Repairs Capacity Building

Catalog Builder

User Guide V1.0

Table of Contents

About This App & Guide

This guide explains how to use the **Repairs Catalog Builder** app to create a clear, consistent set of repair activity decisions that can be exported into a policy manual, shared internally, and reviewed and expanded over time. The app is designed to help users move from learning the taxonomy and prioritization model to making explicit eligibility decisions at the activity level, then exporting those decisions in practical formats (text, CSV, PDF).

The guide is **app-centered**. That means:

- It follows the actual workflow and screens users will see (Learn → Builder → Export).
- It defines concepts only to the extent they help users make correct decisions in the Builder and produce usable exports.
- It emphasizes repeatable process and consistency, so the exported outputs are defensible and comparable across time and across teams.

What success looks like

A user has "successfully used" the app when they can:

1. Navigate the app and understand what each mode is for (Learn, Builder, Export).
2. Make consistent decisions on activities using the status labels (Eligible, Not Eligible/Ineligible, If/When, N/A).
3. Use notes and overrides correctly, without breaking comparability.
4. Export outputs that can be pasted into a policy manual or analyzed in a spreadsheet.

1.1 Intended Audience

This guide is written for users who need to define, document, or apply a home repair scope policy using the Repairs Catalog Builder.

Primary audiences:

- **Repairs Program Managers / Program Directors**
  Responsible for defining scope, aligning delivery to funding requirements, and maintaining a stable policy over time.
- **Construction Leadership (Construction Managers, Project Managers, Inspectors)**
  Need a standardized activity list and prioritization logic to reduce ambiguity and improve consistency between assessment, scoping, and delivery.
- **Operations and Compliance Staff**
  Need exported outputs that can be incorporated into manuals, grant compliance documentation, and internal process documentation.
- **Affiliate Leadership / Board Representatives (Reviewers and Approvers)**
  Need a defensible, readable export of what the program will and will not do, including conditional "If/When" logic.

Secondary audiences:

- **Partner organizations and funders** who review scope documentation.
- **Trainers / TA providers** who support program development and want a consistent structure for coaching.

Assumed skill level:

- Users do not need technical skills. They do need basic familiarity with home repair programs and the idea of standardized categories and prioritization.

1.2 What This App Produces

The app produces a local, explicit decision set for repair activities organized by the Housing Preservation Framework taxonomy:

Category (Pillar) → Sub-Category (SC) → SC Type → Activity

For each Activity, the user records:

- A **Status decision** (Eligible, Not Eligible/Ineligible, If/When, N/A)
- *Optional* **Notes** (especially required in practice for If/When decisions)
- *Optional* **Urgency and Condition overrides**, which affect the Activity's priority label

Export formats (what users walk away with)

The Export mode supports three usable deliverables:

1. **Copy Text**
   Used to paste directly into a policy manual or operating procedures document.
2. **CSV Export**
   Used for spreadsheet review, QA, aggregation, and internal analysis. The export includes column fields such as Pillar, Sub-Category, Type, Activity Name, Selected Status, Priority Label, Final Urgency, Final Condition, and Notes.
3. **Print / Save PDF**
   Used to generate a clean, appendix-style policy output. The PDF structure groups outputs into Eligible Repairs, Conditional Repairs, and Non-Eligible Activities and is formatted as "Appendix A: Construction Activities."

What the app is not

To prevent users from expecting the wrong thing:

- The app does not perform an inspection, diagnosis, or scope for a specific home.
- The app does not replace local eligibility criteria, income qualification, or construction standards.
- The app does not function as a database or shared multi-user system. It saves work locally in the browser.

1.3 Versioning and Change Notes

This guide should be versioned alongside the app, because users will rely on the guide to interpret what exports mean and how to apply labels consistently.

Recommended versioning approach

As you develop this out, or as funding or team capacity and capabilities expand (or contract), you may need to revisit your program activity-level commitments. When performing this review, we recommend adopting a version control of your eligible activities in the event they are disputed from one program year to another.

At minimum, we recommend you track:

- **Export Version:** Track app version as well as export options may change
- **Validity Date(s):** When the version became active and expires
- **Change Summary:** What changed that affects end users
- **Compatibility Notes:** Whether older exports or components of, remain valid

What belongs in change notes

Only include changes that affect user behavior or interpretation, such as:

- New or renamed **Status labels** (Eligible / Ineligible / If/When / N/A)
- Any changes to the **export schema** (CSV column names, PDF headings, grouping logic)
- Updates to the **priority matrix logic** or how urgency/condition are applied
- Any changes to how the app **saves** or restores work (Export options may change within the app so track app version)

Practical warning you should include in version notes

Because the app stores selections locally in the browser, some updates may appear to "reset" work if users clear cache or change devices. Change notes should remind users to export CSV/PDF regularly to preserve a record.

Quick Start (10-minute read)

This app is built around a simple workflow: **Learn → Build → Export**. Do it in that order and you end up with a usable, defensible policy output.

2.1 Learn

Purpose: The Learning Guide page gives users a shared baseline for how the catalog is organized and how prioritization works before anyone starts making decisions.

When to use Learning Guide

Use Learning Guide when:

- You are new to the catalog or the taxonomy structure.
- You need consistency across multiple staff making decisions.
- You plan to override urgency/condition and need to understand what that implies.

What you do in Learning Guide

1. Open the app and go to Learn.
2. Work through the learning modules (Four Pillars, Taxonomy, Deep Dive, Criticality Matrix) in order.
3. Confirm you understand:
    a. The hierarchy: Pillar → Sub-Category → Type → Activity
    b. The prioritization model: Urgency + Condition → Priority

What "done" looks like

You are ready to move on when you can explain, in plain language:

- What each status means (Eligible, If/When, Not Eligible, N/A).
- What Priority 1 vs Priority 4 means and why.

At the end of Learning Guide, select Start Activity Builder to move into the Builder.

2.2 Build

Purpose: Builder mode is where you create your affiliate's policy decisions on each repair activity, using consistent labels and notes.

What you do in Builder mode

For each Activity in the catalog:

1. Find the activity
    a. Use the search bar to find activities by name.
    b. Use filters to narrow by Pillar, Sub-Category, Type, or Status.
2. Select a status (your policy decision)
    a. Eligible: within standard program scope.
    b. Not Eligible/Ineligible: not part of program scope.
    c. If/When: allowed only under defined conditions.
    d. N/A: not applicable in your context.
3. Add notes
    a. Notes are recommended for all decisions and effectively required for If/When decisions. Conditional without conditions is useless later.
4. Override urgency/condition only if needed
    a. Activities have default urgency/condition values. Override only when you have a deliberate policy reason and can explain it.

Recommended build method (so your output is coherent)

- Work pillar-by-pillar to avoid scattering decisions and leaving gaps.
- Use the Status filter to find unselected items and close them out before exporting.

What "done" looks like

You are done building when:

- Every activity you care about has a status (and anything left unselected is intentionally left unselected).
- All If/When activities include clear, usable notes.
- Overrides are defensible.

2.3 Export

Purpose: Export mode turns your Builder decisions into something you can publish, share, analyze, and apply as your program policy.

What you do in Export mode

1. Go to Export Activities.
2. Review your selections grouped by status and category.
3. Export using the format that matches your next use:

Export option A: Copy Text
Use when you want to paste the content directly into a policy manual or operating manual.

Export option B: Export CSV
Use when you want to QA decisions, analyze categories, or maintain a spreadsheet record.

Export option C: Print / Save PDF
Use when you need an appendix-ready artifact that can be filed or shared.

What "done" looks like
You are done exporting when you have at least one durable output saved somewhere that is not your browser cache:

- A PDF saved to your files, or
- A CSV saved to your files, or
- Text pasted into a formal document.

Export is the "make it real" step. The app's local save is helpful, but it is not a records system.

In-depth Guide

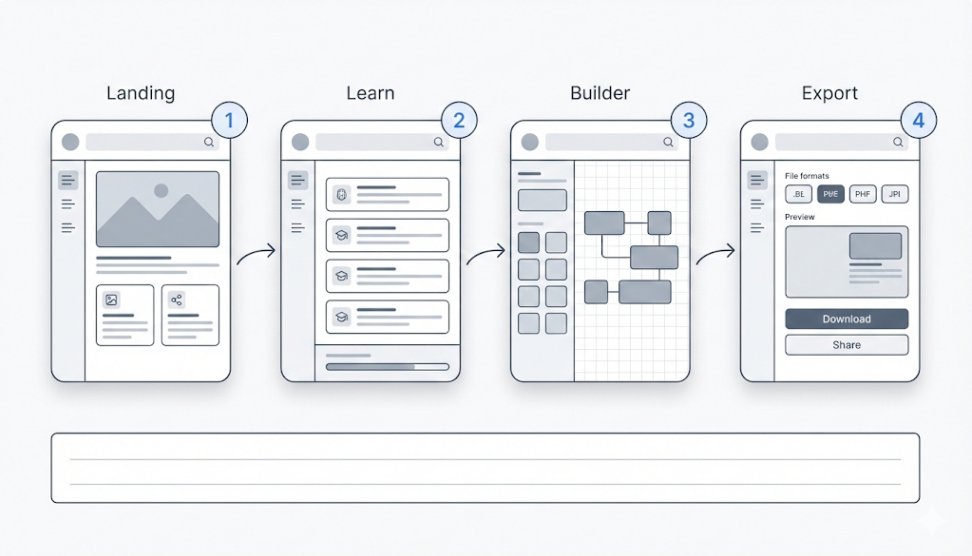
3. App Overview

The app is a guided workflow for turning a shared repairs taxonomy into an affiliate-specific scope and eligibility policy. It's not an inspection tool and it's not a project management system. It's a decision and documentation tool: you review activities, decide what your program will support, and export the result into formats you can actually use.

The app is intentionally structured as four screens that match how people work:

- Learn the system
- Make decisions
- Export a clean policy artifact

3.1 Screen Map (Landing, Learn, Builder, Export)



Landing

- The starting screen.
- Purpose: orient users and send them to the right starting point (typically Learn first, Builder second).

Learn

- A guided learning pathway that explains the taxonomy and the prioritization model.
- Purpose: create shared understanding before decisions get made.
- Output: user readiness, not a file.

Builder (Catalog)

- The working screen where you apply filters/search, review activity definitions, and select statuses.
- Purpose: create your affiliate's policy decisions at the activity level.
- Output: a saved draft decision set stored locally in the browser.

Export (Report)

- The output screen where you review selections grouped by category and status and export them.
- Purpose: generate artifacts (Copy Text, CSV, PDF) that can be stored and shared.
- Output: durable policy deliverables.

3.2 Navigation and Primary Actions

The app is designed around a few repeatable actions. If users understand these, using the app can become easily mastered.

Primary navigation actions

- Move between screens: Learn → Builder → Export is the intended sequence.
- Return to Builder: Users often export, notice a gap, return to Builder, fix it, and export again.

Primary actions inside Learn

- Step forward/back: Users move through learning modules in sequence.
- Start Activity Builder: The handoff from learning to decision-making.

Primary actions inside Builder

- Search: Find activities quickly by keyword.
- Filter: Narrow what you see by Pillar, Sub-Category, Type, and Status.
- Toggle definitions: Show or hide definitions when reviewing activities.
- Select status: Set Eligible, Ineligible, If/When, or N/A.
- Add notes: Document rationale or conditions.
- Override urgency/condition: Adjust prioritization only when you have a defined policy reason.

Primary actions inside Export

- Review grouped outputs: Check what you marked Eligible, Conditional, and Non-Eligible.
- Copy Text: Move policy content into a manual.
- Export CSV: Create a structured dataset for QA and analysis.
- Print/Save PDF: Generate an appendix-ready artifact.

3.3 What Gets Saved Automatically

The app saves your Builder selections automatically in your browser. This is convenient, but it also means you need to treat exports as your real record.

What is saved

- Activity status selections (Eligible, Ineligible, If/When, N/A)
- Any notes entered for activities
- Any urgency/condition overrides

What this implies

- If you refresh the page, your work should still be there.
- If you clear site data, switch browsers, or move devices, your saved work may not follow you.
- Exporting (CSV/PDF) is the recommended way to keep a durable record and share decisions.

Recommended practice

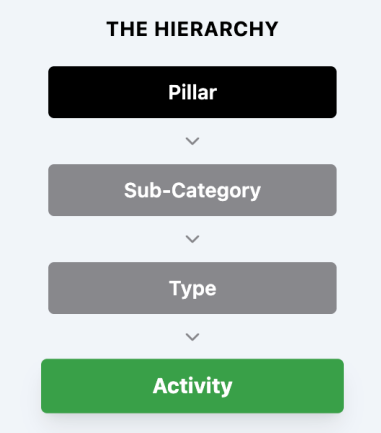
- Export at the end of each work session, even if your policy is still in draft.
- Treat the Builder as your “working copy” and the Export as your “record copy.”

4. Core Concepts the App Uses

4.1 Taxonomy: Pillar → Sub-Category → Type → Activity

This taxonomy is the backbone of the app and the Housing Preservation Framework. It controls how activities are organized, filtered, and exported.

- Pillar: the highest-level grouping (Four Pillars).
- Sub-Category: a major functional area within a pillar.
- Type: a tighter grouping used to cluster similar activities.
- Activity: the actual item you make a decision about.



Why this matters

- Users should make decisions at the Activity level.
- Filtering works best when you think in the same hierarchy the app uses.
- Exports group decisions by this structure, so staying consistent makes your outputs readable and defensible.

4.2 Status Decisions (Eligible, Ineligible, If/When, N/A)

Status is the core policy decision. Everything else is supporting detail.

Eligible

- The activity is within your program scope under normal program operations.
- Use when the activity is consistently allowed and fundable under your standard model.

Ineligible / Not Eligible

- The activity is out of scope for your program.
- Use when you do not fund, deliver, or manage this activity as part of your repairs program.

If/When (Conditional)

- The activity is allowed only under defined conditions.
- Use when eligibility depends on factors like:
  - Funding source restrictions
  - Caps (cost, household, per-project)
  - Special program types (disaster recovery, accessibility-only, energy-only)
  - Partner-provided services
  - Eligibility requirements (household type, geography, structure type)

N/A

- The activity does not apply to your context.
- Use when the activity is structurally irrelevant, legally impossible, or outside the housing type/geography you serve.

Common mistake to avoid

- Using If/When as “maybe.” If it’s “maybe,” then write the conditions that define when it becomes “yes.” Do not establish ambiguity within your decision-making process that can become a perceived bias.

4.3 Criticality Model: Urgency and Condition

The app uses a simple two-factor model to describe criticality and generate a priority label.

Urgency (how severe the consequence is)

- **Critical**: immediate risk to safety, habitability, or major system failure.
- **Emergent**: significant but not immediately catastrophic; likely to become critical.
- **Non-Critical**: important but not expected to threaten habitability in the near term.

Condition (how active the issue is)

- **Active**: currently failing, leaking, unsafe, or visibly deteriorating.
- **Passive**: degraded or aging, but not currently failing; risk is present.
- **Inactive**: stable condition, no immediate signs of failure, preventive in nature.

Why this matters

- The urgency/condition pairing drives the priority label.
- Overrides change how activities appear in priority groupings in the exports.

4.4 Priority Labels (How the Matrix Works)

Priority labels translate urgency and condition into a consistent "what comes first" ordering.

The app's matrix logic can be explained like this:

- Higher urgency + more active condition = higher priority
- Lower urgency + inactive condition = lower priority

A typical interpretation:

- Priority 1: Critical + Active
- Priority 2: Critical + Passive or Emergent + Active
- Priority 3: Critical + Inactive or Emergent + Passive
- Priority 4: Emergent + Inactive or Non-Critical + Active
- Priority 5: Non-Critical + Passive
- Priority 6: Non-Critical + Inactive

How to use priorities

- Priorities help structure scopes (Scope A vs Scope B) and manage pipeline triage.
- They should not replace professional judgment on a specific home. They exist to create consistency in policy and expectations.

4.5 Defaults vs Overrides (When to Touch Them)

Most activities come with default urgency and condition values. These defaults represent a standard interpretation of risk and typical program logic.

Default values

- The baseline prioritization intended for broad use across affiliates.
- Useful for consistency and comparison.

Overrides

- A local adjustment you make when your affiliate's policy, housing stock, climate, funding rules, or delivery model defines risk differently.

When to override

Override only when you can answer both:

1. What is our affiliate's policy reason for treating this differently?
2. Would another staff member make the same override if they followed our policy?

When not to override

- "It feels lower priority" without a policy rule.
- To match one unusual project.
- To make the export look cleaner.

Best practice

- Keep overrides rare.
- If you override, document the rationale in the notes.

4.6 Notes (What to Write and When They Are Required)

Notes are what turn a status selection into a usable policy statement.

When notes are required
Practically, notes are required when you select If/When because the entire point is to document the conditions. Conditional with no conditions is dead weight.

What to include in notes
Use notes to capture:

- The condition(s) that make the activity allowable
- Funding restrictions (specific sources or program types)
- Caps (maximum dollar amount, per-home limits, frequency limits)
- Household eligibility requirements
- Delivery requirements (licensed trade only, partner-only, staff-only)
- Exclusions or boundaries (what you will not do within that activity)

Recommended note style
Write notes so someone outside your team can apply them.

- Use short, specific statements.
- Prefer “Allowed when…” over general commentary.
- If it’s a rule, write it like a rule.

Examples (patterns, not prescriptions)

- “Allowed only when required for accessibility and when household includes an older adult or person with disability.”
- “Allowed only with dedicated funding source; not included in core repairs budget.”
- “Allowed up to $X per household; any amount above requires executive approval.”

In-App Learning Guide

5. Learning Guide Overview

Learning Guide is the app's built-in onboarding and calibration lane. It exists so your team shares the same definitions before anyone starts clicking "Eligible" like it's a personality test.

Learning Guide does not create an exportable output. It creates consistency, which is the only thing standing between a usable policy and a chaotic pile of opinions.

5.1 What Learning Guide Is For

Learning Guide is designed to help users do four things:

1. **Understand the catalog structure**
   Learn the hierarchy you will filter and decide within: Pillar → Sub-Category → Type → Activity.
2. **Understand the decision you are making in Builder**
   You are not choosing "repairs we like." You are defining a policy stance for each activity: Eligible, Ineligible, If/When, or N/A.
3. **Understand prioritization language**
   Learn how Urgency and Condition combine into a Priority label so your team stops using "urgent" to mean "I saw it once and it scared me."
4. **Reduce overrides and clean up notes later**
   The better the Learning Guide baseline, the less time you spend arguing in Builder and rewriting "If/When" notes after exporting.

Use Learning Guide when:

- A new staff member joins the repairs team.
- You are building a policy for the first time.
- You are updating an existing policy and need to re-align on definitions.
- Multiple people will contribute to Builder decisions and you want consistency.

5.2 Module List and Learning Path

Learning Guide is organized as a sequence of modules. The intended path is linear, because it builds understanding in the same order the catalog is structured.

Recommended learning path (in order)

Module 1: Four Pillars Overview
Purpose: Establish the top-level purpose of each pillar so users understand what "belongs" where.

What users should take away:

- What each pillar includes and does not include.
- Why the catalog separates safety, health, performance, and community repair.

Module 2: Taxonomy and Hierarchy
Purpose: Teach users how the catalog is organized so filters and exports make sense.
What users should take away:

- The difference between Sub-Category vs Type vs Activity.
- Why decisions are made at the Activity level.
- How the hierarchy supports consistent grouping in exports.

Module 3: Deep Dive (Definitions and Scope Boundaries)
Purpose: Reduce ambiguity by clarifying definitions users will see in Builder.
What users should take away:

- How to interpret definitions when categorizing an activity.
- How to keep decisions consistent when multiple activities look similar.
- When "N/A" is appropriate versus "Ineligible."

Module 4: Criticality Matrix (Priority Model)
Purpose: Teach the two-factor prioritization model that drives priority labels.
What users should take away:

- How Urgency and Condition work and why both matter.
- What Priority 1–6 generally represent.
- Why overrides should be rare and policy-driven.

Minimum completion standard
If you are trying to move fast, the minimum recommended Learning Guide completion is:

- Taxonomy and Hierarchy
- Criticality Matrix

5.3 Progress and Navigation

Learning Guide is step-based. Users move through content using forward/back navigation and progress indicators.

How to navigate

- Use Next and Back to move through steps in sequence.
- Revisit prior steps when definitions or priority logic are unclear.
- Treat Learning Guide as reference, not a one-time ritual. People forget things. Constantly.

What "progress" means here

Progress is not a credential. It's a reminder of where you are in the learning path.

A user is ready to move into Builder when they can confidently answer:

- What level do I make decisions at? (Activity)
- What does If/When mean and what should accompany it? (Conditions, written in notes)
- What do Urgency and Condition represent?
- Why do priorities exist in the app? (Consistency and defensible triage)

Common learning mistakes (and how to avoid them)

- **Skipping definitions and then overusing overrides later -** If someone starts overriding a bunch of urgency/condition values, it usually means they skipped the matrix logic or misunderstood condition states.
- **Treating If/When as "maybe" -** If someone cannot explain the "when," it is not ready for Builder decisions yet.
- **Confusing Ineligible vs N/A -** Ineligible means "we choose not to do it.", N/A means "this does not apply in our context."

5.4 Transitioning to the Builder

Once the Learning Guide has done its job, the app hands users into the Builder using Start Activity Builder.

When to transition
Transition when:

- You understand the hierarchy and can use it to locate activities.
- You understand the four status options and what they imply in policy.
- You can explain urgency vs condition and what priority means.

What to do immediately after entering Builder (recommended)
This sets users up for clean decisions and an easier QA pass later.

1. **Pick one pillar to start with**
   Work pillar-by-pillar. It reduces gaps and makes exports cleaner.
2. **Turn on definitions (at least at the start)**
   Read definitions for the first set of activities until you are calibrated.
3. **Use filters before search**
   Filters give you structure. Search is great when you already know what you want.
4. **Decide status first, notes second, overrides last**
   The decision is the status. Notes explain it. Overrides should be rare and justified.
5. **Plan your "unselected" strategy**
   Decide whether your policy must cover every activity (complete coverage) or only activities you commonly see (targeted coverage). Either is valid, but the QA process is different.

Section C: Builder Mode (Catalog)

6. Builder Overview

Builder Mode is where the app earns its keep. This is the working screen where you:

- **Find activities** (search + filters)
- **Make decisions** (status)
- **Document decisions** (notes)
- **Optionally adjust priority logic** (urgency/condition overrides)
- **Prepare clean outputs** (by reducing "unselected" gaps)

The Builder is your drafting workspace. The Export screen is your record output. Treat it that way and life gets easier.

6.1 Builder Layout and Controls

Builder is organized into three functional zones:

A) Filter panel (left sidebar)
Used to narrow the catalog to a manageable working set:

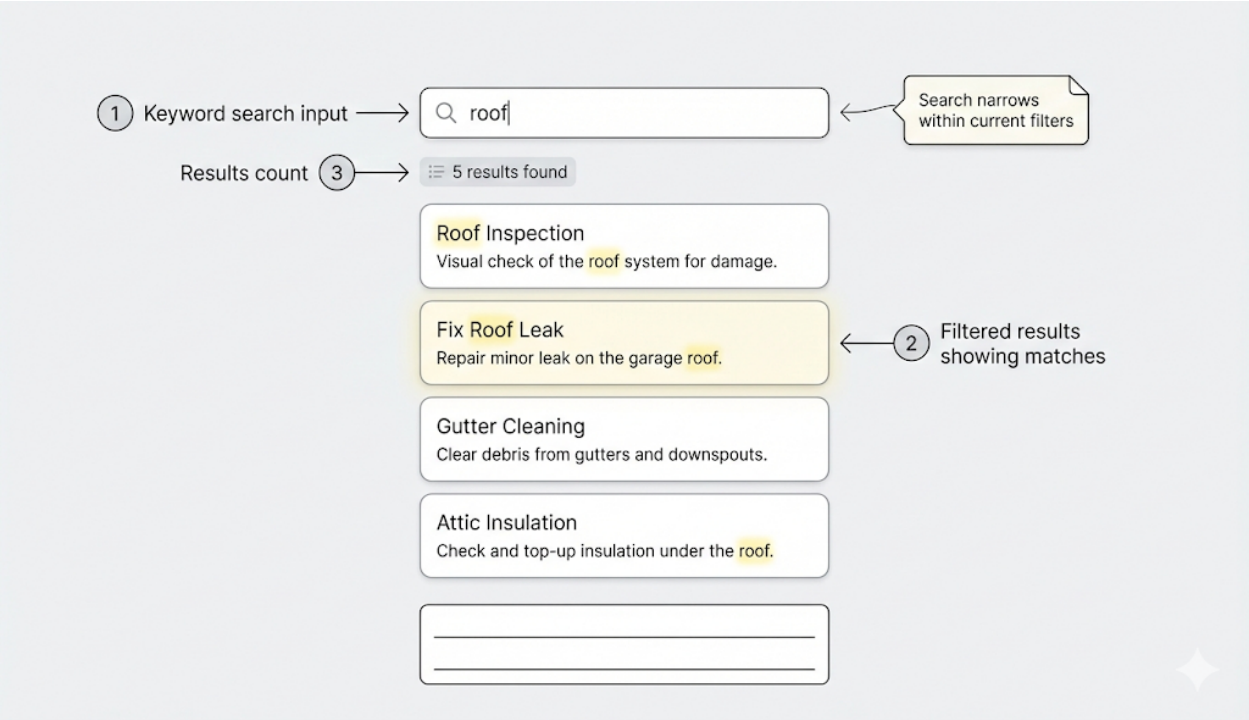
- Pillar
- Sub-Category
- Type

- Status (including “Unselected”)

B) Main catalog list (center)
Where activity cards appear after filters/search are applied.
C) Activity decision controls (within each activity card)
Where you:

- Select a status
- Add notes
- Override urgency/condition (if permitted in the UI)

Breadcrumb/location indicator (context cue)
When you filter deep (Pillar → Sub-Category → Type), the app shows you “where you are” so you do not forget what you filtered into and accidentally make decisions out of context.



6.2 Search

Search is the fastest way to locate specific activities when you already know what you're looking for.

What search does well

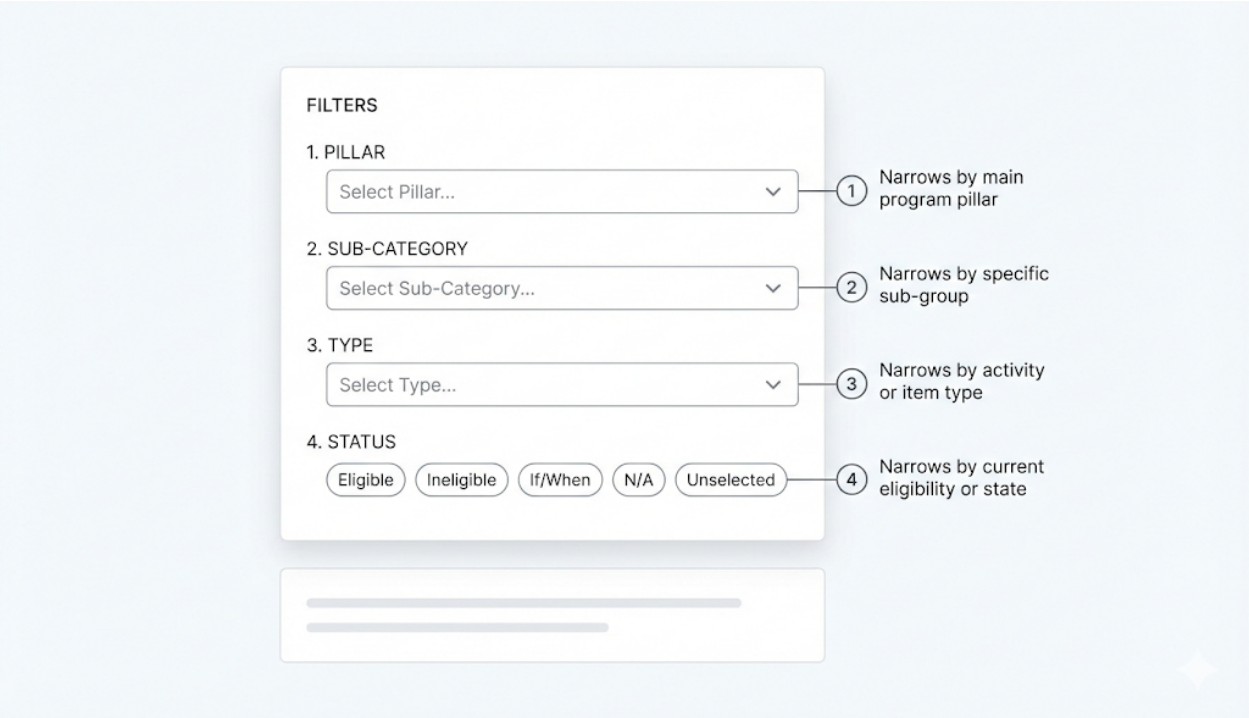
- Finds activities by keywords in the activity name
- Surfaces related items when naming is consistent (examples: "roof," "gutter," "mold," "ramp," "panel," "water heater")

How to use search effectively

- Use short keywords first ("roof" instead of "roof replacement and flashing")
- Try the system name, not the homeowner description ("water heater" instead of "no hot water")
- If you cannot find something, use a broader category term ("electrical" instead of "GFCI")

Search and scope decisions

Search should help you locate an activity, not decide its policy stance. Humans love mixing those up.



6.3 Filter Panel (Pillar, Sub-Category, Type, Status)

Filters are what make Builder usable. They are also what prevents the "I made 40 decisions and have no idea where I was" problem.

Pillar filter

What it narrows: Everything displayed will fall under the selected pillar.

When to use it: Almost always. Starting with a pillar is the cleanest way to build and later export coherent outputs.

Sub-Category filter

What it narrows: Only items under the selected sub-category of your chosen pillar.

When to use it: When you want to complete a functional area in one pass (ex: Structural Components) without getting distracted by adjacent issues.

Type filter

What it narrows: A tighter grouping inside a sub-category, usually clustering similar activity families.

When to use it: When you are making detailed decisions and want consistency across closely related activities.

Status filter (including Unselected)

This filter is your best friend and worst enemy because it shows you what you did and what you avoided.

What it narrows: Items by decision state:

- Eligible
- Ineligible
- If/When

- N/A
- Unselected (no decision yet)

What "Unselected" actually means

- You have not made a policy decision for that activity.
- It is not "ineligible," not "N/A," not "pending," not "later," not "I looked at it and felt something."
- It is simply: no decision recorded.

How to use Unselected correctly

- Early build: Use it to track progress and avoid losing your place.
- Mid build: Use it to identify which areas are incomplete before moving to another pillar.
- Pre-export: Use it as a QA list to decide whether you are exporting a complete policy or a partial one.

Two valid strategies for Unselected

1. **Complete coverage strategy**
   You aim to decide every activity in the catalog. Unselected should go to zero (or as close as your organization requires).
2. **Targeted coverage strategy**
   You only decide what your program actually expects to encounter. Unselected items remain, but that is intentional and documented.

If you do targeted coverage, be explicit about it. Otherwise, someone will assume missing items are secretly allowed or secretly banned. Humans do love mystery when it is inconvenient.

How search interacts with filters (the practical version)

Think of it like this:

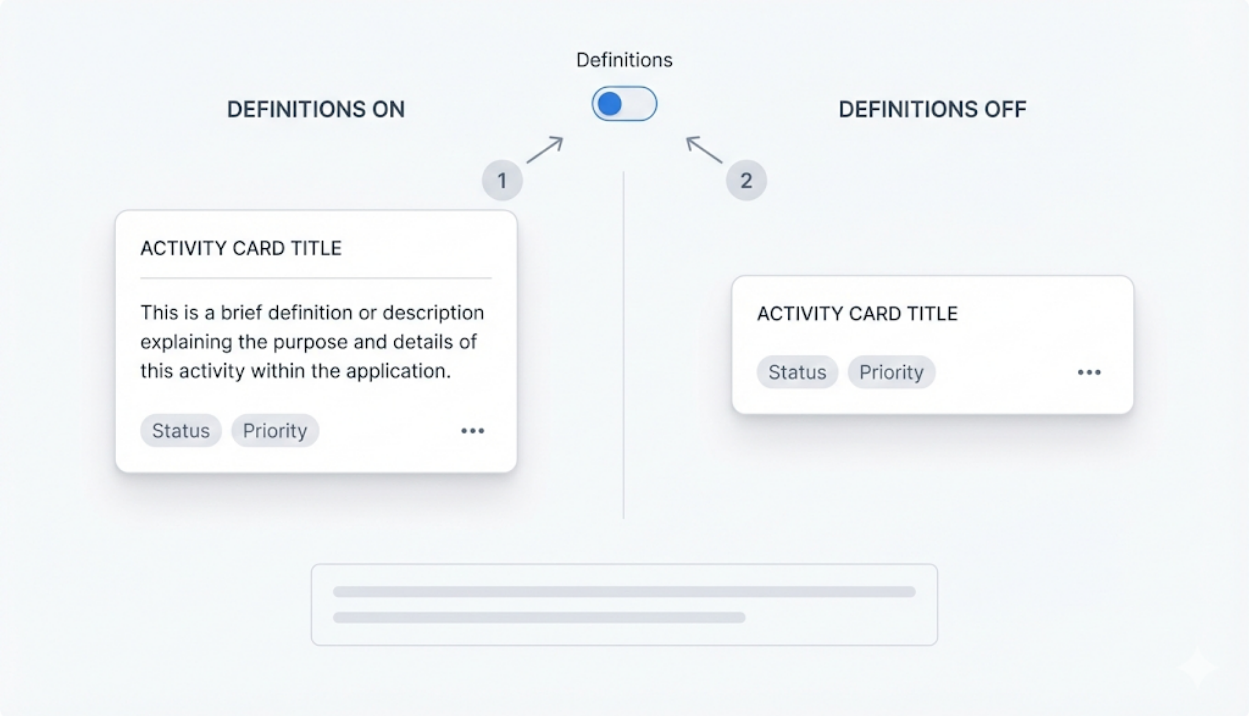
- **Filters define the neighborhood**
- **Search finds the house**

Best practice:

1. Filter to Pillar (and Sub-Category if you can)
2. Use search within that filtered set to locate specific activities
3. Make decisions consistently inside that scope

Common mistake:

- Searching without filters and making decisions across random pillars in one sitting. Your export will look like a junk drawer.



6.4 Definitions Toggle

Definitions are there to prevent category drift and “close enough” decision-making.

When to turn definitions ON

- At the start of a pillar
- When multiple users are building decisions
- When two activities sound similar and you need the boundary
- When you are tempted to override urgency/condition (often a definition issue, not a priority issue)

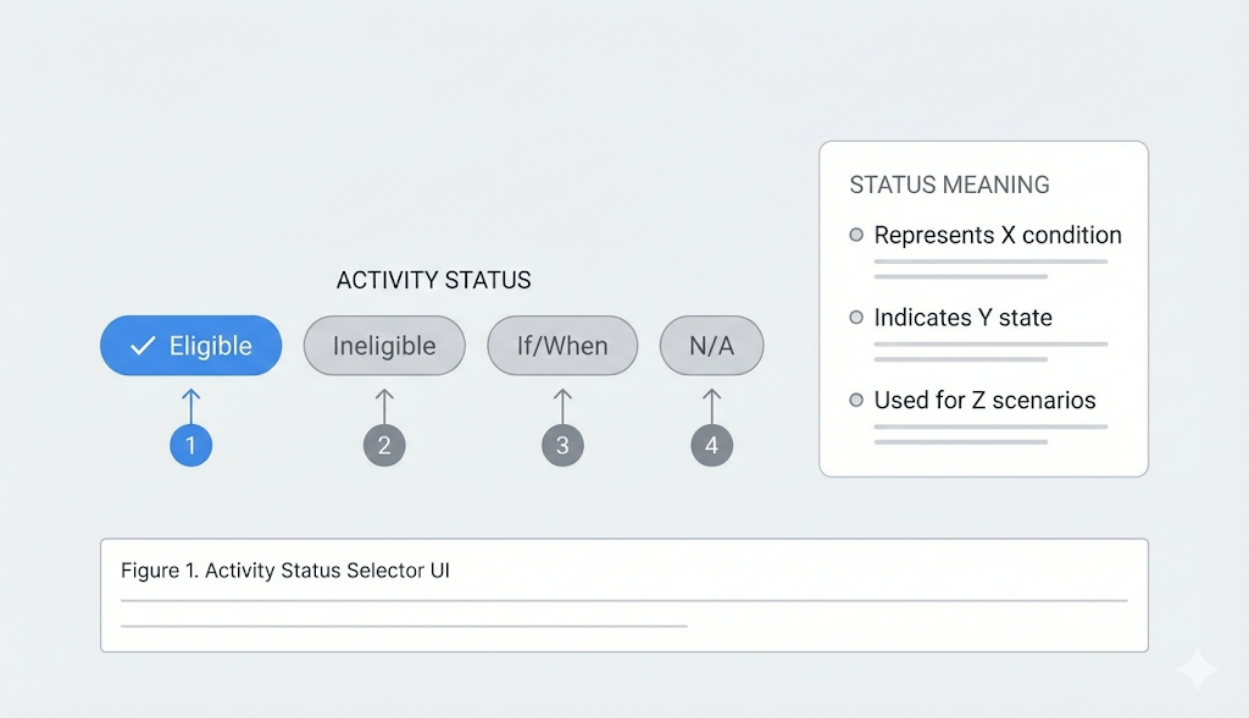
When definitions can be OFF

- After your team is calibrated and you are moving quickly
- When you are reviewing decisions you already made

Best practice

- Definitions ON during decision-making, OFF during speed-review.

7. Making Decisions on Activities



7.1 Selecting a Status

Status is the core decision. Everything else supports it.

Eligible

- This activity is within normal program scope.
- You can do it consistently under your standard delivery and funding model.

Ineligible

- The program will not fund or deliver this activity.
- Use this when you want a firm boundary.

If/When

- The activity is allowed only under defined conditions.
- If you select this, notes are not “nice to have.” They are the whole point.

N/A

- The activity does not apply to your context (housing type, geography, legal constraints, or program design).

Consistency rule

If two staff members would apply different statuses to the same activity, your “policy” is really just vibes. Fix that before exporting.

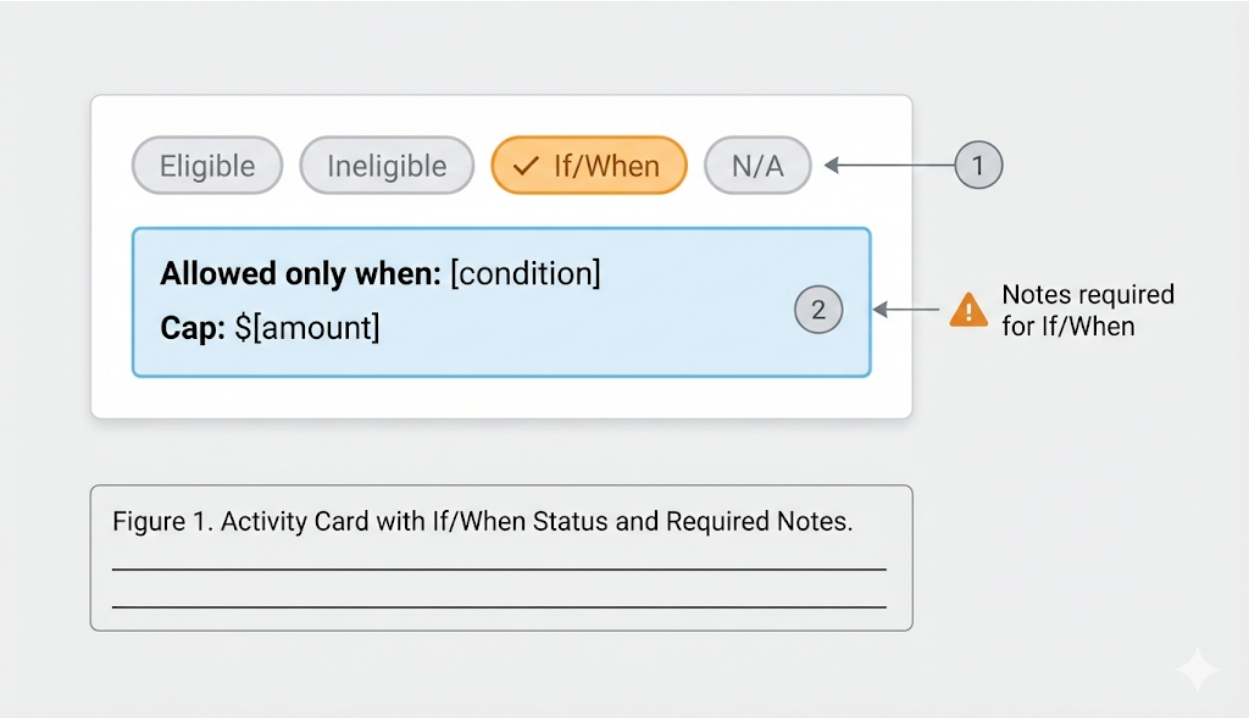


Figure 1. Activity Card with If/When Status and Required Notes.

7.2 Using If/When Properly

If/When is a conditional policy statement. Use it when eligibility depends on a rule, not a mood.

Good reasons for If/When

- Funding restrictions (allowed only with dedicated funds)
- Caps (allowed up to a maximum cost or only once per household)
- Program type (allowed only under accessibility initiative, disaster recovery, energy program)
- Delivery constraints (allowed only via licensed contractor, partner-led delivery)
- Eligibility requirements (allowed for specific household types or targeted populations)

Bad reasons for If/When

- “We might do it.”
- “If the homeowner is nice.”
- “If the construction manager has time.”

If it is truly discretionary, you still need a rule that defines discretion, otherwise you are exporting ambiguity.

7.3 Adding Notes

Notes are how you turn a selection into a usable policy artifact.

Notes should capture:

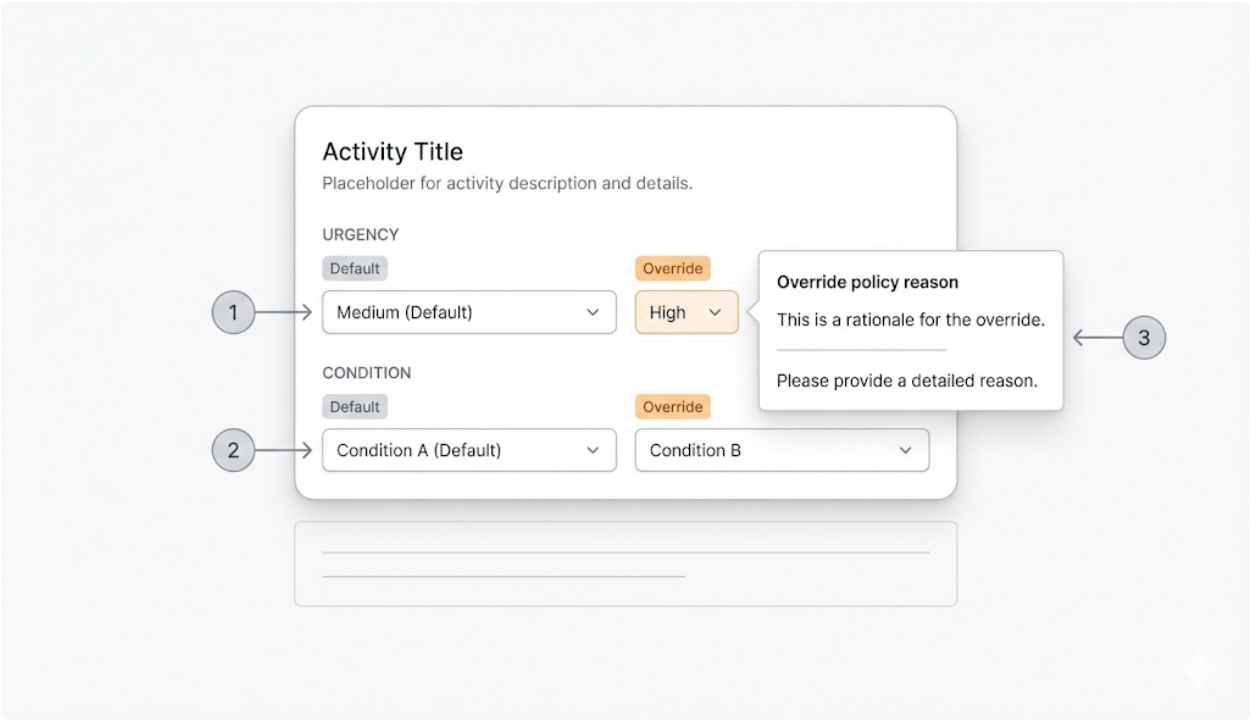
- The condition(s) that make the activity allowable
- Any caps, limits, or thresholds
- Delivery requirements (licensed trade, partner-only, staff-only)
- Funding constraints
- Exclusions and boundaries (what is not included)

Write notes like someone else will apply them

Because someone else will apply them. Probably under deadline. Probably incorrectly unless you help them.

Recommended note patterns

- “Allowed when…”
- “Allowed only if…”
- “Not included unless…”
- “Requires…”
- “Maximum of…”



7.4 Overriding Urgency and Condition

Overrides change how an activity shows up in priority groupings and can affect how users interpret Scope A vs Scope B.

Only override when:

- You have a defined affiliate policy reason
- You can explain the rule clearly in notes
- Another staff member would make the same override using the same policy

Do not override to:

- Make exports look cleaner
- Fit a one-off project
- Express personal judgment about one house

Best practice

Keep overrides rare. If overrides are frequent, the issue is usually: unclear definitions, inconsistent decision rules, or skipping calibration in Learn Mode.

7.5 Reviewing Unselected Items

Unselected is your QA heatmap.

Use Unselected to:

- Confirm you completed a pillar before moving on
- Identify gaps before exporting
- Decide whether you are exporting a complete policy or a draft

Pre-export decision

Before exporting, make a conscious choice:

- "This export represents a complete policy for our program scope."
  or
- "This export represents a draft or a targeted coverage set."

If you export without stating which one, people will assume it is complete. People assume things constantly. It's their hobby.

8. Recommended Build Workflow

8.1 Pillar-by-Pillar Build Sequence

This is the cleanest method, and it produces exports that read like a policy instead of a scavenger hunt.

Recommended sequence

1. Select Pillar
2. Work Sub-Category by Sub-Category
3. Use Type filtering when needed for consistency
4. Use Status filter (Unselected) to close gaps
5. Export and review
6. Repeat for the next pillar

Why this works

- Reduces missed items
- Keeps decision logic consistent
- Produces more coherent grouping in export outputs

8.2 Team Review and Consensus Method

Builder decisions go wrong when teams do not decide how they decide.

Recommended roles

- **Owner/Facilitator:** drives the session, keeps scope rules consistent
- **Construction voice:** checks feasibility and delivery constraints
- **Program/compliance voice:** checks funding rules and eligibility constraints
- **Recorder:** ensures notes are written in policy language, not conversational commentary

Recommended decision cadence

- Make status decisions first
- Identify items requiring If/When and write conditions immediately
- Reserve overrides for a dedicated second pass

How to handle disagreement

- If disagreement is about meaning, re-check definitions
- If disagreement is about policy, document the rule in notes
- If disagreement is about one weird house, do not encode it into the catalog

8.3 Quality Checks Before Export

Do this every time you export. It takes minutes and prevents embarrassing policy artifacts.

QA Check 1: Coverage clarity

- Are you exporting a complete policy or targeted coverage?
- Does the team agree?

QA Check 2: Unselected review

- Filter to Unselected in each pillar
- Confirm items are either decided or intentionally left undecided

QA Check 3: If/When completeness

- Filter to If/When
- Confirm every conditional item includes usable conditions in notes
- Remove “maybe” language and replace with rule language

QA Check 4: Override sanity

- Review overridden urgency/condition items as a set
- Confirm there is a policy reason and a note explaining the rationale
- If overrides are common, stop and recalibrate. You are building chaos.

QA Check 5: Export preview

- Go to Export screen and scan grouping
- Look for gaps or oddly placed items
- Return to Builder to fix anything confusing before generating the record copy

Section D: Export Mode (Report)

9. Export Overview

Export Mode is where your Builder decisions become something you can file, share, and defend later. The Export screen is designed for two things:

1. **Review**: see your decisions grouped in a way that reads like policy.
2. **Output**: generate artifacts that can live outside the app (text, CSV, PDF).

Builder is drafting. Export is publishing.

9.1 What the Report Shows

The Export screen presents your selections organized for human readability and policy use.

What you will see

- A consolidated view of your decisions, grouped by **status** (Eligible, Conditional/If-When, Non-Eligible).
- Within each status group, items are organized by the taxonomy structure (Pillar and Sub-Category), so the output reads like a structured appendix instead of a random list.
- Conditional items are presented with the expectation that **notes contain the conditions**.

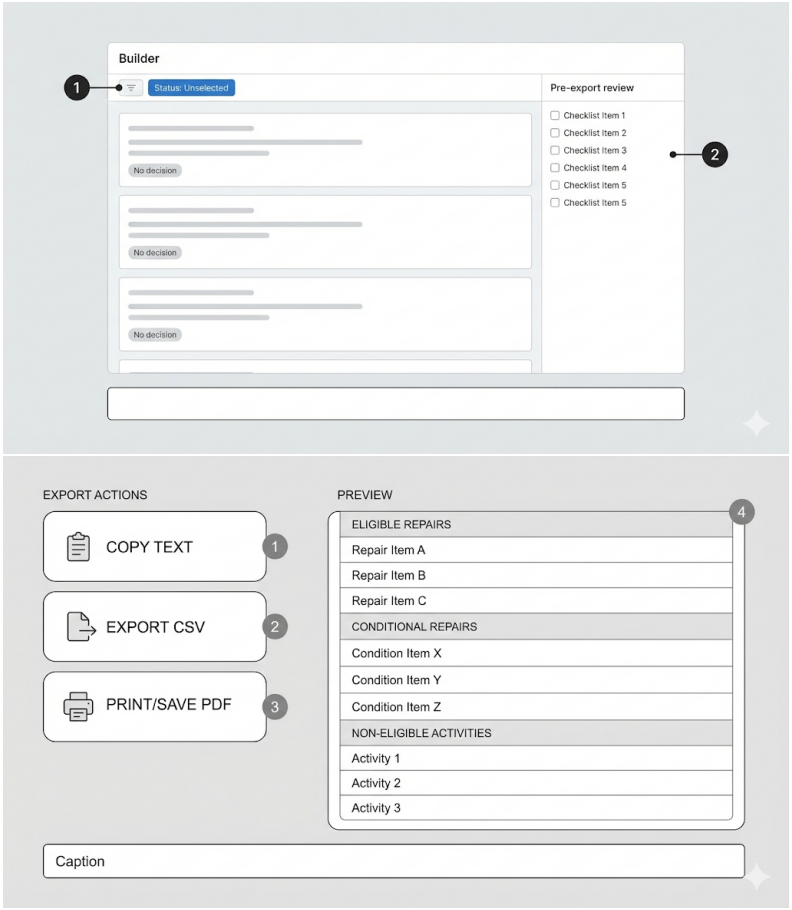
What the report is for

- Sanity-checking: “Do these decisions make sense when grouped?”
- Spotting gaps: “Why is this sub-category empty?”
- Catching unclear logic: “These conditionals have no usable conditions.”

What the report is not

- A substitute for Builder decisions. Export will not fix messy choices. It will faithfully display them, like a mirror you did not ask for.

10. Export Options



10.1 Copy Text Output (For Manuals)

Best for

- Dropping directly into a Repairs Program Policy Manual or Operating Manual.
- Creating an “Appendix: Eligible Activities” section without manually rewriting lists.

What it typically includes

- Headings by status category (Eligible, Conditional, Non-Eligible).
- Grouping by Pillar/Sub-Category.
- Activity names and associated notes where applicable (especially for conditional items).

When to use

- You are building or updating a policy document.
- You want the output to be readable without Excel.
- You need to share scope decisions with leadership or partners.

Common mistake

- Copying text before doing a conditional notes review. Conditional lists without conditions are just long ways of saying “we did not finish.”

CSV Field Dictionary

Field	Meaning
Pillar	Broad category for the item.
Sub-Category	Specific sub-grouping within pillar.
Type	Classification of the entry.
Activity Name	Descriptive name of the activity.
Selected Status	Current operational state.
Priority Label	Level of importance or urgency.
Final Urgency	Concluding urgency assessment.
Final Condition	Concluding condition assessment.
Notes	Optional comments or details.

Caption space below.

10.2 CSV Export (For Analysis)

Best for

- QA and completeness checks.
- Internal analysis (counts by pillar, how many conditionals, how many unselected).
- Sharing with a data team or importing into spreadsheets.

What CSV enables

- Filtering: “Show me all If/When activities and their notes.”
- Coverage auditing: “What is still unselected?”
- Consistency checking: “Do we have wildly different priorities inside the same type?”
- Version comparisons: “What changed between last quarter and now?”

How to use it well

- Treat CSV as your audit file, not just an export.
- Store it with a date and version label in the filename.
- Use it as the foundation for change tracking and governance review.

Common mistake

- Exporting CSV and assuming it is self-explanatory to someone else. If your notes are vague, CSV will preserve that vagueness at scale.

10.3 Print / Save PDF (For Appendices)

Best for

- An appendix-ready policy artifact that can be filed, emailed, or attached to documentation.

- Board packets, funder documentation, and “official” record copies.

What PDF is good at

- Readability and consistency
- Preserving a snapshot in time
- Making it harder for someone to casually edit scope decisions without noticing

When to use

- Any time you want a formal output that should not be casually reworded.
- Any time you finish a major scope update and want a record copy.

Common mistake

- Printing before verifying conditional notes and overrides. PDF makes messy decisions look official, which is exactly how organizations accidentally institutionalize confusion.

11. How to Use Exports Downstream

11.1 Policy Manual Integration

Use **Copy Text** or **PDF** to populate a policy section such as:

- "Eligible Repairs"
- "Conditional Repairs (If/When)"
- "Non-Eligible Activities"

Recommended placement

- As an appendix in your Repairs Program Policy Manual (easiest to update without rewriting the whole manual).
- As part of your scoping and eligibility section if your manual structure is simpler.

Best practice

- Include a short paragraph before the appendix that explains:
  - The version/date
  - How conditional items are interpreted
  - Who approves changes
  - How often the appendix is reviewed

11.2 Spreadsheet Review and QA

Use **CSV** for internal QA before you call something "final."
Recommended checks:

- Count activities by status (do you have suspiciously high If/When?)
- Filter to If/When and review notes quality
- Filter to Unselected and confirm your coverage strategy
- Sort by priority label and scan for outliers (activities that look mis-prioritized)

If you are doing governance seriously:

- Maintain a "master" spreadsheet file where each export is stored as a new tab labeled by date/version.
- Use differences between tabs to generate change notes.

11.3 Sharing and Version Control

Because the app saves locally in the browser, exports are the shareable truth.

Recommended version control practice

- Save exports with a naming convention:
  - AffiliateName_RepairsCatalogPolicy_vX.Y.Z_YYYY-MM-DD.pdf
  - AffiliateName_RepairsCatalogExport_vX.Y.Z_YYYY-MM-DD.csv

Store exports in a shared location

- Team drive folder, SharePoint, or whatever your organization uses to hoard documents.
- Keep PDF as the "record copy" and CSV as the "audit copy."

Governance reminder
If you cannot answer "who approved this scope and when," you do not have policy. You have a document.

11.4 Export QA Checklist (Quick, but mandatory if you want fewer headaches)

Before you export anything you will share:

1. Filter to **If/When** and confirm every item has usable conditions in notes.
2. Review **overrides** as a set and confirm they are policy-driven.
3. Decide your **coverage strategy** and document it (complete vs targeted).
4. Scan the Export grouping to make sure nothing looks misplaced or empty.
5. Export **PDF for record** and **CSV for audit** when finalizing a version.

Section E: Administration and Support

This is the part where we talk about the unsexy reality: how your work is stored, how it gets lost, how to keep records, and how to troubleshoot the predictable ways humans and browsers sabotage progress.

12. Data Storage, Recovery, and Portability

12.1 What Local Save Means (and Its Limits)

The app automatically saves your work **locally in your browser** while you use Builder Mode. This is convenient, fast, and also not a records system.

Local save is good for:

- Drafting decisions over multiple sessions
- Leaving and coming back later
- Iterating without exporting every five minutes

Local save is not good for:

- Cross-device continuity (laptop → desktop)
- Multi-user collaboration (two people, one set of decisions)
- Permanent storage (cache clearing happens, often accidentally)
- Audit trails or governance documentation

Bottom line:

If you want your work to exist outside your browser, you must export it.

12.2 Backup Best Practices

If you want fewer "we lost it" moments, treat exports as your backups.

Minimum recommended backup routine

- **During drafting:** Export CSV at the end of each working session.
- **When finalizing:** Export both **CSV + PDF** and store them somewhere shared.

Recommended file naming convention

Use a naming scheme that makes it obvious what version is "current" and what is historical.

- AffiliateName_RepairsCatalogExport_vX.Y.Z_YYYY-MM-DD.csv
- AffiliateName_RepairsCatalogPolicy_vX.Y.Z_YYYY-MM-DD.pdf

If you do not include date and version in the filename, you are inviting your future self to suffer.

Recommended storage location

- A shared drive folder with controlled access (SharePoint/Drive/Team folder)
- A "Policy Exports" subfolder, with a simple index file if you want to be grown-up about it

12.3 Troubleshooting Data Loss

If your selections disappeared, it's almost always one of these:

Common causes

- Browser cache or site data was cleared
- You switched browsers (Chrome vs Safari, etc.)
- You switched devices
- You are in private/incognito mode (which often does not persist local storage)
- A security policy or extension cleared storage

What to do

1. Check if you exported a CSV/PDF previously (your real backup).
2. If no exports exist, you may need to redo the decisions.

Prevention

- Export regularly.
- Do not rely on local save as a final storage method.
- Use one primary browser for this tool.

13. Troubleshooting and FAQs

13.1 I cannot find an activity

Likely causes

- Filters are too narrow (you filtered it out)
- You are searching using a homeowner description instead of a system/activity name

Fix

- Clear filters one at a time until the list expands again
- Try broader search terms ("electrical" instead of "GFCI")

13.2 My export looks wrong or incomplete

Likely causes

- You have lots of "Unselected" activities
- You built across random pillars without completing sections
- Your conditional items have no notes, so the output reads like a shrug

Fix

- In Builder, filter to **Unselected** and decide whether to complete or intentionally leave gaps
- Filter to **If/When** and verify notes exist and are meaningful
- Export again after cleanup

13.3 My Conditional list is confusing

Likely cause

- If/When was used as "maybe" instead of "allowed under conditions"
- Notes are missing, vague, or inconsistent

Fix

- Rewrite conditional notes into rule language:
  - "Allowed only when…"
  - "Allowed only if…"
  - "Requires…"
  - "Up to $X…"

Rule of thumb

If another staff member cannot apply your condition without calling you, the note is not done.

13.4 The PDF print layout looks odd

Likely causes

- Browser print settings (margins, scaling, headers/footers)
- Printing to paper instead of "Save as PDF" with standard settings

Fix

- Use "Save as PDF"

- Set margins to default
- Disable browser headers/footers if they clutter the page
- Avoid printing directly to paper until the preview looks clean

---

13.5 Two people made different decisions for the same activity

Cause

- No shared policy rule existed
- Learn Mode was skipped or treated as optional
- Notes were not used to document the rule

Fix

- Decide the policy rule as a team
- Document the rule in the notes
- Revisit similar activities and align decisions for consistency

14. Glossary

Keep this short in the main guide and expand in an appendix if needed.

- **Pillar** - Top-level category used to organize the catalog (Four Pillars).
- **Sub-Category** - Major functional area within a pillar (example: Structural Components).
- **Type** - A grouping inside a Sub-Category that clusters similar activities.
- **Activity** - The specific item you make a decision about in Builder Mode.
- **Eligible** - In standard program scope.
- **Ineligible / Not Eligible** - Out of program scope.
- **If/When (Conditional)** - Allowed only under defined conditions that must be described in notes.
- **N/A** - Not applicable in your context.
- **Urgency** - How severe the consequence is if the issue is not addressed (Critical, Emergent, Non-Critical).
- **Condition** - How active or present the issue is (Active, Passive, Inactive).
- **Priority Label** - A consistent triage label derived from urgency + condition.
- **Override** - A deliberate adjustment to default urgency/condition values based on affiliate policy.

15. Appendix

15.1 Export Field Definitions (CSV Column Dictionary)

Use this as a reference when reviewing or analyzing CSV exports.

- **Pillar:** Top-level category
- **Sub-Category:** Functional grouping within pillar
- **Type:** Sub-grouping within sub-category
- **Activity Name:** The activity being decided
- **Selected Status:** Eligible / Ineligible / If/When / N/A / Unselected
- **Priority Label:** Priority classification derived from urgency + condition
- **Final Urgency:** Default or overridden urgency
- **Final Condition:** Default or overridden condition
- **Notes:** User-entered policy rationale or conditions

15.2 Priority Matrix Reference

This guide assumes a simple matrix logic:

- Higher urgency + more active condition = higher priority
- Lower urgency + inactive condition = lower priority

Use priorities to structure policy and triage, not to pretend you inspected an actual house through a dropdown.

15.3 Example Notes Library (If/When Templates)

Funding-restricted

- “Allowed only with dedicated funding source; not included in core repairs budget.”

Cap-limited

- “Allowed up to $____ per household; amounts above require approval.”

Program-type limited

- “Allowed only under accessibility program scope.”

Eligibility limited

- “Allowed only when household includes ____ and need is documented.”

Delivery constraint

- “Requires licensed contractor; volunteer labor not permitted.”

Boundary/exclusion

- “Includes ____; excludes ____.”