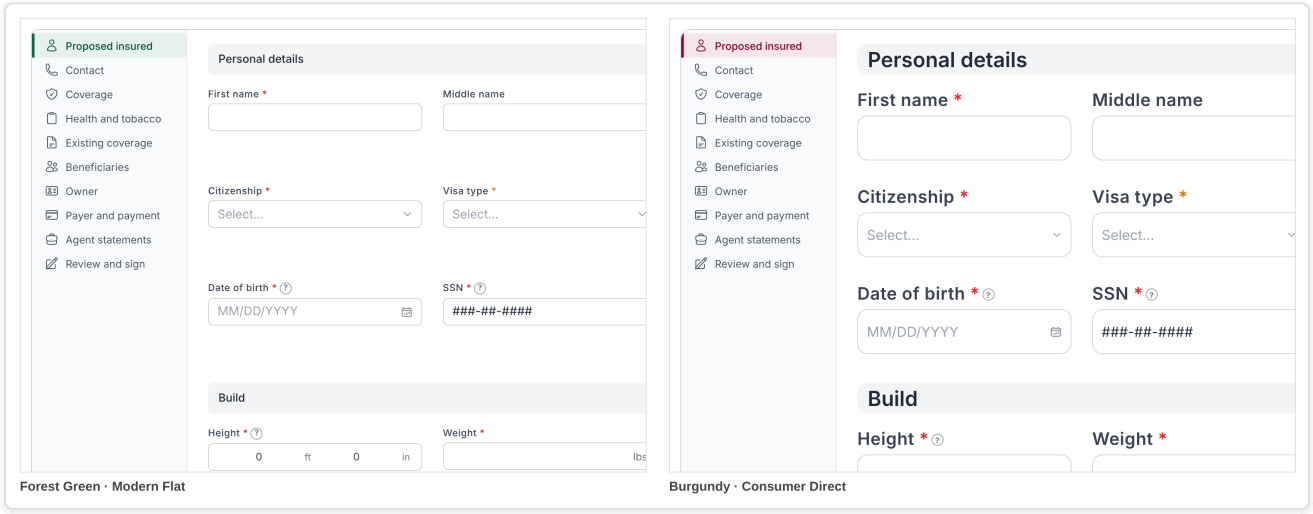
The **Style** dropdown controls the visual shape of components: border radius, shadows, typography weight, border treatment, and control sizing (input and button height). Field boxes resize automatically when you change the style, and buttons grow with the theme so a Consumer Direct primary button reads larger than a Forms Ledger one.

Theme	Personality	When to Use
Modern Flat (default)	Rounded, airy, current SaaS aesthetic	General-purpose mockups
Enterprise Classic	Dense, sharp, no-nonsense	Matching existing enterprise systems
Material	Card-based with elevation shadows	Material Design familiarity
High Contrast	Bold borders, large text, dark shadows	Accessibility-focused mockups
Forms Ledger	Forms-faithful, paper density, square corners	Form-heavy annuity/new-business platforms
Advisor SaaS	Rounded cards on a lightly tinted canvas	Modern advisor portals
Consumer Direct	Large type, tall inputs, pill buttons	Direct-to-consumer application flows
Bank Grade	Crisp borders, taller inputs, polished	Bank and bancassurance systems

The toolbar shows the words "Color" and "Style" in front of the two dropdowns only when it is wide enough to fit them beside any theme name in full, and leaves them off otherwise (in a 1440-pixel window with the rail open, for one). On a narrower toolbar still, it shortens long theme names; hover over either dropdown to see its full name.

Combining Themes

Color and style themes are fully independent. "Enterprise" color with "Modern Flat" style, or "Horizon" color with "High Contrast" style, are both valid combinations. Both are saved with your project file.



The same corner of a screen in two theme combinations. Color and style change independently, and any combination is valid.

11. Saving and Loading

Trellis saves your work as `.trellis` files. This section covers where your work lives, how saving and opening work, and what keeps your work safe in between.

Where Your Work Lives

- **This browser:** Every change is kept in the browser's storage within half a second, so a reload or an accidentally closed tab brings your project back. The browser holds one project, the one you have open, and every Trellis tab shares that one copy: work in one tab at a time, because two tabs open on two projects overwrite each other's copy, and a reload brings back whichever changed last.
- **Your `.trellis` file:** Saving writes the project to a file on your computer. On Chrome and Edge, once a project has a file, every change is also written to that file two seconds later.
- **Component packs:** Packs live in the browser too, in its IndexedDB, separately from your projects (section 12).

All of it stays on your own machine, under the Trellis site, and none of it is sent anywhere.

> **Note:** A reload brings your project back, but not its file or its name. The top bar reads Untitled; on Chrome and Edge nothing more is written to the file, and the next Save asks where to save (pick the same file to carry on in it). Until you save, Trellis also counts the restored project as saved: **New** replaces it without offering to save it first, and exports show no asterisk. After a reload, save first.

The Project Menu

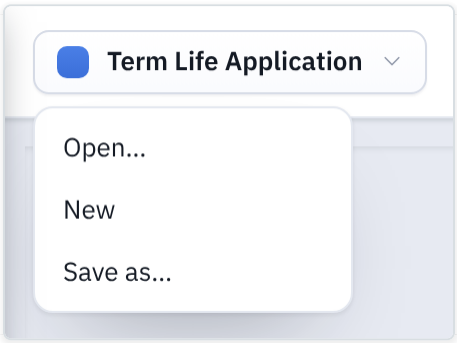
Click the project's name at the left of the top bar to open the project menu:

- **Open…** opens a `.trellis` file, or an older `.json` export. If the file was saved with an older format version, it is upgraded automatically, and anything you should know about the upgrade is shown as a brief message. On Chrome and Edge an opened `.json` file stays the project's file, written to as it is; use **Save as…** to carry on in a `.trellis` file.

- **New** starts an empty project. If the current project has unsaved changes, Trellis saves them first: to its file if it has one, or by asking where to save it if it does not (on Firefox and Safari, by downloading a copy). Cancel that save and nothing changes. The new project keeps your current color and style themes.
- **Save as…** always asks where to save, and the project then carries on in the new file: the file it had keeps what it held, and changes from then on go to the new one (on Firefox and Safari, it downloads a copy).

Home's **Open project…** and **New project** do the same, and New project then opens the empty project on Screens.

> **Tip:** Open replaces the project in this browser without asking. If that project has changes no file holds, save it first.



The project menu, under the project's name in the top bar.

Saving Your Work

Save is on the canvas toolbar. What it does depends on where you are, and its tooltip says which:

- **With no file yet** (Chrome and Edge), Save opens the save dialog, and the project keeps that file from then on, until a reload (see the note above).
- **With a file open**, Save stamps a revision (below). The file is already saving itself as you work.
- **On Firefox and Safari**, which cannot save to a file you choose, Save downloads a copy of the `.trellis` file. A banner across the top of the window says so: "Your browser doesn't support auto-save to disk. Your work is saved in this browser only. Use Save to download a copy."

If writing to the file ever fails (it was moved or deleted, say), a warning banner says so, your work stays safe in the browser, and the next Save lets you choose a new location. Closing the page while a write to your file is still pending asks before leaving.

Revisions and Unsaved Changes

Every project carries a revision number, which goes up by one each time you save after making changes: with Save, with Save as…, or in the save New makes first. Every export carries it: the image's watermark reads "Trellis r7", each spreadsheet's first line says "Revision: 7", and each JSON file has a `revision` number. When you export with changes made since the last Save, the image and the spreadsheets add an asterisk ("Trellis r7*", "Revision: 7*") and the JSON sets `dirty` to true. Click Save before exporting a deliverable you are sending out, and the asterisk goes.

The automatic writes to your file keep your work safe but do not change the revision; only a save you make does.

> **Tip:** The `.trellis` file is standard JSON under the hood. It contains your components, their positions and properties, your data, lists, forms, XML values and saved conditions, and your theme selections. You can share it with colleagues, store it in version control, or keep it alongside your requirements documentation.

12. Component Packs

Most of what you build on one platform, you build again on the next project: the same button variants, the same preconfigured fields carrying their formats and requirement levels, the same beneficiary rows and detail cards.

A **component pack** is your own library of those pieces. You save a configured component once, and it is available in every project you open in that browser. Trellis ships no packs of its own: everything in yours is something you built and named.

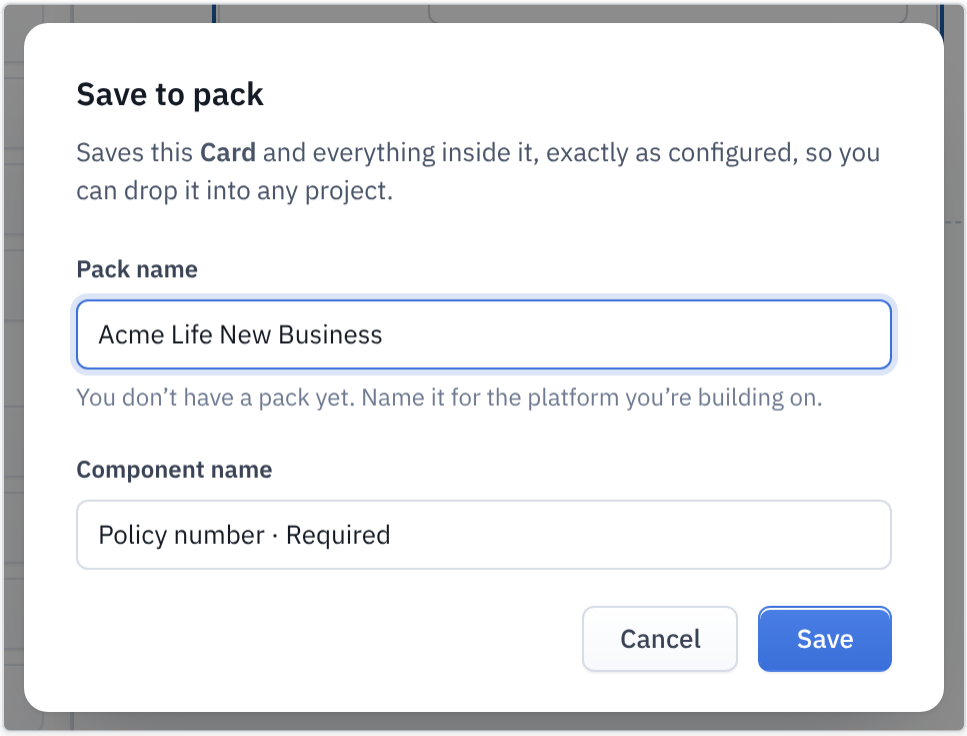
Saving a Component

Select a component and click the ⊞ button on the small toolbar that floats above it; it sits first, before duplicate and delete. There is also a **Save to Pack** button at the foot of the Properties Panel.

Whatever you save comes with everything inside it. Save a Card and you save the fields in it, their labels, their formats, their requirement levels, exactly as you configured them. What a saved component does not carry is data: a pack holds shapes, not a project's data, so a field saved to a pack keeps everything it shows but not its data name, party, XML values or forms. A saved component keeps the conditions that read fields inside it, each reading the stamped copy's own fields; a condition that reads anything outside what you saved is left out, and the toast after the save counts them. A pack holds no saved conditions: a slot that used one keeps a copy of its rules.

The first time you save, Trellis asks for two things:

- **Pack name:** Name it after the platform you are building on, not after the project. A pack is a design system's worth of components, and you will point many projects at it.
- **Component name:** Name it for what it is, not where you used it. `Delete · Secondary · Icon` will still make sense in six months; `Button from the beneficiary screen` will not.



The first save also creates the pack. After that, only the component name is asked for.

After the first save, only the component name is asked for. New components go into whichever pack is currently active.

Trellis refuses a save in a few cases, and says which:

- **A name already used in that pack.** Names have to be unique so you can tell two saved components apart.
- **A component that is currently rendering nothing:** a Modal Dialog with **Show on the canvas** switched off, or a parked Detail Panel. Its preview picture would be blank, so show or open it first and save again. A collapsed Collapsible Frame still draws its header, so it saves normally.
- **A full pack.** A pack holds 200 components. The limit counts saved components, not the pieces inside them.

The **Save to Pack** button is visible but grayed out when you have more than one component selected. Save the container that holds them instead.

> **Tip:** In the pack tab's pictures, names are never truncated, however long they are and however narrow you leave the panel; in its list view a long name is cut short, and hovering over it shows it whole. Long, descriptive names cost nothing, which is why a naming convention pays off here more than it usually does.

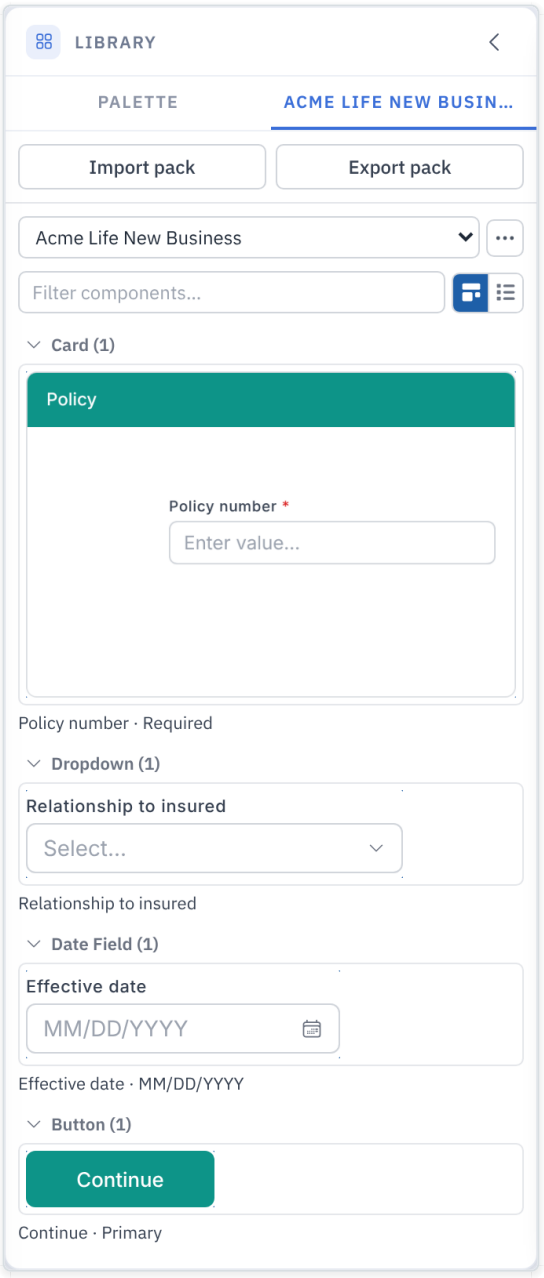
Linked Fields

If a saved component contains a field and something pointing at it, such as an Inline Error with its **Linked field** set, the link travels with it. When you stamp a copy, the link re-points itself to the stamped copy's own field, not to the original.

If you save only one half of such a pair, Trellis warns you before saving and names the link it has to drop. The field being pointed at is not coming along, so the link cannot survive. The saved component is fine; it simply arrives without that link.

The Pack Tab

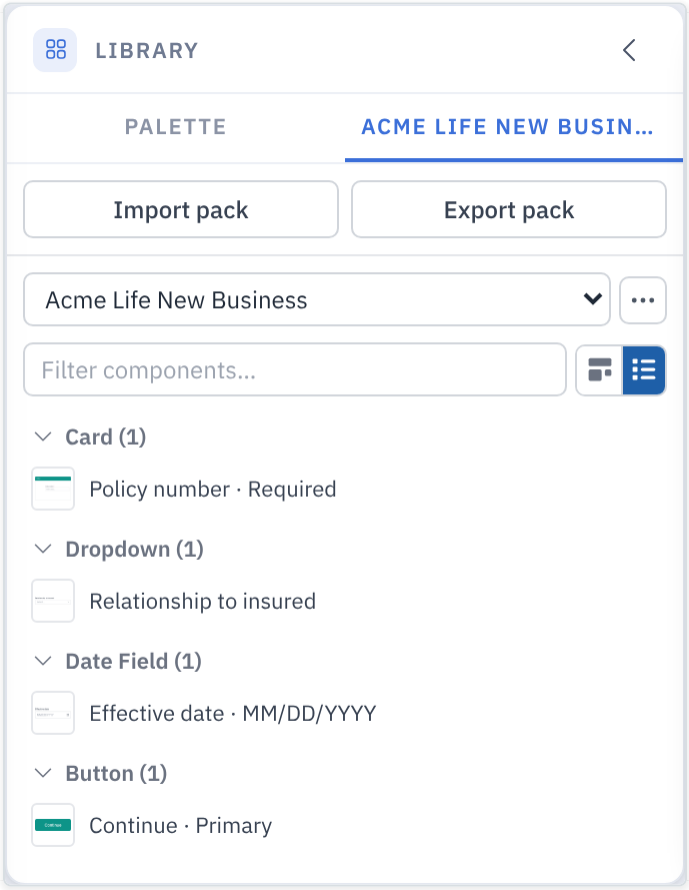
The library has two tabs: **Palette**, the components Trellis ships, and one named after your active pack (**My Components** until you have one).



Saved components are grouped by type and drawn at their real size, so a button looks like a button and a card looks like a card.

Your saved components are grouped by type with a count per group, and each is drawn at the size you built it; nothing is squeezed into a uniform tile. Hovering over one shows a larger preview to the right, which mostly matters for anything too wide to fit the panel.

The two small buttons beside the filter box switch views. The second is a dense list (small icon, name only) for when a pack gets big enough that you want to scan names rather than look at pictures.



The list view trades the pictures for density. Your choice is remembered.

Drag the library's right edge to widen it, from its starting 200 pixels up to 480; double-click the edge to snap back. As it widens, large components grow toward their real size while small ones stay exactly as they are; nothing is blown up to fill space. The library remembers its width, its tab and its view.

Stamping a Component

Drag a saved component from the pack tab onto the canvas, or into any container that accepts it: the same drag you would use for a component from the Palette.

What lands is an ordinary component. It is not linked to the pack, and not linked to the original you saved from: rename its fields, resize it, delete the original, delete the whole pack, and the stamped copy is unaffected. Its fields arrive with no data, so name their data in the Properties Panel as you would for a field you built by hand (section 8); from then on it behaves in every way like one you built by hand, including in your exports, and its conditions read the stamped copy's fields.

Dropping into a container puts the component in that container's current view. Drop into the second tab of a Tab Container and it belongs to that tab.

Packs and Projects

A project file records **which pack it uses**, by reference. It does not contain the pack. That has two consequences worth knowing:

- Opening a project on a machine that does not have the pack still opens the project. Every stamped component is already an ordinary part of the canvas and renders normally. Only the pack tab notices anything is missing.
- Sharing a project with a colleague who should also get your components is a **two-file handoff**: send the `.trellis` project and the `.trellispack` pack.

Backing Up, Sharing and Importing

Export pack downloads the whole pack, components and their previews, as a single `.trellispack` file. **Import pack** reads one back in. Both buttons sit at the top of the pack tab.

Because packs live in your browser rather than in your project files, that exported file is your only backup. Clearing your browser data removes your packs and leaves your saved project files untouched. After ten saves without an export, the panel reminds you.

An imported pack becomes the active pack. Importing one that is already in your library does not replace it: the import arrives as a second pack, its name followed by a number ("Acme Life (2)"), and the pack you had stays as it was.

The ⋯ button beside the pack selector holds **Rename pack…**, **New pack…** and **Delete pack…**. They are deliberately kept out of the row with Import and Export, so that reaching for a backup cannot land on a delete. Deleting a pack or a saved component never changes any canvas: the components you already stamped are yours.

When the Pack Tab Can't Show You a Pack

Five things it may say instead of a list:

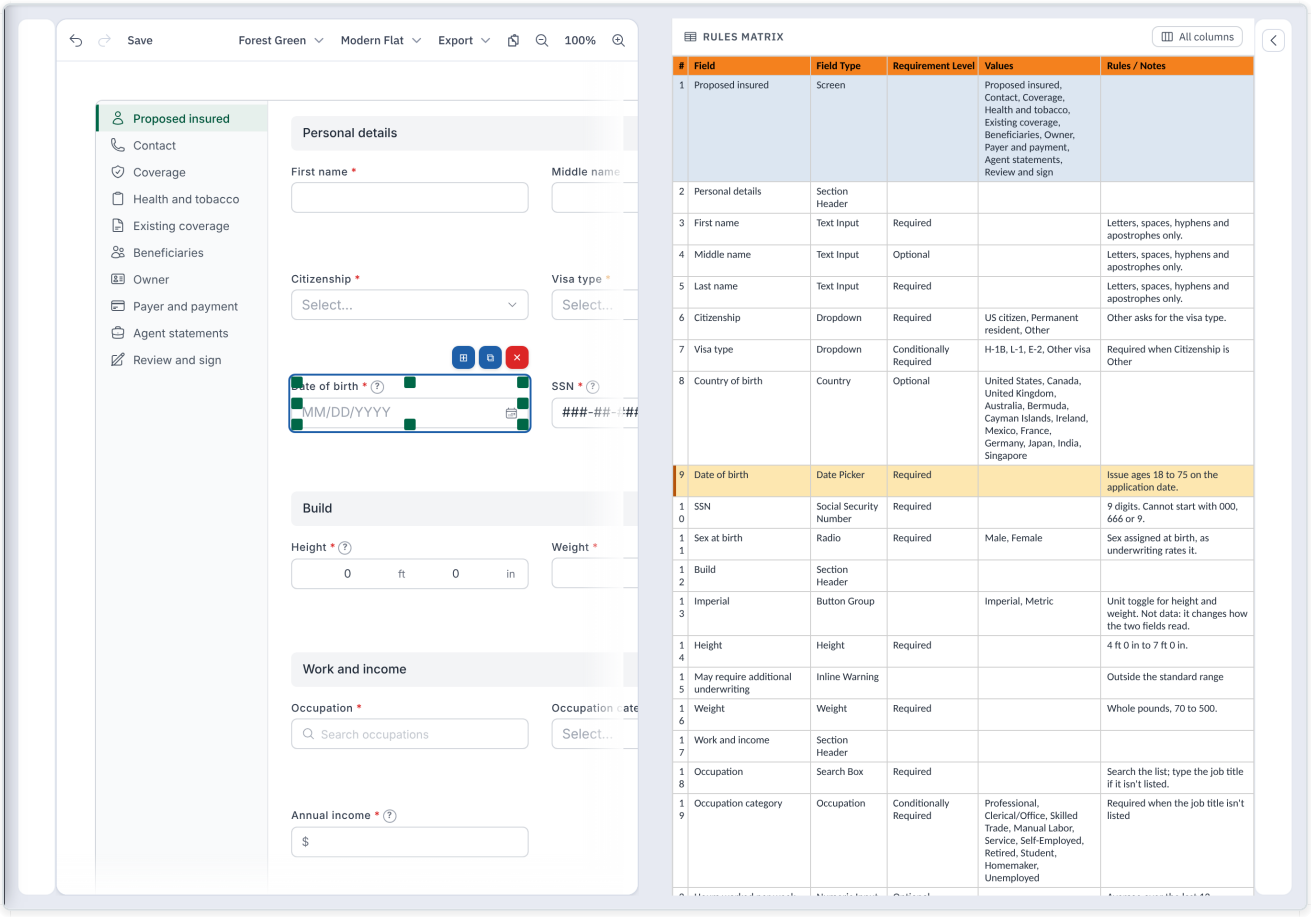
- **"No pack selected."** You have not made one yet, or the project does not point at one. Save any component to create one, import a pack file, or choose one of your packs.
- **"Acme Life is empty."** (with your pack's name) The pack exists and has nothing in it yet.

- **"Acme Life isn't in your library."** The project points at a pack this browser does not have, usually because it was saved on another machine or in another browser. **Import pack…** reads the pack file in, and **Use another pack…** picks one you have. Your canvas is untouched either way.
- **"Acme Life is in your library but can't be opened."** The pack is there, but its stored copy cannot be read; the tab says why. **Import a replacement…** reads a pack file in and points the project at it, or **Use another pack…** picks a different one.
- **Storage is blocked.** The tab says packs need browser storage, which this session is blocking (private-browsing windows usually do). The save buttons and Import pack are grayed out, and everything else in Trellis (canvas, saving, exports) works normally.

> **Note:** Packs are stored in your browser's IndexedDB, separately from the local storage your project auto-save uses, because saved components include preview images that would quickly exhaust the smaller budget. Both are on your own machine, under the Trellis site, and neither is sent anywhere. See section 11 for how the three places your work can live relate to each other.

13. Review

Review in the rail puts the Rules Matrix beside the canvas: a live, spreadsheet-like view of the Rules Grid for the screen you are on, built from the same rows and columns as the Rules Grid export. Use it to walk a client through a screen's rules, or to check your own, and change things as you go.



Review on the demo's Proposed insured screen. Clicking the Date of birth row selected the field on the canvas.

- **Rows:** One row per component on the screen in view, in screen order, with a row under a field for each of its inline feedback messages, colored as the Rules Grid is: blue for containers, pink for fields flagged with Missing requirements. Rules / Notes prints a line for each slot that holds something before the notes, as the Rules Grid does (section 14): the demo's Visa type reads "Required when Citizenship is Other".

- **Columns:** Six by default: #, Field, Field Type, Requirement Level, Values and Rules / Notes. **All columns** shows every column of the Rules Grid, scrolling sideways; **Main columns**, as the button then reads, goes back to the six.
- **Selection:** Selecting a component on the canvas highlights its row and scrolls to it, and a selected container tints the rows of what it holds. Clicking a row selects its component on the canvas and scrolls to it; a message's row selects its field. If the selected component is not on the screen in view, the matrix says so.
- **Layout:** Drag the divider between the canvas and the matrix to share the width differently. In Review the Properties Panel starts as a thin strip at the right edge: click its chevron to open the panel, and the chevron at the panel's edge to put it back. Trellis remembers both in this browser.

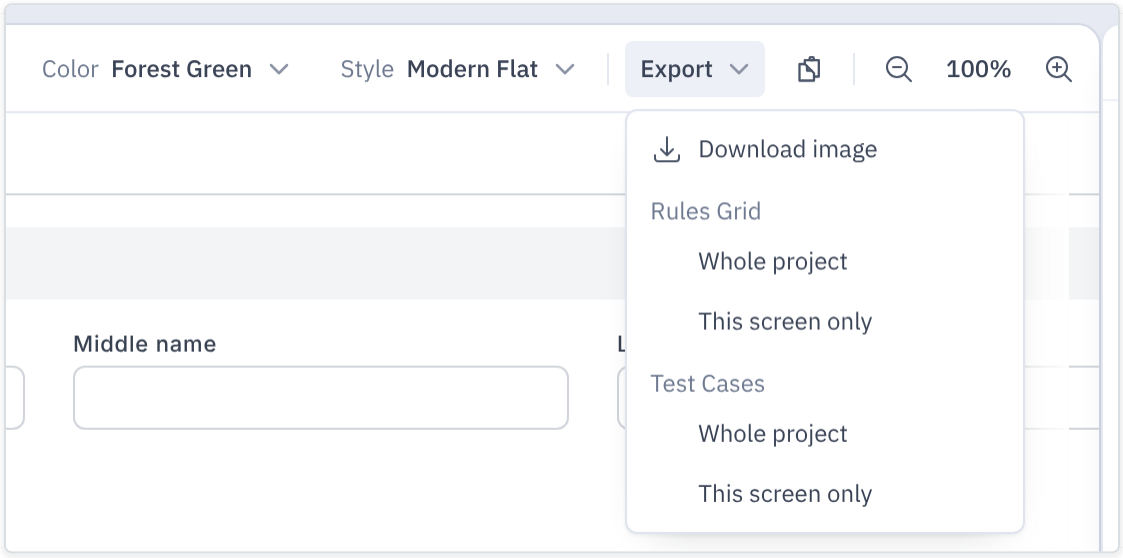
The canvas works as it does on Screens: select, move, edit in the Properties Panel once it is open, undo, copy and paste, and the matrix updates as you do. The library is put away in Review, so add new components on Screens.

14. Exports

Trellis produces seven exports, each serving a different downstream need: an image of the screen, the Field Inventory, the Rules Grid, the Test Cases, the Data map, the Forms matrix and the XML mapping.

Where to Find Them

- **Home** has one card per project export: Field Inventory (XLSX, JSON), Rules Grid (XLSX), Test Cases (XLSX), Data map (XLSX, JSON), Forms matrix (XLSX) and XML mapping (XLSX). Home's Rules Grid and Test Cases cover the whole project, and the Forms matrix and the XML mapping are only ever the whole project.
- **The canvas toolbar's Export menu** holds **Download image** (a PNG of the screen in view) and, under **Rules Grid** and under **Test Cases**, **Whole project** and **This screen only**.
- **The copy button** beside Export (its tooltip reads "Copy an image of the canvas to the clipboard") copies the same PNG as Download image to your clipboard, for pasting into documents, slides or chat.



The Export menu on the canvas toolbar.

Whole project covers every tab, screen, and step, open or not. **This screen only** covers the screen, tab or step you are on in each container, with every Detail Panel and Modal Dialog on it,

open or parked. In both, every row of the Rules Grid and the Test Cases says which screen it is on and which tabs or steps.

Every spreadsheet starts with a line naming the project, the revision and the scope, for example "Trellis | Term Life Application | Revision: 7 | Whole project", and every file is named after the project, for example `trellis-term-life-application-rules-grid-whole-project.xlsx`. With changes made since the last Save, the revision carries an asterisk, and the JSON files say so with `"dirty": true` (section 11).

Image (PNG)

What it is: A high-resolution picture of the screen in view, cropped to your content with a subtle revision watermark.

When to use it: Stakeholder reviews, email attachments, slide decks, requirements documents where a visual reference is needed.

Output: A 2x resolution PNG cropped to the bounding box of everything on the canvas, with 24px of padding. Only the open tab, screen, or step of each container is drawn, and the authoring chips for Detail Panels and hidden Modal Dialogs are left out. A faint watermark in the bottom-right corner shows the revision number.

Field Inventory (JSON)

What it is: A structured data file listing every field in your project with its properties, organized by section.

When to use it: Developer handoff where the downstream team needs machine-readable field definitions to build from.

Output: A `.json` file containing the revision, `dirty` (whether there were unsaved changes), and an array of sections, each with its fields. Per field: label, type, component type, tooltip, its inline feedback messages, format, requirement level, disabled state, default value, validation bounds, a hyperlink's URL and target, the data name and party where the field has data, and, where it holds one, its Required when (`requiredWhen`), its Shown when (`shownWhen`) and each option's Offered when (`offeredWhen`), each its sentence (`text`) with the condition beside it (`logic`), or the words as typed; a message's When prints its sentence, or its words, as `when` , with `logic` beside it on the message.

Field Inventory (Excel)

What it is: The same field inventory as the JSON export, formatted as a readable spreadsheet.

When to use it: BA documentation, requirements review meetings, attaching to Jira tickets or SharePoint.

Output: An `.xlsx` file with the first line, column headers, and one row per field. Columns: Section, Label, Field Type, Component Type, Tooltip, Format, Requirement Level, Disabled, Default Value, Min Length, Max Length, Min, Max, Date Direction, URL, Target, Notes (left blank for developers), Data Name and Party.

Rules Grid (Excel)

What it is: A comprehensive validation and business rules spreadsheet with color-coded rows for containers and for fields flagged with Missing requirements. Review shows the same grid live (section 13).

When to use it: Developer handoff (especially for teams building the screen manually, where it serves as the primary build reference), client validation against their business rules, QA handoff, and compliance review. The Rules Grid is where your Rules and Notes surface.

Output: An `.xlsx` file with frozen header rows and one row per component, in screen order, with a row under a field for each of its inline feedback messages. Columns: #, Field, Field Type, Component Type, Tooltip, Format, Requirement Level, Disabled, Field Length, Values, Min Length, Max Length, Min, Max, Date Direction, Rules / Notes, Missing Requirements, Screen, and Tab or Step. Rows are color-coded:

- Orange header row
- Blue rows for containers (screens, tabs, frames)
- Pink rows for fields flagged with Missing requirements
- White rows for standard fields

The Rules / Notes column holds, in order, the auto-generated default (like "Default: Monthly" for a dropdown with a pre-selected value), a line for each slot that holds something ("Monthly: offered when Paying by bank draft (Payment method is Bank draft)" on the demo's Premium frequency), then your notes as typed.

	A	B	C	D	E	F	G
1	Trellis | Term Life Application | Revision: 2 | Whole project						
2							
3	#	Field	Field Type	Component Type	Tooltip	Format	Requirement Level
4	1	Proposed insured	Screen				
5	2	Personal details	Section Header				
6	3	First name	Text Input	Text Box			Required
7	4	Middle name	Text Input	Text Box			Optional
8	5	Last name	Text Input	Text Box			Required
9	6	Citizenship	Dropdown	Dropdown			Required
10	7	Visa type	Dropdown	Dropdown			Conditionally Required
11	8	Country of birth	Country	Dropdown			Optional
12	9	Date of birth	Date Picker	Date Picker	Age at issue comes from this date.	MM/DD/YYYY	Required
13	10	SSN	Social Security Number	Masked Input	Used for underwriting and tax reporti	XXX-XX-XXXX	Required
14	11	Sex at birth	Radio	Radio			Required
15	12	Build	Section Header				
16	13	Imperial	Button Group				
17	14	Height	Height	Numeric Input	Height without shoes.	X'XX"	Required
18	15	May require additional underwriting	Inline Warning	Inline Warning			
19	16	Weight	Weight	Numeric Input		XXX lbs	Required
20	17	Work and income	Section Header				
21	18	Occupation	Search Box	Search Box			Required
22	19	Occupation category	Occupation	Dropdown			Conditionally Required

The top of the demo's Rules Grid, drawn from the export's own cells: the first line names the project, the revision and the scope, orange marks the column headers, and blue marks the screens.

Test Cases (Excel)

What it is: A QA spreadsheet of test scenarios for every field someone fills in, each with a test input, an expected result, and a priority.

When to use it: Building test plans. Trellis writes the standard scenarios a field's settings imply; your QA team adds the ones only they know.

Output: An `.xlsx` file with frozen headers. Columns: #, Field, Field Type, Test Type, Scenario, Test Input, Expected Result, Priority, Screen, and Tab or Step. The fields it covers are the input fields of section 8 except Search Input, and File Upload and E-Signature as well. For each one:

- A **Valid** standard input that is accepted (P1).
- For Required and Conditionally Required fields, a **Required** empty submission that is refused (P1), except where a Required when holds a condition (below).
- **Invalid** inputs for fields with a format, such as a malformed email address, phone number, SSN, postal code, date or time, or text in a number field (P1).
- **Boundary** rows at and past each minimum and maximum length or value (P2), and **Constraint** rows for date directions and earliest and latest times (P2).

- One **Valid** row per option for choice fields, and Checked and Unchecked for checkboxes and toggles (P2).
- One **Feedback** row per inline feedback message on any field (section 7), unless its When holds a condition (below): its When as the scenario, or "Condition met" where none is written, and the message as the expected result, such as "Error: Must be at least $25,000" (P1 for an Error, P2 for the rest).
- One or more **Condition** rows for each slot on a screen that holds a condition (section 9): the Scenario the answers that make the row, the Test Input "(per scenario)", and the Expected Result what the screen does: "Error: Required field, on an empty submission", "Accepted, on an empty submission", "Shown", "Hidden", "Opens", "Stays closed", "Offered: Monthly", "Not offered: Monthly", the message, or "No message" (P1 for a Required when and an Error's rows, P2 for the rest). They take the place of the field's Required row, or of a message's Feedback row; a slot of words keeps the row it had.

Data Map (Excel and JSON)

What it is: One row per piece of data in the project: its name and party, its type, whether it is required, the screens that collect it, the forms that print it and the XML values it feeds.

When to use it: Before changing or deleting a field, a form or an XML value, when you need the whole picture on paper. Developer handoff for the XML file and the forms. Finding data that is collected but goes nowhere.

Output: An `.xlsx` file with one sheet, "Data Map": the first line and seven columns: Data Name, Party, Type, Required, Screens, Forms, XML Values. Required reads Yes, No, Conditionally, or Varies when the fields that collect the data disagree. Screens are listed in the order you see them, forms by name with their number, and XML values by their path; a cell with more than one lists them separated by semicolons. The `.json` file carries the same rows and, after them, the project's lists, its saved conditions, what decides each Applies to row, and its forms and XML values with every setting they have; a when that holds a condition prints its sentence as `text`, its `logic` beside it naming fields, options and saved conditions by name. Both are on the Data map card on Home.

Forms Matrix (Excel)

What it is: Every form in the project in package order, with its signers, its editions and the data printed on it: the Forms view's four tabs as four sheets, Forms, Signers, Editions and Data on forms.

When to use it: Handing the forms over to whoever builds the print package. Checking each form's settings, signers and editions with a client, outside Trellis.

Output: An `.xlsx` file named after the project, for example `trellis-term-life-application-forms-matrix.xlsx`. Every sheet starts with the project's first line; the Forms sheet's second line says what decides products, languages and provinces or states, for example "Products decided by Plan · Primary insured." Each sheet's columns are its tab's, every one of them, with the words the tab shows: a cell the tab leaves at Not set is empty, a list names every item, and each signer, edition and piece of data names its form. The first three rows stay in place as you scroll. It is on the Forms matrix card on Home.

XML Mapping (Excel)

What it is: Every XML value by party, with its codes, and the messages when the project has any: the XML view's two tabs as two sheets, XML values and Codes, and a Messages sheet as the Lists page lists them.

When to use it: Developer handoff for the new business XML file: what is sent, for whom, in what form, and as which codes.

Output: An `.xlsx` file named after the project, for example `trellis-term-life-application-xml-mapping.xlsx`. Every sheet starts with the project's first line. The XML values sheet has a row per XML value and party, Message first ("All", or the messages it is in), then every setting the XML values tab has, with Data and Fields naming every piece of data and field; a setting that does not apply is empty. The Codes sheet lists each option of a value of type Code with its code, as typed. The first three rows stay in place as you scroll. It is on the XML mapping card on Home.

15. The Dependency Map

The Dependency Map shows how one thing in your project connects to everything else (screens, fields, data, XML values and forms), what decides it, and what deleting it would touch. It shows and never changes anything; the line at the foot of the map says so: "The map shows. It never changes anything."

Opening the map

Click **Dependency Map** in the rail. The map opens on the whole project, centered on nothing: its gaps to the left and its screens to the right. Click a screen to center on it.

- **Gaps** are what is missing: fields with no data, data with no XML value, data left behind (no field carries it any more, though an XML value or a form still does), and XML values and forms nothing uses. An XML value whose value is fixed or calculated needs no data, so it is not one. A form with no Included when, a form with a signer with no Signs when, and an XML value with a row with no Sent when (each Conditionally Required with neither words nor a condition in its when), an XML value with an option that has no code, and data that decides Products or Languages but whose options carry none of the list's items are gaps too, each on its own card. A Conditionally Required field with no Required when, anything with a broken condition, and conditions that depend on each other are gaps too. A File Upload and an E-Signature have no card, so an empty Required when on one is not.

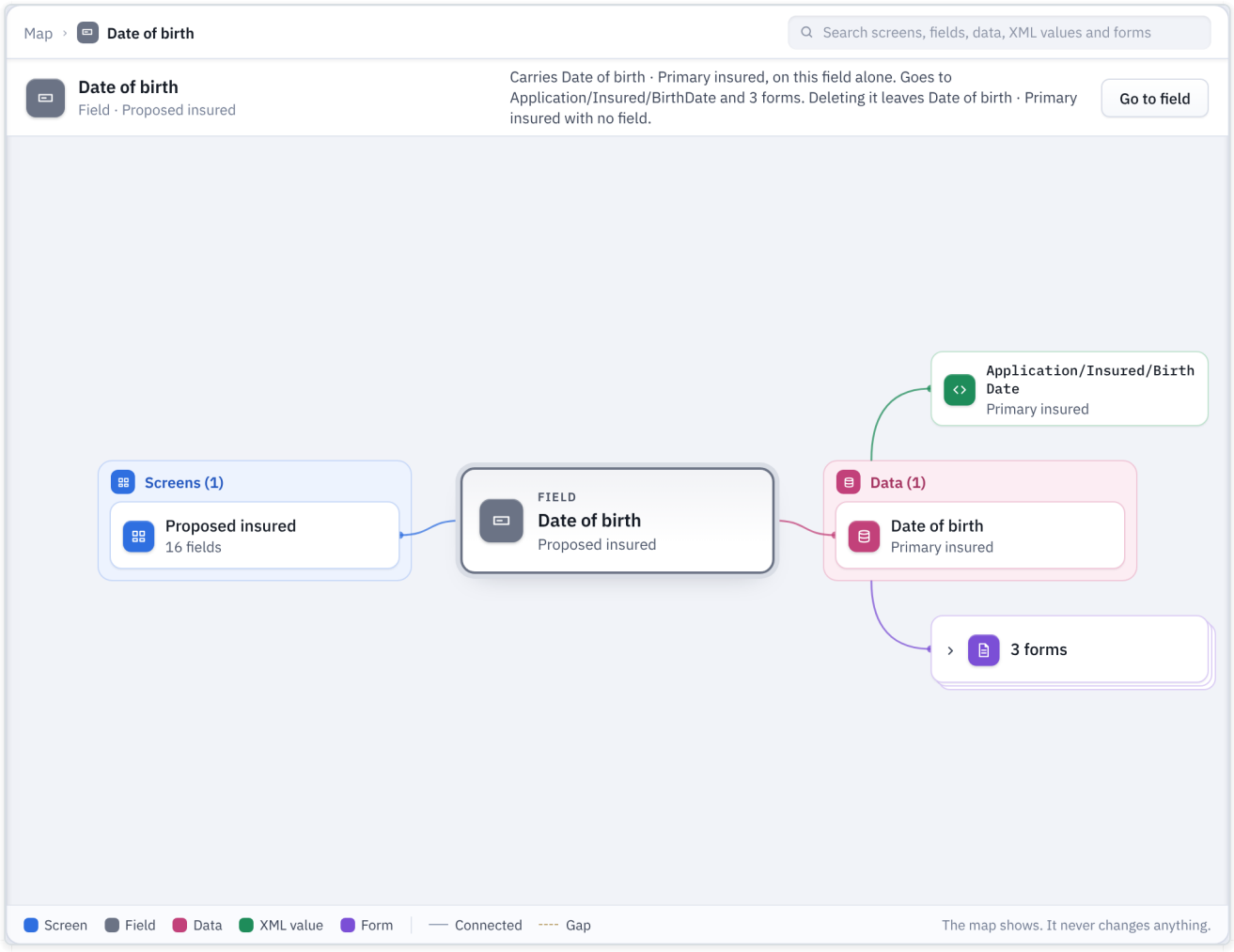
Each card is a count that opens to show what it counts, or one thing you can center on.

Centering on something

The map centers on one thing at a time: a screen, a field, a piece of data, an XML value or a form. There are four ways in:

- **Search screens, fields, data, XML values and forms**, the search at the top of the map. Type part of a name and pick a result with the mouse or with Enter.
- **Show on map**, in a field's Data section and on every row of the Forms tab and the XML values tab.
- A card's name, anywhere on the map.

- A name in the list a USED BY count opens, in the Forms and XML tables.



The map centered on the demo's Date of birth field, from Show on map in its Data section. The strip under its name says what it carries, where it goes and what deleting it would leave behind.

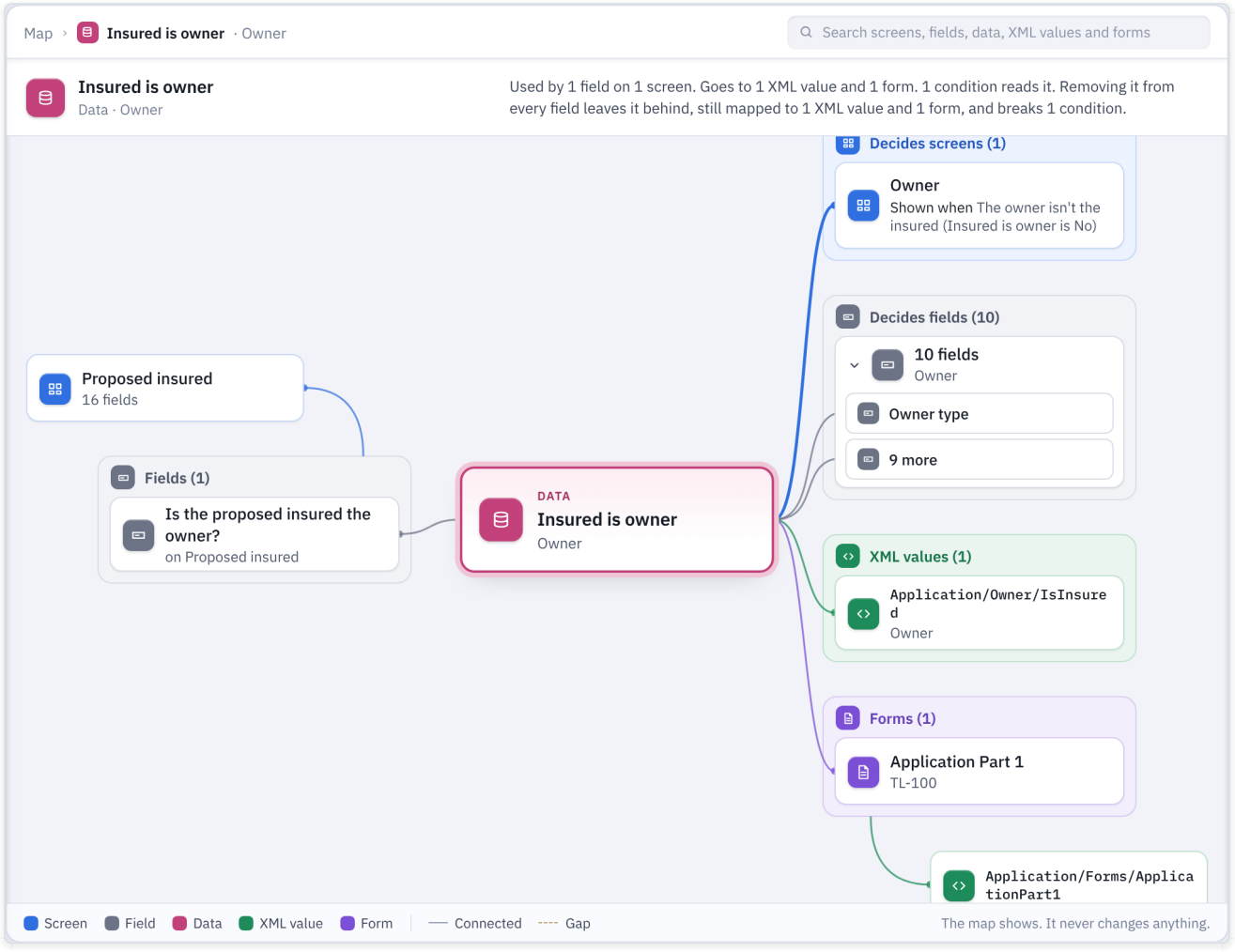
Reading the map

What uses the centered thing sits to its left, and where it goes to its right, out to two steps away. The strip under its name says, one sentence each, what uses it, where it goes, how many conditions read it, and what deleting it would touch, including a piece of data that would go with it because nothing else would be left on it, and the conditions it would break.

Conditions draw as groups of their own, first on their side. On the right of a piece of data, or of a field with no data, **Decides screens**, **Decides fields**, **Decides XML values** and **Decides forms** hold what the conditions that read it decide; on the left of what a condition decides, **Decided by fields** and **Decided by data** hold what it reads. The centered thing prints its own conditions under its

name, and a card under a Decides heading prints the lines that read the centered thing, as Rules / Notes prints them: centered on the demo's Insured is owner, the Owner screen's card reads "Shown when The owner isn't the insured (Insured is owner is No)". A frame's, panel's or dialog's condition shows on each field inside it, named for it.

Next to the centered thing, a group of more than eight cards folds into one card per part: data by party, and fields by screen, or, around a screen, into the fields with data and the fields with none. Screens, XML values and forms past eight show seven cards and a card saying how many more. Two steps out, three or more of a kind fold into one card (two, where they would not otherwise fit). A group that would show a single folded card shows it open, and a group taller than the map ends in a card saying how many more. Where even that leaves a side taller than the map, the side is laid out whole and the map scrolls inside its frame, opening with the centered thing in the middle. Two cards that would read the same are told apart by a number. A piece of data collected on more than one screen reads Shared. The legend marks each line between cards as **Connected** or as a **Gap**.



The map centered on the demo's Insured is owner. Under Decides screens, the Owner screen's card reads its Shown when, and the strip says how many conditions read it.

Leaving the map

For a screen, a field, an XML value or a form, the strip has one way out. **Go to screen** opens the canvas on that screen, and **Go to field** opens it on that field, selected, through tabs, steps, collapsed frames and closed panels, as the top bar's search does. **Open in the XML list** and **Open in the Forms list** open that view on its row, on the tab that row is on: a form whose signer has no Signs when opens on Signers, and an XML value with an option that has no code on Codes. A piece of data that decides a row of Applies to has **Open in Lists**, which opens the Lists page on that row; other data has no way out. To go back to the whole project, click **Map** at the left of the bar above the map, or press Escape.

16. Keyboard Shortcuts

Shortcut	Action
Cmd/Ctrl + Z	Undo (on every view)
Cmd/Ctrl + Shift + Z	Redo
Cmd/Ctrl + Y	Redo (alternate)
Cmd/Ctrl + K	Jump to the search in the top bar
Cmd/Ctrl + C	Copy selected components
Cmd/Ctrl + V	Paste
Escape	Clear selection; close the Export menu, the search results, the library opened from + Insert, or the condition builder without saving (an open pick first); on the map, go back to the whole project
Enter or F2	On a when cell or a Reads as cell that holds a condition: open the builder
Arrow keys	Nudge selected components 1px
Shift + Arrow keys	Nudge selected components 8px
Ctrl/Cmd + Click	Toggle component in/out of multi-selection
Up/Down, then Enter	Move through search results and open one

> **Note:** Keyboard shortcuts are suppressed when you are typing in a text field, textarea, or dropdown. Copy, paste, nudge and Escape's clear act on the canvas, on Screens and in Review; undo and redo work on every view.

17. Known Limitations

Trellis is in beta. These are current limitations to be aware of:

- **No keyboard Delete/Backspace.** Use the delete button (✕) in the floating toolbar above a selected component, or the Delete button at the bottom of the Properties Panel.
- **No drag-out of containers.** To move a component from one container to another, copy it, paste it into the new container, and delete the original.
- **Group duplicate is not currently supported.** Multi-select supports moving, copying, arrow-key nudging, and deleting (Delete Selected, in the Properties Panel), but the floating toolbar and its duplicate button only appear for single selections.
- **Silent invalid drops.** Dropping a component onto a surface that does not accept it does nothing (no error message).
- **Validation properties are export-only.** Min/max length, min/max value, date direction, and the other rules appear in your exports but do not enforce anything on the canvas. Trellis is a requirements authoring tool, not a runtime form engine. Conditions decide nothing on the canvas either: Trellis never shows, hides, opens, requires or offers anything by one.
- **What a condition can't say yet** is listed in section 9.
- **No card on the map for a File Upload, an E-Signature or a Stepper's step**, so their conditions show nowhere there.
- **Desktop browsers only.** On a phone, trellismade.com opens a tour of the product instead of the app. Use a desktop browser (Chrome or Edge recommended for full save-to-file support).
- **A reload forgets the project's file.** The project comes back, but reads Untitled, is no longer written to its file, and counts as saved until you save it again, so New will not offer to save it first. Save after a reload (section 11).
- **One project per browser.** Every Trellis tab shares the browser's one copy of the open project, so two tabs open on two projects overwrite each other's copy. Work in one tab at a time.
- **No cloud save or accounts.** All data lives in your browser and local files. Share a project by sending the `.trellis` file directly. If the recipient should also get your saved components, that is a two-file handoff: send the `.trellispack` file too (section 12).
- **Packs are per-browser.** Your component packs live in the browser you saved them in, not in your project files and not on another machine. Exporting a pack to a `.trellispack` file is the

only way to back it up or move it, and clearing your browser data removes them. Private-browsing windows usually block the storage packs need. See section 12 for the whole picture.

Trellis is created by Nick Picarella.