The ArcGIS Compendium — Volume H: Practice and Reference

ON THIS PAGE

The ArcGIS Compendium — Volume H: Practice and Reference

Putting it all together — two complete worked projects, forty how-do-I recipes, a troubleshooting encyclopedia, and a 170-term glossary.

One of eight volumes. Approximately 22601 words. Chapters cross-reference the whole set by number.

In this volume:

Worked Project: From Raw Data to Published Dashboard

The city of Milbrook has roughly nine thousand street trees and exactly one record of them: a spreadsheet. A contractor crew spent last summer walking every block with a tablet, and what they handed the Parks Department is a CSV file — a plain text table where commas separate the columns — with about 8,700 rows in it. The parks director wants something she can open on a Monday morning: a live map of the trees, a count of how many are in poor condition, a chart of the most common species, and a list of which trees have gone longest without inspection. She wants it to update itself as crews keep working. She wants it shared with her team and nobody else, at least for now.

That request — CSV in, dashboard out — is the single most common end-to-end job in web GIS, and it exercises almost everything this Compendium covers. The tool-by-tool chapters tell you how each piece works; this chapter shows you the connective tissue: the order to do things in, the decisions that only matter three steps later, and the places where a shortcut now becomes a rebuild next month. We will walk the whole project in sequence, and at every step point you to the chapter that covers that step in depth.

Working backward from the finished product

Before touching the data, sketch the pipeline in reverse, because every layer of this stack constrains the one below it:

The **dashboard** (Chapter 28) displays elements — indicators, charts, lists — that are fed by a **web map** (Chapter 6). The web map styles and configures a **hosted feature layer view** (Chapter 10), which is a filtered, permission-controlled window onto a **master hosted feature layer**. The master layer's shape is fixed by its **schema** — the list of fields, their types, and their allowed values (Chapter 12). And the schema is only as good as the **cleaned data** you publish (Chapters 5 and 11).

Read that chain forward and you get the project plan: clean the CSV, design the schema, publish the master layer, create views, build the map, build the dashboard, share it. Seven steps. The order matters because mistakes flow downhill: a sloppy field type chosen at publish time surfaces as a chart that refuses to sum; a layer shared to the

wrong audience surfaces as a dashboard that shows teammates a blank map. Fixing problems at the bottom of the stack after the top is built is possible — we will do it deliberately near the end of this chapter — but it is never free.

Step 1: Inspect the raw CSV

Open the file in a spreadsheet program and just look at it for ten minutes. This is the data evaluation habit from Chapter 5 applied to your own data, and it will save you hours. Milbrook's file has these columns:

```
Tree ID, SPECIES, dbh, HeightFt, Cond., Date Planted, LastInsp, Y, X, Notes, Crew
```

Scrolling through, the problems announce themselves, and they are the same problems nearly every real-world CSV has:

- **Free-typed categories.** The species column contains `Red Maple`, `red maple`, `Acer rubrum`, and `maple - red`, all meaning the same tree. The condition column has `Good`, `good`, `fair-ish`, and one memorable `DEAD?`.
- **Mixed date formats.** Some dates read `06/14/2025`, some `14-Jun-25`, some `2025-06-14`. Software cannot reliably guess which is which — is `06/07` June 7th or July 6th?
- **Coordinate trouble.** The `Y` column should hold latitude (Milbrook is in New England, so roughly positive forty-something) and `X` longitude (roughly negative seventy-something). A few dozen rows have those values swapped, and a handful are blank entirely.
- **Duplicates and near-duplicates.** A few `Tree ID` values appear twice — the crew re-surveyed some blocks.
- **Cryptic abbreviations.** `dbh` is *diameter at breast height*, the standard forestry measurement of trunk thickness, recorded here in inches. You know that because you asked the contractor. Always ask; never guess what a column means.

> **Tip:** Before you change a single cell, save an untouched copy of the original file somewhere safe and do all your cleaning on a duplicate. When a question comes up in month three — "did the contractor really record that tree as dead?" — the raw file is your only evidence.

Step 2: Clean the data before it touches ArcGIS

You can clean data after publishing, using the field calculator and editing tools (Chapter 13), but cleaning in a spreadsheet first is faster, safer, and reversible. Work through the problems in a deliberate order.

Rename the headers. ArcGIS will turn column headers into field names, and field names live forever in expressions, pop-ups, and code. Convert them now to lowercase, underscore-separated, unambiguous names: `tree_id`, `species_common`, `dbh_in`, `height_ft`, `condition`, `planted_date`, `last_inspected`, `latitude`, `longitude`, `notes`, `crew`. Note the unit suffixes — `dbh_in` tells every future reader the number is inches without a data dictionary. Chapter 12 covers naming in depth.

Standardize the categories. Build yourself a small lookup: every messy species spelling on the left, the canonical form on the right. Use find-and-replace to collapse the variants, and split species into two columns — `species_common` (`Red Maple`) and `species_botanical` (`Acer rubrum`) — because the director's chart wants common names while the city arborist wants botanical ones. Collapse the condition column to exactly four values: `Good`, `Fair`, `Poor`, `Dead`. That `fair-ish` becomes `Fair`; the `DEAD?` gets flagged for a field check and recorded as `Poor` in the meantime, with a note.

Fix the dates. Convert every date to the ISO format `2025-06-14` (year, month, day). It is the one format that nothing misreads, and it sorts correctly even as plain text.

Audit the coordinates. Sort by latitude. Any value that is negative, or wildly outside Milbrook's range, is either swapped with its longitude or wrong. Swapped pairs are a two-column fix. The rows with blank coordinates cannot be mapped at all — cut them into a separate "needs field visit" file rather than deleting them. Chapter 37 shows how a field crew resolves exactly this kind of backlog.

Resolve the duplicates. For each doubled `tree_id`, keep the row with the later inspection date. Your `tree_id` is about to become the key that ties every future inspection to a physical tree, so it must be unique.

> **Watch out:** Spreadsheet programs love to "help." Excel will silently reformat your ISO dates back to its regional format, strip leading zeros from ID codes, and

convert long numbers to scientific notation — sometimes just from opening and re-saving the file. After any save, spot-check a few known-tricky cells before moving on.

Step 3: Design the schema on paper

The schema is the layer's skeleton: which fields exist, what type each one is, and what values it accepts. Chapter 12 is the full treatment; here is the applied version. Ten minutes with a notepad now prevents the most painful category of rework later, because while *adding* a field to a published layer is easy, *changing a field's type* essentially is not — you rebuild and republish.

FIELD	TYPE	NOTES
`tree_id`	Text	Unique key. Text, not a number — IDs like `T-00042` carry leading zeros, and you will never do math on them.
`species_common`	Text	Constrained by a list of values (a domain) after publishing.
`species_botanical`	Text	Free text; maintained by the arborist.
`dbh_in`	Double (decimal number)	Diameter at breast height, inches. Numeric so charts can average it.
`height_ft`	Double	Estimated height, feet.
`condition`	Text	Domain: Good, Fair, Poor, Dead.
`planted_date`	Date	Blank for older trees is acceptable.
`last_inspected`	Date	Drives the "overdue" list on the dashboard.
`crew`	Text	Internal only — will be hidden from any public audience.

FIELD	TYPE	NOTES
`notes`	Text	Internal only, generous length.

Two decisions here deserve a sentence each. First, `condition` is text with a **domain** — a fixed list of allowed values — rather than free text, because every chart, filter, and color ramp downstream depends on there being exactly four condition values, forever. Domains are the mechanism that keeps `fair-ish` from ever coming back (Chapter 12). Second, the measurements are numeric types, because a number stored as text can be displayed but not summed, averaged, or sized by — and Smart Mapping (Map Viewer's engine for suggesting data-driven styles, Chapter 7) and Dashboards both need real numbers.

Step 4: Publish the hosted feature layer

A **hosted feature layer** is your data living in Esri's cloud as a queryable service: apps ask it questions ("how many trees where condition is Poor?") and it answers, which is precisely what a live dashboard needs. Chapter 10 covers the machinery; here is the walkthrough.

Sign in to ArcGIS Online and go to `Content > New item`. Drag in the cleaned CSV. When prompted, choose to add the file **and create a hosted feature layer** — the other option merely stores the file, like a document in a drive, and cannot power a map. The publishing wizard then asks two important sets of questions.

Location. Tell it your coordinates live in the `latitude` and `longitude` fields. ArcGIS will assume plain latitude-longitude values are in WGS84 — the global standard used by GPS — which is what a field tablet produces, so accept it. If your coordinates ever come from a state or national grid system instead, this is the moment it matters; Chapter 3 (Coordinate Systems) explains how to recognize and declare the difference.

Field types. The wizard shows each column with its guessed type. Check every one against your schema table. This is the moment a guess becomes permanent: if `dbh_in` shows as a string because one stray cell contained text, fix the cell or override the type *now*. Confirm dates registered as date fields, not text.

Give the layer a name that declares its role — `Milbrook_Trees_master` — and publish. Then open the new layer's item page and do three pieces of immediate housekeeping. In the settings, turn on **delete protection**, the checkbox that prevents the item from being accidentally deleted; this layer is about to become the single source of truth for the whole project. In the `Data` tab, switch to the fields view and attach the domains you planned — for `condition`, edit the field and define its list of values. And fill in the item's summary and description while you still remember what everything means; Chapter 5's advice about evaluating other people's metadata applies double to writing your own.

> **Tip:** Adopt a naming convention on day one and never deviate: the word `master` in the master layer's name, and each view named for its audience. Six months from now, `Milbrook_Trees_master` and `Milbrook Trees – Field Edit view` will be self-explanatory in a content list; three items all named `trees` will not be.

Step 5: Set up the master-and-views pattern

Here is the architectural move that separates a durable project from a fragile one. You are about to have three audiences with three different needs: field crews who must *edit* the data, the parks leadership who must *see* everything but change nothing, and — someday — the public, who should see most trees but not internal notes or staff names. One layer cannot safely serve all three.

A **hosted feature layer view** solves this. A view is not a copy — it is a separate item that reads the master layer's data live, but carries its own independent settings: its own sharing, its own editing permissions, its own visible fields, its own row filter. Edit a tree through one view and every other view reflects it instantly, because underneath there is only one dataset. Chapter 10 explains the internals; the practice looks like this.

From the master layer's item page, choose `Create View Layer`. Make two views:

- **`Milbrook Trees – Field Edit view`.** In this view's settings, enable editing (allow updates and adds). This is the only path through which anyone touches the data, and it will be shared only with the field crew's group.

- `Milbrook Trees — Reporting view`. Editing off. This read-only view is what the web map, and therefore the dashboard, will consume. If leadership should not see the `crew` and `notes` fields, hide them in this view's field settings; Milbrook's director wants them, so this view shows all fields.

The master itself is now shared with *no one* and touched by *no app*. Every future consumer plugs into a view. When the city later wants a public tree map, you will not rebuild anything — you will create a third view that hides `crew` and `notes`, filters out `Dead` trees, and gets shared publicly, all without the master or the other views changing at all.

> **Watch out:** It is tempting to skip views and point everything at the master layer "just for now." The trap springs the first time you need to give someone partial access — you cannot hide fields or disable editing for one audience on a single layer, so you end up either over-sharing or rebuilding every app against new layers. Views cost two minutes now; retrofitting them means re-pointing every map and app later.

Step 6: Build and style the map

Open Map Viewer (Chapter 6 is the complete reference) and start a new map. Add the **Reporting view** — not the master — via `Layers > Add`. For a basemap — the reference map that sits underneath your data — pick a muted, low-color canvas; nine thousand dots need a quiet background, a cartographic judgment Chapter 4 unpacks.

For styling, the director's first question is "where are the problem trees?", so style by `condition`. Select the layer, open the styling panel, choose `condition` as the attribute, and pick the unique-values style — one color per category. Smart Mapping (Chapter 7) will propose colors; override them with intuition-matching ones: green for Good, yellow for Fair, orange for Poor, a desaturated gray-brown for Dead. Resist pure red-versus-green as the only distinction — a meaningful fraction of viewers cannot tell them apart, which is why the orange and the gray are doing real work here (Chapter 4 covers color accessibility).

At citywide scale, 8,700 points overlap into mush. Two remedies, both in the layer's display settings: enable clustering or aggregation so dense areas summarize themselves

at small scales, or set a visible-scale range so individual trees appear only as you zoom in. Milbrook uses clustering, styled to show the count. Skip labels for now — nine thousand labeled points is noise, not information (Chapter 9 covers when labels earn their place).

Save the map as `Milbrook Tree Inventory`.

Step 7: Pop-ups that answer the next question

Click any tree and the default pop-up dumps every field, raw. Chapter 9 covers pop-up craft fully; the applied version takes five minutes. In the pop-up configuration, set the title to the species and ID, and trim the field list to what a human standing at that tree wants: species, condition, trunk diameter, height, planted date, last inspection, notes.

One number the director will always want is not stored anywhere: *how long since this tree was inspected?* Storing it would be a mistake — it goes stale the moment you write it. Compute it instead with **Arcade**, ArcGIS's built-in expression language (Chapter 8). Add a new pop-up expression:

```
Round(DateDiff(Now(), $feature.last_inspected, 'years'), 1)
```

Name it "Years since inspection" and it appears in the pop-up like a field, always current, calculated fresh at click time. This one-liner is Arcade's whole value proposition in miniature: derived facts without stored staleness.

Step 8: Assemble the dashboard

A **dashboard** is a single screen of live, linked elements — maps, counters, charts, lists — all reading from the same data (Chapter 28 is the complete reference). From the saved web map's item page, choose `Create app > Dashboards`, name it `Milbrook Tree Program`, and the map arrives as the first element. Now add the elements the director asked for; each one, when added, asks for a data source — point every one at the tree layer *inside the web map*, so they inherit the map's configuration.

ELEMENT	CONFIGURATION	WHAT IT ANSWERS
Indicator	Count of all features	"How many trees do we have?"
Indicator	Count filtered to `condition = Poor`	"How big is the problem?"
Pie chart	Categories from `condition`, colors matched to the map	"What shape is the inventory in?"
Serial (bar) chart	Categories from `species_common`, sorted by count, top slice only	"What are we growing?"
List	Sorted by `last_inspected`, oldest first	"Who needs a visit?"
Category selector	Values from `species_common`; lives in the dashboard's header or side panel rather than the body	Interactive filter for everything else

Two configuration habits make the difference between a screen of widgets and an instrument. First, **match the colors**: the pie chart's condition colors should be identical to the map's, so the reader's eye learns the encoding once. Second, and this is the step people miss, **wire the elements together with actions**. Elements do not talk to each other until you tell them to.

Open the category selector's settings and find its actions: add a **filter** action targeting the map, both indicators, the pie chart, and the list. Now choosing `Red Maple` in the selector filters the entire dashboard to red maples — the map dims to just those points, the counts recalculate, the overdue list narrows. Then open the *map element's* actions and set its extent change — the extent is simply the area the map currently shows — to filter the indicators and charts. Now the numbers follow the map: pan to the Riverside neighborhood and the dashboard reports on Riverside. This selector-filters-everything, map-extent-filters-numbers wiring is the standard pattern for an operations dashboard, and Chapter 28 catalogs its variations.

Test it like a skeptic: click a pie slice, pick a species, pan the map, and verify every number changes when it should and only then. Then save.

Step 9: Share it with the right people

Nothing you have built is visible to anyone else yet — new items in ArcGIS Online are private by default. Sharing is controlled through **groups**: named collections of members that items can be shared to (Chapter 34 covers the administration side).

Create a group called `Milbrook Parks – Tree Program` and invite the director and her staff. Then share into it — and here is the critical mental model — **the whole chain, not just the dashboard**. A dashboard is a window onto a map, which is a window onto a view layer. A member opening the dashboard needs permission to all three. Share the dashboard, the web map, and the *Reporting view* to the group. The master layer stays unshared — nobody needs it, because nothing points at it. Separately, share the *Field Edit view* to the field crew's group. Because editing is enabled on that view itself, crew members need nothing more than membership and access to edit trees from their phones — no special group capability is involved in editing *features*, only in co-owning *items*, which is a different situation entirely (Chapter 30 covers the field editing setup; Chapter 34 covers group capabilities).

> **Watch out:** The classic sharing failure is a dashboard shared correctly whose *layer* is not — the recipient sees the dashboard frame, empty charts, and a map with a missing layer. Because you, the owner, can see everything you own, it all looks fine on your screen. ArcGIS Online generally warns about under-shared dependencies when you share an app, but do not rely on the warning: test as your audience. Have a group member open the dashboard, or use a second test account. Chapter 39 has the full diagnostic tree for "it works for me but not for them."

The director opens `Milbrook Tree Program` on Monday morning. It works. You are done — for nine days.

Step 10: The mid-project schema change

Then the email arrives: the director wants to plan next quarter's maintenance, so every tree needs a **priority** — High, Medium, Low, or None — and the dashboard should filter

by it. This is the moment the architecture either pays off or collapses. Because you built master-and-views, it pays off. Here is the disciplined sequence for changing a schema that live apps depend on.

Change the master, not a view. Fields belong to the underlying dataset. On the *master* layer's item page, open `Data > Fields` and add a field: name `priority`, type text. Define its domain — High, Medium, Low, None — right there. Views read the master's schema, so the field now exists everywhere at once.

Check each view's field list. A view that exposes all fields will pick up the new one automatically; a view configured with an explicit list of visible fields will not show newcomers until you edit that view's field settings and include it. Check both views — thirty seconds each — because a field that a view cannot see is a field its dashboard cannot chart.

Backfill the values. The new field is empty on all 8,700 rows, and empty fields make lifeless charts. Rather than hand-editing, calculate initial values from data you already trust: in the master layer's `Data` tab, select the `priority` field and use its calculate option with a small Arcade expression — condition `Poor` becomes `High`, `Fair` becomes `Medium`, `Good` becomes `Low`, `Dead` becomes `None` (removal is a different workflow). It is a starting point the arborist can refine tree by tree through the Field Edit view (Chapter 13 covers editing workflows; Chapter 22 covers calculation at scale in Pro).

Ripple upward through the stack. The data knows about priority; the apps do not yet. Add `priority` to the map's pop-up field list. Then, in the dashboard, add a second category selector driven by `priority` and wire its filter action to the same targets as the first. Ten minutes, no rebuilds, and nothing the director had bookmarked ever broke or moved.

> **Tip:** The safe schema-change vocabulary is *add*, never *rename* or *delete*. Every pop-up expression, dashboard element, filter, and Arcade snippet refers to fields by name; rename a field and each of those references breaks quietly, in whatever app you forgot to check. If a field is truly wrong, add its replacement, migrate the

values, and retire the old field from view configurations — leaving the column itself in place until you are certain nothing references it.

And notice what did *not* happen during any of this: downtime. Crews kept editing through the Field Edit view, the director's dashboard kept answering, and the master quietly grew a new column. That is the master-and-views pattern earning its keep.

The pattern worth stealing

Strip away the trees and the project is a template you will reuse for potholes, streetlights, inspections, assets of any kind:

1. **Quarantine the raw data**, then clean a copy: standardized headers, controlled categories, ISO dates, audited coordinates, unique keys (Chapters 5, 11).
2. **Design the schema on paper** before publishing, because types are nearly permanent and fields are forever (Chapter 12).
3. **Publish one master layer**, protect it, document it, and share it with no one (Chapter 10).
4. **Create a view per audience** — edit access for the few, read access for the many — and point every app at a view (Chapter 10).
5. **Build the map once, deliberately**: quiet basemap, meaning-carrying colors, computed values via Arcade instead of stored staleness (Chapters 6, 7, 8, 9).
6. **Wire the dashboard**, don't just populate it — selectors and map extent driving every element (Chapter 28).
7. **Share the whole chain to groups**, then test as your audience, not as yourself (Chapter 34).
8. **Change schemas at the master and ripple outward**, adding rather than renaming (Chapters 12, 13).

Milbrook's story continues in Chapter 37, where the field crew takes over: the blank-coordinate backlog gets resolved tree by tree with Field Maps, and the same master layer you built here becomes the backbone of a full field collection operation. When something in your own version of this project misbehaves, Chapter 39 (The Troubleshooting Encyclopedia) is organized around exactly the symptoms this pipeline

can produce. And when you need just one step of this chapter in isolation — "how do I add a field again?" — Chapter 38's recipes have you covered.

Worked Project: A Field Collection Operation End to End

Every summer, the Parks and Recreation department of a mid-sized city inspects the playground equipment in its forty-one parks. Until now, that meant a clipboard, a paper checklist per structure, a digital camera, and a week of someone retyping everything into a spreadsheet that disagreed with last year's spreadsheet. This chapter follows the whole operation as it moves to ArcGIS: designing the data, building the mobile experience in Field Maps, surviving a real collection day, reviewing the results, reporting them, and — the part most write-ups skip — looking back honestly at which design decisions held up and which didn't.

The cast: you are the GIS coordinator. Maria and Dev are the two seasonal inspectors who will do the field work. Sandra is the parks supervisor who wants to see progress without asking anyone. None of the three of them care about GIS; they care about broken swings.

This chapter leans on ideas explained in depth elsewhere — schema design in Chapter 12, the Field Maps product family in Chapter 30, hosted feature layers in Chapter 10, dashboards in Chapter 28. Here you'll see them assembled into one working system, in order, with the mistakes left in.

Design the Form First, Then the Schema

The single most useful habit in field data collection is to design backwards from the form the inspector will see, not forwards from the data you imagine wanting. So before touching any software, you sit down with Maria, Dev, and a printout of last year's paper checklist, and ask: what do you actually record, per what thing, how often?

The answers shape everything:

- They inspect individual pieces of equipment — a slide, a swing set, a climber — not whole playgrounds. A park can have a dozen pieces.
- Each piece gets inspected repeatedly: once in the summer sweep, again if a complaint comes in, again after repairs.
- Per inspection they record an overall condition, a handful of specific checks (hardware, surfacing, entrapment hazards), photos, and free-text notes only when something is wrong.
- The piece of equipment itself has facts that never change per inspection: what it is, who made it, when it was installed.

That last distinction is the pivot of the whole design.

One layer for assets, one table for inspections

The naive design is a single point layer where every inspection is a new point, with the equipment details typed in every time. It feels simple and it rots immediately: three inspections of the same slide become three points in slightly different places, with "Slide", "slide - big", and "SLIDE (yellow)" as the equipment type, and no way to ask "show me this slide's history."

The durable design is two datasets joined by a relationship:

1. **The equipment layer** — a point feature layer (a layer where each record has a location on the map) with one point per physical asset. It changes rarely: when equipment is installed, replaced, or removed.
2. **The inspections table** — a related table with no geometry of its own. Each row is one inspection of one asset, stamped with a date. It grows forever, and that's fine.

The link between them is a **relationship**: a formal one-to-many connection where one equipment point can own many inspection rows. In ArcGIS the tidiest way to wire this is with GlobalIDs. A **GlobalID** is a long, automatically generated identifier that ArcGIS guarantees is unique — you never type it, never see it in the form, and never worry about two inspectors inventing the same asset number. The equipment layer's GlobalID field is the origin key; the inspections table carries a matching **GUID field** (a field typed to hold those identifiers) that points back at its parent. Chapter 12 (Schema

Design) covers the mechanics; here it's enough to know that when Maria taps a slide in Field Maps and adds an inspection, the app fills in that linking field silently.

> **Tip:** The test for "asset field or inspection field?" is one question: does the answer change between visits? Manufacturer — no, asset field. Condition — yes, inspection field. Anything you catch yourself wanting in both places belongs on the inspection, because the asset copy will drift out of date.

The equipment layer, field by field

FIELD	TYPE	DOMAIN?	WHY IT EXISTS
Asset ID	Text	No	Human-readable tag matching the metal ID plate on the equipment
Equipment type	Text	Yes — coded values	Slide, Swing set, Climber, Spinner, Spring rider, Composite structure, Other
Park name	Text	Yes — coded values	Which park it belongs to
Manufacturer	Text	Yes — coded values	The handful of vendors the city buys from, plus Unknown
Install year	Integer	No	Age drives replacement planning
Status	Text	Yes — coded values	In service, Out of service, Removed
Notes	Text	No	The escape hatch

A **domain** is a rule attached to a field that restricts what values it can hold. A **coded value domain** is the most common kind: a fixed pick-list, where each entry has a stored code and a friendly label. Domains are the difference between a field that says `Swing set` forty-one times and one that says `swing`, `Swings`, `swing set (4)`, and `sw` — which is to say, the difference between data you can count and data you can only

read. On a phone, domains have a second job: they turn typing into tapping, and tapping is what people actually do while standing on wood chips in the sun. Chapter 12 goes deep on domain strategy; the one-line version is *anything you will ever want to count, chart, or filter gets a domain*.

The inspections table, field by field

FIELD	TYPE	DOMAIN?	WHY IT EXISTS
Inspection date	Date	No	When it happened
Inspector	Text	Yes — coded values	Who did it (Maria, Dev, plus room to grow)
Overall condition	Text	Yes — coded values	Good, Fair, Poor, Unsafe
Hardware check	Text	Yes — coded values	Pass, Fail, Not applicable
Surfacing check	Text	Yes — coded values	Pass, Fail, Not applicable
Entrapment check	Text	Yes — coded values	Pass, Fail, Not applicable
Action needed	Text	Yes — coded values	None, Monitor, Repair, Remove from service
Comments	Text	No	Required only when something failed — more below
QA status	Text	Yes — coded values	Needs review, Approved, Returned to field

Two of these deserve a word. **Action needed** exists because Sandra's real question isn't "what condition is it in?" but "what do I have to do about it?" — and making the inspector answer that on site, while looking at the thing, produces far better answers

than inferring it later at a desk. **QA status** exists because you, not the inspectors, will be the last set of eyes on every record; it's the field that drives the review loop in a later section. It defaults to *Needs review* so nothing slips through unexamined.

Photos are handled by **attachments** — files stored with a record rather than in a field. You enable attachments on the inspections table, not the equipment layer, so every photo is tied to the visit where it was taken.

You also switch on **editor tracking**, a layer setting that automatically stamps each record with who created or last edited it and when. It costs nothing and quietly answers half of all future "wait, who changed this?" questions. Note that the Inspector domain field stays anyway — editor tracking records the ArcGIS account, and on a shared device that's not always the human holding the phone.

Building and Publishing the Layers

There are two reasonable construction paths. Field Maps Designer — the browser-based companion app covered in Chapter 30 — can create a new layer with fields and pick-lists directly, no desktop software involved, and for a single standalone layer it's the fastest path in the ecosystem. But the equipment-to-inspections relationship is easiest to build properly in ArcGIS Pro, so that's the route here.

In Pro, you create a file geodatabase, build the two datasets and their domains, and create the relationship class between them using GlobalID as the origin key (Chapter 12 walks through this exact operation). You add a dozen equipment points for one park by hand — real ones, from memory and the paper records — so that testing happens against plausible data rather than an empty map. Then you publish the whole thing as one **hosted feature layer** (Esri's term for a layer that lives in ArcGIS Online — or your ArcGIS Enterprise portal — and can be read and edited over the web; Chapter 10 is the full treatment), making sure of three settings on the resulting item, all reachable from the item's page under its settings:

- **Editing enabled**, so field edits can be saved at all.
- **Sync enabled**, which is the capability that permits taking data offline and merging changes back later. No sync, no offline — this is the setting people forget.
- **Attachments enabled** on the inspections table, for photos.

Watch out: Get the schema as close to final as you can *before* collection starts. Adding a field mid-operation is usually survivable; renaming, retyping, or deleting one after data exists ranges from annoying to destructive, and any change to the layer generally forces every field device to remove and re-download its offline area. Schema surgery during a live operation is how sync errors are born.

The remaining equipment points — the other forty parks' worth — get loaded from last year's spreadsheet, geocoded (matched to locations on the map from their park names and descriptions) and then nudged into position over aerial imagery. Last year's checklist results ride along too, loaded as a first generation of inspection rows, so each asset starts the season with history attached. Imperfect positions are acceptable here, and knowing *why* they're acceptable matters: the inspectors will correct obviously misplaced points in the field, and the point's job is to be tappable and unambiguous, not survey-grade. Chapter 11 (Creating Data) covers these load-in paths.

Configuring the Experience in Field Maps Designer

A hosted layer is not a field app. The layer defines what *can* be stored; Field Maps Designer defines what Maria and Dev actually see. You start by making a web map in Map Viewer (Chapter 6) containing the equipment layer and, beneath it, the parks boundaries for context, styled simply: equipment symbolized by its Status field so out-of-service items look visibly different. Then you open that map in Field Maps Designer — `App launcher > Field Maps Designer`, then pick the map — and work through its sections.

The form

The form builder is a drag-and-drop canvas: fields on one side, the form layout in the middle, per-element settings on the other. You build two forms — one for the equipment layer (used rarely, when adding a new asset) and one for the inspections table (used constantly). For the inspections form:

- Order follows the inspector's physical walk-around: condition first, then the three checks, then action, then comments. Matching the form to the body's route around the equipment is worth ten minutes of anyone's time.

- Overall condition, the three checks, and Action needed are marked **required** — the app won't let a record save without them.
- Inspection date gets a default of the current date.
- Inspector is required; QA status is hidden from the form entirely, left to its *Needs review* default, because it belongs to the office, not the field.
- Comments uses **conditional logic**: a rule, written as a small expression in Arcade (ArcGIS's built-in scripting language for exactly this kind of job — Chapter 8), that makes the field appear — and become required — only when Overall condition is Poor or Unsafe, or any check failed. When everything passes, the form stays short. When something's wrong, the app itself insists on an explanation.

That last pattern — *short form on the happy path, demanding form on the failure path* — is the single highest-leverage form design trick in field work. It respects the inspector's time on the ninety percent of equipment that's fine and concentrates rigor exactly where it pays.

> **Tip:** Before anyone drives anywhere, install Field Maps on your own phone, open the map, and fake three inspections standing in the parking lot — including one failure so the conditional comments logic fires. Every form has one field that seemed fine on a monitor and is maddening on a phone in sunlight. Find it now, not in front of two inspectors on day one.

Sharing

The map, the layer, and (shortly) the dashboard all go into a group — "Playground Inspections 2026" — and Maria's, Dev's, and Sandra's accounts join it. The inspectors need account types that permit editing data; Sandra only needs to view. Chapter 34 (Administration) covers user types and why the cheapest license that can edit is usually the right one for seasonal field staff.

Going Offline

City parks are exactly where cellular coverage goes to die — and even where coverage is fine, an app that assumes connectivity punishes you at random. So the operation is designed offline-first: devices carry the data with them and reconcile later.

In Field Maps Designer's offline settings for the map, you define **offline map areas**: named, pre-packaged chunks of the map — data plus basemap — that a device downloads once, over Wi-Fi, before heading out. You draw four areas covering the city's four quadrants, roughly ten parks each, rather than one city-wide area. Smaller areas download faster, sync faster, and fail smaller.

The basemap deserves thought. Aerial imagery is genuinely useful in the field — "which of these three swing sets am I standing at?" is answered instantly by the photo beneath the points — but imagery is heavy, and package size grows explosively with each additional zoom level of detail you include. You cap the detail at "clearly see individual play structures" and accept that inspectors can't zoom in to blades of grass. Each quadrant package lands at a size a phone downloads over office Wi-Fi in a few minutes.

Sync — the process that uploads a device's local edits and pulls down everyone else's — is two-way and incremental: only changes travel, not the whole dataset. Field Maps can also auto-sync opportunistically whenever it finds a connection, which you leave on.

> **Watch out:** The offline map area is a copy, frozen at download time. If you edit the web map's styling, or another inspector adds an asset that a colleague's stale area doesn't know about, the fix flows through sync or a re-download — not automatically. Establish one ritual: every device syncs at the start of the day, at lunch, and before going home, whether or not anything seems to need it. Devices that go days without syncing are where data goes to disappear.

Collection Day

Monday, 7:40 a.m. Maria and Dev sit in the office with coffee, downloading their quadrants over Wi-Fi — Maria takes northwest, Dev takes northeast. You watch them each open the offline area, confirm the parks render, and fake-then-delete a test inspection. Then they leave, and the operation stops being a diagram.

8:55 a.m., first park. Maria's blue location dot sits confidently in the parking lot. She walks to the first swing set, taps its point, taps the button to add a related inspection record, and the form does its job: five taps for condition and checks, action *None*, no comments demanded. She takes one wide photo of the whole structure. Ninety seconds,

done. The related-records design is already paying off — the swing set's point shows last year's inspection right below this one.

9:30 a.m., GPS under the trees. The third park is the wooded one, and here comes the classic problem: under tree canopy, GPS accuracy degrades badly. Maria adds a newly discovered spring rider — not in last year's inventory — and the app wants to place it fifteen-odd meters into a picnic area, with the on-screen accuracy readout showing a number far worse than the open-sky figure. Two remedies, both worth teaching before day one. First, **GPS averaging**, a Field Maps setting that collects a stream of position fixes over several seconds and uses their average, smoothing the worst jitter. Second — and better here — she places the point manually instead of trusting GPS: while adding the feature, she pans the map until the collection crosshair sits on the playground footprint she can *see* in the aerial imagery, and drops the point dead-on. Survey-grade GPS receivers exist for work where centimeters matter; for "which piece of equipment is this," imagery-assisted placement is free and sufficient. Chapter 30 covers the high-accuracy receiver world.

11:15 a.m., a failure. Dev, across town, finds a climber with a cracked weld. Condition *Poor*, hardware check *Fail* — and the comments field materializes, required, exactly as designed. He writes two sentences, then photographs the crack. Here's where photo discipline matters: he takes one wide context shot and one close-up of the defect, and in the close-up he holds his thumb next to the crack for scale. That convention — wide shot always, close-up per defect, something for scale — was on the one-page cheat sheet you printed, and it will make the QA reviewer's life (yours) enormously easier than forty ambiguous mid-distance photos would.

12:30 p.m., lunch sync. Both inspectors are told to sync from anywhere with signal. Maria's sync completes in under a minute — remember, only the changes travel, though photos make up most of the payload. Dev's sync fails with a network error mid-upload. This is the moment untrained field staff panic and, in the worst version of the story, delete and reinstall the app — destroying every unsynced record. The rule you drilled: **a failed sync loses nothing.** The edits are still on the device; find better signal and try again. He retries from the sandwich shop's Wi-Fi and it goes through.

2:00 p.m., the collision. Both inspectors, finishing early, independently head to the same central park. Maria inspects the composite structure at 2:05; Dev inspects it at

2:20, unaware — his offline copy can't see her unsynced record. Is this a disaster? Walk through what sync will actually do. They didn't edit the same row: each *added* a new inspection record, and additions never conflict — the structure simply ends up with two inspections dated the same day, which QA will spot and resolve by keeping the more complete one. A true **sync conflict** — two devices editing the *same existing record* while both offline — is resolved by ArcGIS with a last-in-wins rule at sync time, which is precisely why this schema pushes nearly all field activity into *new* inspection rows and leaves the equipment records almost untouched. The design didn't prevent the collision; it made the collision boring.

4:45 p.m., back at the office. Both devices sync on Wi-Fi. You open the web map at your desk and watch the day materialize: thirty-one equipment points carrying fresh inspections, two brand-new assets, one cracked weld glowing in the attachments. Day one of eight, done.

> **Tip:** Print a one-page field cheat sheet: the sync ritual, the photo convention, the under-canopy placement trick, and "a failed sync loses nothing — retry, never reinstall." The app is good; the app plus a laminated page is a system.

The QA Review Loop

Tuesday morning, before the inspectors head out, you spend forty minutes being the second pair of eyes. This is the loop that separates a defensible inspection program from a pile of phone data, and the QA status field is its engine.

Your review setup is a second web map — the QA map — that only the office sees. It carries the same layers, but the inspections are filtered to *QA status = Needs review*, and the equipment points are styled by whether their latest inspection passed or failed. For each pending record you check four things:

1. **Completeness.** Required fields made emptiness impossible, so this is really a photo check: does every failed check have a legible photo behind it?
2. **Plausibility.** A brand-new composite structure rated *Poor* with no comment is probably a fat-finger. A *Fail* on surfacing with a photo of pristine rubber tile is a mistap.

3. **Geometry.** Any new assets sitting in ponds or parking lots get nudged onto the playground using the imagery.
4. **Duplicates.** Monday's double-inspection of the composite structure shows up immediately as two same-day rows under one asset; you keep Dev's (it had better photos) and delete Maria's.

Each record then gets its QA status flipped: *Approved* if clean, *Returned to field* if something needs a re-visit. Returned records aren't edited at the desk into correctness — the inspectors' map filters inspections to show *Returned to field* prominently, so Maria and Dev see their own bounce-backs the next time they're near that park and fix them at the source. Desk-editing field observations you didn't personally make is how inspection data quietly turns into fiction; the loop always routes corrections through the person with eyes on the equipment. Chapter 14 (Data Quality) treats validation and QA workflows in full generality; this stripped-down version — a status domain, a filtered QA map, a daily habit — is the level most operations really need.

Two records on Tuesday fail plausibility, both mistaps, both returned and fixed within the day. The loop costs forty minutes each morning and converts "we collected data" into "we stand behind this data."

Reporting: Sandra's Dashboard

Sandra's requirement, verbatim: "I want to know how far along we are and what's broken, without calling anyone." That is a dashboard requirement, and Chapter 28 covers ArcGIS Dashboards completely; here is the minimal build that satisfied her, four elements on one screen:

- **A gauge of completion.** Equipment inspected this season versus total in-service equipment. This is the number Sandra opens the page for.
- **An indicator of open problems.** A count of inspections where Action needed is Repair or Remove from service and the season is current — with the element styled to look alarming when it's above zero. One item, the cracked climber, shows on day one.
- **A bar chart by park.** Failed checks grouped by park name, which took no preparation *because park name was a domain* — a foreshadowing of the retro, since

the chart is only trustworthy because the values were only ever tappable, never typable.

- **A map** of the equipment, symbolized by latest condition, so "what's broken" is also "where."

The dashboard reads the same hosted layer the phones write to, so it updates as syncs land — Sandra watches the gauge climb during the week without anyone compiling anything. You share it to the group, send her the link, and the Friday status meeting becomes five minutes long.

> **Watch out:** A dashboard aggregates whatever has synced — including the not-yet-reviewed. Sandra should either see numbers filtered to *QA status = Approved*, or be told plainly that today's figures are provisional until the morning review runs. Choose one; the silently mixed version is how a mistapped *Unsafe* becomes a councilmember's phone call.

The Retro: What Held Up and What Didn't

Eight collection days later, forty-one parks and a few hundred inspections are in. The honest scorecard:

DECISION	VERDICT	WHY
Separate asset layer + related inspections table	Right, emphatically	History per asset, boring sync conflicts, next season starts with zero schema work
Domains on everything countable	Right	The dashboard's charts worked untouched; not one value needed cleanup
Conditional, required comments on failures	Right	Every failure has an explanation; no passing record wasted anyone's time

DECISION	VERDICT	WHY
QA status field driving a review loop	Right	Caught mistaps within a day; made "Approved" mean something
Pre-built offline areas by quadrant	Right	Zero connectivity drama all week
Equipment type domain's *Other* category	Wrong-ish	Eleven assets landed in *Other*; the domain lacked See-saw and Balance beam. Fixable — coded value domains can gain entries with data in place — but each *Other* now needs a human to reread notes and reclassify
No "last inspected" info surfaced on the asset	Wrong	Inspectors had to open related records to see whether a colleague had already covered a park; a map styled by days-since-last-inspection (an Arcade job — Chapter 8) would have prevented the day-one collision outright
Required Inspector pick-list	Half-right	Editor tracking already captured accounts; the field earned its keep only on the one afternoon the inspectors shared a device with a dead battery's login
Asset ID as free text	Wrong	Two typos against the physical ID plates surfaced during QA; a stricter input pattern, or barcode labels scanned by Field Maps, is the season-two fix

The pattern in the misses is worth stating: every one is a place where the schema trusted a human to type or remember something the system could have constrained or displayed. The pattern in the hits is the mirror image — everywhere the design made the right action the easy action (tapping a domain, a form that demanded explanations only for failures, sync that forgave), the data came back clean without anyone being nagged.

Next season's prep is now an afternoon: extend the equipment type domain, add the days-since-inspection styling, print barcode stickers, re-share the same maps to a new group of seasonals. The system compounds — which is the real argument for doing the schema work up front, and the reason this chapter spent a third of its length on two tables and their pick-lists.

For the office-side sibling of this project — raw data to published analysis — see Chapter 36. For quick recipes that assume this kind of setup already exists, Chapter 38's cookbook has several entries on forms, offline behavior, and sync. And when something in your own operation breaks in a way this narrative didn't cover, Chapter 39 (The Troubleshooting Encyclopedia) has a whole section on sync errors and their cures.

Cookbook: Forty How-Do-I Recipes

Every recipe here answers a question that starts with "how do I..." and ends with a real task someone needed done by Friday. Each one follows the same four-part shape: **Goal** (what you get), **Path** (where the work happens), **Steps** (the shortest reliable route), and **Gotcha** (the one thing most likely to bite you). Recipes are terse on purpose. When you want the *why* behind a step, follow the chapter cross-reference; every recipe points back to the chapter that explains its machinery in full.

Unless a recipe says otherwise, assume you are signed in to ArcGIS Online and working in the current Map Viewer, the browser-based map editor with a panel of tools down each side. Esri renames buttons more often than it changes what they do, so paths describe the stable landmarks: the left side of Map Viewer holds *content* tools (layers, basemap, legend), the right side holds *settings* tools (styles, filters, pop-ups, labels). The other recurring landmark is the **item page**, the detail page you reach by clicking any item's title in `Content`. Item pages carry tabs, usually Overview, Data, Visualization, Usage, and Settings, and many recipes live there rather than in the map.

Tip: A good rule for guessing where a task lives: if it changes how a layer *looks in one map*, it belongs in Map Viewer. If it changes what a layer *is* everywhere (its data, its schema, its sharing), it belongs on the item page. That single distinction resolves most "where is that setting" confusion.

Maps and layers

1. Add a curated layer from Living Atlas

Goal: Bring authoritative, ready-made data (boundaries, demographics, environment) into your map. **Path:** `Layers > Add > Browse layers`, then switch the source to Living Atlas. **Steps:** Open the add-layer browser, choose Living Atlas as the collection to search, type a plain-English keyword, and click Add on a result. Click the item title first if you want to read its description and update cadence. **Gotcha:** Some Living Atlas layers are subscriber or premium content that requires sign-in for viewers or consumes credits, the usage currency your organization buys and spends on certain services (Chapter 34). Check the item's badge before building a public map on it. Chapter 5 (Finding Data) covers how to evaluate what you find.

2. Save your own copy of someone else's map

Goal: A map you can edit, based on a map you can only view. **Path:** Open the map, then `Save and open > Save as`. **Steps:** Open the shared map in Map Viewer, choose Save as, give it a new title and tags, and save into your own content. You now own the map configuration. **Gotcha:** Save as copies the *map*, not the *layers*. The layers still belong to their original owners; if one is deleted or unshared, your copy breaks too. To own the data itself, export or copy the layers separately (Recipe 28).

3. Filter a layer to show only some features

Goal: Display just the rows that match a condition, without deleting anything. **Path:** Select the layer, then open the filter tool in Map Viewer's settings panel. **Steps:** Add an expression, pick a field, pick an operator (equals, contains, greater than), and set the value. Stack multiple expressions with and/or logic. Save the map to keep it. **Gotcha:** A filter hides features in this map only; the data is untouched and other maps still show everything. If a filter seems to match nothing, check for stray spaces or capitalization differences in the field values.

4. Extract one state from a national layer

Goal: A standalone layer containing only, say, Colorado, cut from a nationwide dataset. **Path:** Filter first (Recipe 3), then the analysis tools in Map Viewer, or the item page's export options if you own the layer. **Steps:** Filter the national layer to your state. Run the extract-style analysis tool (look for a name like Extract Data or Export Features under `Analysis > Tools`); it honors the layer's active filter. Save the result as a new hosted layer, a layer whose data now lives in your own ArcGIS Online content. **Gotcha:** Extraction from someone else's layer requires that the owner allowed export, and analysis runs consume a small number of credits. Chapter 16 (Proximity and Overlay) explains the analysis toolbox this recipe borrows.

5. Swap a layer's data source without rebuilding the map

Goal: Newer data behind the same layer, with every map and app that uses it updating automatically. **Path:** The layer's item page, under `Update Data` (for layers published from a file). **Steps:** Prepare a replacement file with the *same fields and geometry type*. On the item page, choose the update or overwrite option and supply the new file. The layer keeps its item ID, so nothing downstream breaks. **Gotcha:** Overwriting replaces every existing record irreversibly, and it fails or mangles fields if the new file's schema differs from the old one. Export a backup first (Recipe 28). Chapter 10 (Hosted Feature Layers) covers overwrite versus append in depth.

> **Watch out:** Deleting a layer and re-adding a new one with the same name is *not* a swap. The new item has a new ID, and every map that referenced the old one now shows a broken layer. Always prefer overwriting in place.

6. Change or customize the basemap

Goal: A background map that suits your data instead of competing with it. **Path:** `Basemap` in Map Viewer's content panel. **Steps:** Open the basemap gallery and pick one. For a muted look under thematic data, choose a light or dark gray canvas. To go further, add any layer as a basemap layer or restyle a vector basemap with the basemap editor available from the gallery. **Gotcha:** Basemap choice is saved per map. If your organization uses a custom basemap gallery, options you saw in tutorials may be absent; an administrator controls that list (Chapter 34).

7. Group and reorder layers

Goal: A tidy layer list where related layers travel together. **Path:** The `Layers` panel; drag to reorder, use each layer's option menu to group. **Steps:** Drag layers up or down to control drawing order (top of the list draws on top of the map). Select multiple layers or use the layer menu to create a group; the group gets one visibility checkbox and its own name. **Gotcha:** Drawing order matters more than it looks: polygons above points will hide them. Keep points and lines above filled polygons unless you intend the opposite.

8. Set the map's opening view

Goal: The map opens exactly where and how you want, every time, for everyone. **Path:** Map properties in Map Viewer, plus `Bookmarks` for named views. **Steps:** Pan and zoom to the view you want, then use the map properties or save behavior to capture it as the initial extent, the area the map shows when it opens. Add bookmarks for other locations you or your users jump to often. **Gotcha:** In many setups, saving the map saves the current view as the opening view, so an absent-minded save while zoomed into one parcel becomes everyone's opening screen. Check the extent before you save.

9. Style the same layer two ways at once

Goal: One dataset shown twice, for example as colored polygons and as labeled outlines. **Path:** The layer's option menu in `Layers`; look for a duplicate or copy option. **Steps:** Duplicate the layer inside the map, rename both copies so you can tell them apart, then style and filter each independently. They share the same underlying data. **Gotcha:** Duplicates double the features the map has to draw. On big layers this can slow rendering noticeably; consider whether one layer with smarter symbology (Chapter 7) can do the job alone.

10. Copy a map to another account

Goal: A colleague in a different account or organization gets their own editable copy of your map. **Path:** A shared group, then Save as from their side. **Steps:** Create a group, invite the other account, and share the map (and any private layers it uses) to the group. The colleague opens the map and uses Save as (Recipe 2) to take their own copy. **Gotcha:** Their copy still points at *your* layers. For a fully independent handoff, they also need copies of the data. Within the same organization, an administrator can instead

transfer item ownership outright (Recipe 40); ownership cannot be transferred across organizations.

Styling, labels, and pop-ups

11. Label only some features

Goal: Labels on the important features, silence from the rest. **Path:** The layer's label settings in Map Viewer's settings panel. **Steps:** Enable labels and add a label class. Each label class accepts its own filter, so set a condition like population greater than 100,000, or status equals Active. Only matching features get that class's label. Add more classes for different tiers with different sizes. **Gotcha:** Label classes with overlapping filters produce doubled labels. Make the filters mutually exclusive. Chapter 9 (Pop-ups, Fields, and Labels) covers label classes and label text driven by Arcade, ArcGIS's built-in expression language, in full.

12. Color features by category

Goal: Each category (land use, status, party) gets its own color. **Path:** `Styles` in the settings panel, with the layer selected. **Steps:** Choose the category field as the attribute. Pick the types-style drawing option (unique symbols per value). Adjust individual colors by clicking each category's swatch; drag rare categories into an "other" bucket. **Gotcha:** More than about eight colors stops being readable as categories and starts being visual noise. Merge or group minor values. Chapter 4 (Cartographic Design) explains why.

13. Size symbols by a number

Goal: Bigger circles where the number is bigger. **Path:** `Styles`, with a numeric field chosen as the attribute. **Steps:** Pick the counts-and-amounts (size) drawing style. Set sensible minimum and maximum symbol sizes and check the histogram to see where your data actually falls, dragging the handles to clip outliers. **Gotcha:** One extreme outlier can flatten every other symbol to the minimum size. Clip the range or use a filter to handle the outlier separately.

14. Show density with a heat map

Goal: A glowing density surface instead of ten thousand overlapping dots. **Path:** `Styles` on a point layer; choose the heat map drawing style. **Steps:** Select the heat

map style, then tune the influence radius slider until clusters read clearly at your working zoom level. Optionally weight by a numeric field so big incidents count more than small ones. **Gotcha:** Heat maps are zoom-dependent: a radius tuned at city scale looks like soup at state scale. Decide the zoom your audience will use and tune there. Chapter 7 covers when a heat map misleads.

15. Make a layer semi-transparent

Goal: See the basemap or a lower layer through this one. **Path:** The layer's properties in the settings panel; look for transparency or opacity, and effects for finer control. **Steps:** Select the layer, open its properties, and drag the transparency slider. For polygon layers, consider transparent fills with solid outlines instead, set inside `Styles > symbol style`. **Gotcha:** Transparency applied to the whole layer also fades the outlines and labels with it. If you only want a lighter fill, change the fill color's own transparency in the symbol settings instead.

16. Show only a few fields in the pop-up

Goal: A pop-up with five useful lines instead of forty raw ones. **Path:** `Pop-ups` in the settings panel. **Steps:** Open the pop-up configuration, find the fields list content block, and use the select-fields control to uncheck everything you don't need. Reorder what remains and rewrite the display names into plain English. **Gotcha:** Hiding a field in the pop-up doesn't hide it from anyone who queries the layer or opens its table. For actual privacy, remove or restrict the field itself (Chapter 10 covers views that hide fields).

17. Format numbers and display names in pop-ups

Goal: "1,240,500" instead of "1240500.000000", and "Median income" instead of "MED_INC_CY". **Path:** `Fields` in the settings panel (this controls formatting for pop-ups and labels alike). **Steps:** Select the field, set decimal places, and turn on the thousands separator. Edit the display name. Repeat for each field your pop-up shows. **Gotcha:** The display name is a map-level courtesy; the real field name never changes, and Arcade expressions must use the real one.

18. Show a photo in the pop-up

Goal: A picture of the site, right in the pop-up. **Path:** `Pop-ups > Add content`, choosing an image block; or attachments if the layer stores photos. **Steps:** If each feature has a field containing an image URL, add an image content block and point its source at that

field using the curly-brace field placeholder. If photos were collected as attachments (Field Maps does this), add the attachments block instead. **Gotcha:** Image URLs must be publicly reachable over HTTPS, or your viewers see broken squares. Links to a photo *page* (like a sharing site's viewer) won't render; you need a direct link to the image file itself.

19. Show a computed value with Arcade

Goal: A pop-up line the data doesn't contain, like a percentage or a category derived from two fields. **Path:** `Pop-ups > Add content > Arcade`, or an attribute expression referenced in text. **Steps:** Add an Arcade element and write a short expression, for example dividing one field by another and rounding: `Round($feature.UNITS_SOLD / $feature.UNITS_TOTAL * 100, 1)`. Give it a friendly title and use it like any field. **Gotcha:** Guard against dividing by zero and null values; a one-line `IIf()` check saves an ugly error in the pop-up. Chapter 8 (Arcade) takes you from this one-liner to fluency.

20. Style by two attributes at once

Goal: Color shows one variable, size shows another, on the same symbols. **Path:** `Styles`, after adding two attribute fields. **Steps:** Add both fields as style attributes. Map Viewer's smart mapping offers combined styles such as color-and-size; pick it and tune each dimension separately. **Gotcha:** Two variables is the ceiling for most audiences. If you're tempted by a third, make two maps instead. Chapter 7 covers bivariate styles and when they genuinely help.

Data and fields

21. Turn a spreadsheet of addresses into points

Goal: A CSV of street addresses becomes a point layer on the map. **Path:** `Layers > Add > Add layer from file` in Map Viewer, or `Content > New item` and drag the file in. **Steps:** Supply the CSV, confirm the field types it guessed, and choose to locate features by address rather than coordinates; this runs each row through the geocoder, the service that converts addresses into map coordinates. Map your address columns to the parts it expects, then publish as a hosted layer. **Gotcha:** Geocoding street addresses consumes credits in proportion to row count; locating by latitude and longitude

columns is free. Spot-check a sample of the placed points before trusting all of them. Chapter 11 (Creating Data) walks through every import path.

22. Fix points that landed in the wrong place

Goal: Points that appeared off the coast of Africa (or in the wrong hemisphere) land where they belong. **Path:** Usually the import step itself, re-run. **Steps:** Points at exactly zero latitude, zero longitude mean empty coordinate values. Points mirrored into a wrong hemisphere usually mean latitude and longitude columns were swapped or signs were dropped; re-import and assign the columns explicitly. Points shifted by a consistent few hundred meters suggest a coordinate system mismatch; re-import declaring the correct one. **Gotcha:** Don't drag points to fix a systematic error; fix the import. Chapter 3 (Coordinate Systems) explains why declaring the wrong system shifts everything by the same sneaky offset.

23. Add a field to a hosted layer

Goal: A new column on a layer you own, ready to hold data. **Path:** The layer's item page, `Data` tab, switched to the fields view. **Steps:** Open the Data tab, switch from table view to fields view, and add a field: name it (short, no spaces; this is permanent), give it a friendly alias, and choose a type (text, integer, decimal, date). Then populate it by editing or calculating (Recipe 25). **Gotcha:** The field *name* cannot be changed later, only the alias can. Choose the type carefully too: numbers stored as text can't be summed or sized by. Chapter 12 (Schema Design) is the deep dive.

24. Fix date fields imported as text

Goal: A column of "3/14/2024"-style strings becomes a true date field you can filter and chart by time. **Path:** Item page `Data` tab: add a date field, then calculate it. **Steps:** Add a new field of type date (Recipe 23). Use the calculate option on the new field with an Arcade expression that parses the text field into a date. `Date($feature.DateText)` handles ISO-style text (2024-03-14) reliably; for other formats, split the text into year, month, and day parts with Arcade's text functions and build the date from those. Verify a sample, then stop using the text field. **Gotcha:** Ambiguous day/month order (is 3/4 March 4 or April 3?) parses silently wrong. Check rows where day and month differ enough to prove the order before you trust the whole column.

Watch out: Dates in ArcGIS are stored in universal time (UTC) and displayed in local time. If every date in your layer looks one day early or late, you are almost certainly seeing a timezone shift from import, not corrupted data. Chapter 39 (Troubleshooting) has the full diagnosis.

25. Calculate a field conditionally

Goal: Fill a field with different values depending on another field, for example a size class of Small, Medium, or Large. **Path:** Item page `Data` tab, or the table opened in Map Viewer; choose the field, then Calculate. **Steps:** Pick the target field and open the calculator with Arcade. Write a conditional such as `When($feature.POP > 100000, "Large", $feature.POP > 10000, "Medium", "Small")`. Preview on a feature, then run. **Gotcha:** Calculation rewrites every row that isn't filtered out, immediately, with no undo. Apply a filter first if you only mean to update some rows, and export a backup (Recipe 28) if the layer matters.

26. Give a field a dropdown list

Goal: Editors pick "Open / Closed / Pending" from a list instead of typing free text. **Path:** Item page `Data` tab, fields view; open the field and create a list of values. **Steps:** Select the field and add a list (this is a *domain*, a set of allowed values). Enter each value with a readable label. From then on, editing forms present a dropdown. **Gotcha:** A domain constrains future edits but doesn't clean existing data; run a calculation to standardize the mess already in the column. Chapter 12 covers domains and why they're the cheapest data-quality tool you have.

27. Append new records to an existing layer

Goal: Add this month's 200 new rows to a layer without touching the existing ones. **Path:** The layer's item page, `Update Data`, choosing the append (add features) option rather than overwrite. **Steps:** Prepare the new rows in the same schema as the layer. Choose the append option, upload the file, and confirm the field mapping the tool proposes, correcting any mismatched pairs. **Gotcha:** Mismapped fields fail quietly, leaving columns of nulls. Review the field mapping screen carefully instead of accepting it on autopilot, and count features before and after.

28. Export a backup copy of a hosted layer

Goal: A downloadable snapshot of your data, insurance against every destructive recipe in this chapter. **Path:** The layer's item page, `Export` (on or near the Overview tab). **Steps:** Choose an export format: file geodatabase for a full-fidelity backup, CSV for tables, shapefile only if some other software demands it. The export lands in your content as a downloadable item. **Gotcha:** Shapefiles truncate field names to ten characters and mangle long text; prefer file geodatabase for backups. Exports are snapshots, not syncs; they don't update when the layer does.

29. Join a spreadsheet to boundaries

Goal: A CSV of statistics keyed by county name or code becomes mappable county polygons. **Path:** `Analysis > Tools` in Map Viewer; look for the join features tool. **Steps:** Add both the boundary layer and the table to the map. Run the join tool, matching the polygon layer's identifier field to the table's identifier field, and save the result as a new layer. Style it (Recipe 12 or 13). **Gotcha:** Join by a code (like a standardized county code) whenever one exists. Names fail on punctuation, abbreviations, and spelling drift, and every failed match becomes a silent hole in your map. Chapter 16 covers joins alongside spatial overlay.

30. Delete test features without wrecking the layer

Goal: The junk points from testing disappear; the real data survives. **Path:** Map Viewer's table view and edit tools, with a filter as a safety rail. **Steps:** Filter the layer to isolate the junk (a test-user field, a date range, an obvious location). Open the table, verify by eye that everything matching is truly junk, select, and delete. Remove the filter. **Gotcha:** Deletion is permanent on a hosted layer. If you find yourself deleting more than a screenful, export a backup first, and consider enabling delete protection in the item's settings afterward so nobody deletes the *layer itself* by accident.

Analysis

31. Count points in polygons

Goal: Each neighborhood polygon gets a count (or sum, or average) of the points inside it. **Path:** `Analysis > Tools`; look for a summarize-within style tool. **Steps:** Choose the polygon layer as the area layer and the point layer as the layer to summarize. Ask for the count, plus any statistics of numeric fields (sum of sales, mean of inspection score). Run it and style the output polygons by the new count field. **Gotcha:** Points sitting

exactly on a shared boundary are edge cases: depending on the tool's spatial test they can be counted in both polygons or in neither, and points outside every polygon vanish from the totals. Sanity-check the summed counts against your point count. Chapter 16 is the full treatment.

32. Buffer around features

Goal: A zone of a fixed distance around points, lines, or polygons, for "what's within 500 meters" questions. **Path:** `Analysis > Tools`; the buffer creation tool. **Steps:** Pick the input layer, enter a distance and unit, and choose whether overlapping buffers should merge (dissolve) into one blob or stay separate per feature. Run it; the output is a polygon layer you can use in overlays. **Gotcha:** A straight-line buffer is not a travel distance. "Within 500 meters of a station" as the crow flies ignores rivers, fences, and street networks; for drive-time or walk-time zones you want service areas, covered in Chapter 18 (Network Analysis).

33. Find features inside (or near) other features

Goal: The subset of layer A that falls within layer B: parcels inside a flood zone, schools within a district. **Path:** `Analysis > Tools`; overlay, spatial-join, or find-by-location style tools. **Steps:** For a permanent answer, run an overlay tool intersecting layer A with layer B and save the result. For a lighter-weight answer, run a find-by-location style tool, which selects layer A features by their spatial relationship to layer B without computing new geometry. Style or count the result. **Gotcha:** "Within", "intersects", and "completely within" are genuinely different questions; a parcel half inside a flood zone passes intersects but fails completely-within. Decide which one your real question means before running anything.

34. Merge many small polygons into big ones

Goal: Counties become regions; dozens of small zones become five big ones. **Path:** `Analysis > Tools`; the dissolve-boundaries style tool. **Steps:** Add a field to the input that names each target region (Recipe 23 and 25), then run dissolve using that field. Adjacent polygons sharing a value merge into one. Ask for statistics if you want populations summed into the merged shapes. **Gotcha:** Polygons that share a value but don't touch can stay as separate pieces or merge into one multi-part feature (a single record whose shape has several disconnected parts) depending on the tool's setting. Multi-part features are legal but confuse counting; pick deliberately.

35. Check the credit cost before running an analysis

Goal: No surprise on the organization's credit bill. **Path:** Inside any analysis tool's pane, before you press run. **Steps:** Fill in the tool's inputs completely, then use the estimate-credits control that appears near the run button. Read the number, decide, and only then run. For repeated heavy work, note that the same operations in ArcGIS Pro against local data cost nothing. **Gotcha:** The estimate reflects the inputs as configured; enlarging the input layer or unchecking a filter afterward changes the real cost. Credits and who can spend them are Chapter 34's territory.

> **Tip:** Analysis cost scales with feature count. Filtering a layer to your study area *before* running a tool (Recipe 3) is the single easiest way to cut both credit use and runtime.

Sharing, collaboration, and handoff

36. Make a layer public safely

Goal: Anyone can see the map; nobody can edit, scrape everything, or see fields they shouldn't. **Path:** The layer's item page: `Settings` first, then the Share control. **Steps:** Before sharing, review the item settings: editing off (or share an editable layer via a read-only *view* instead, see Chapter 10), export disabled unless you mean it, delete protection on. Remove or hide sensitive fields. Then set sharing to Everyone. **Gotcha:** Public means public to scripts as well as people; the layer's underlying service endpoint (Chapter 32) can be queried by anyone. Never rely on pop-up configuration to hide sensitive data (Recipe 16); remove it from the shared item entirely.

> **Watch out:** Once something has been public, assume it has been copied. Un-sharing stops future access but does not recall data already downloaded. Share the minimum, and when in doubt, share a filtered view rather than the master layer.

37. Share with exactly three people

Goal: Three named colleagues see the map; nobody else does. **Path:** `Groups`, then the item's Share control. **Steps:** Create a group with membership set to invitation-only and visibility restricted. Invite the three accounts. Share the map *and every private layer in*

it to the group. Members now see it in their group content. **Gotcha:** Sharing the map but forgetting its layers is the classic failure: your colleagues open the map and see blank shapes or errors. Most sharing dialogs offer to update the layers' sharing to match; say yes. Chapter 34 covers groups, roles, and the sharing model end to end.

38. Let editors change only their own records

Goal: Field staff each edit what they collected, and can't touch (or even see) each other's work. **Path:** The layer's item page, `Settings` tab, editing section. **Steps:** Enable editing, then look for the ownership-based controls that restrict what editors can do to features they didn't create: typically options to edit only their own features and, if desired, see only their own. Pair with a view layer for managers who need to see everything. **Gotcha:** Ownership is tracked by editor fields the layer must be configured to record; data loaded before editor tracking was enabled has no owner and behaves unexpectedly. Chapter 13 (Editing Workflows) and Chapter 30 (Field Operations) build whole workflows on this switch.

39. Add a legend to an app

Goal: Viewers of your app can tell what the colors mean. **Path:** In Map Viewer, the `Legend` panel to preview it; in the app builder (Instant Apps, Experience Builder, Dashboards), the legend widget or setting. **Steps:** First fix the legend's raw material in the map: rename layers to plain English, since the legend displays layer names and style labels verbatim. Then, in your app's builder, enable the legend component or widget and place it. **Gotcha:** Legends are generated, not drawn: whatever your style says is what appears. A layer named "acs_pop_2023_v4_final" will say exactly that to the public. Chapters 26 through 29 cover each app builder's own legend quirks.

40. Hand off content when someone leaves

Goal: A departing teammate's maps, layers, and apps keep working under a new owner. **Path:** An administrator's view of `Organization > Members`, using the change-ownership controls; or each item page's owner setting. **Steps:** An administrator reassigns the member's items to a new owner, ideally into a designated folder. Verify the transferred layers still draw in their maps, and check for scheduled tasks, private groups, and credentials only the old account held. **Gotcha:** Transferring ownership does not transfer group memberships or app registrations. The platform generally refuses to delete a member who still owns content, and the delete flow will push you to reassign

or delete their items on the spot; don't let that last-second prompt become your handoff plan. Transfer deliberately first, verify second, delete never in a hurry. Chapter 34 covers offboarding as an administrative routine.

When a recipe fails

Recipes are terse by design, and terseness assumes a healthy platform underneath. When a step's button isn't where a recipe says, trust the goal over the path: Esri moves controls between releases, but the item page / Map Viewer split and the tabs on the item page have stayed stable for years. When a step runs but produces something wrong (empty joins, shifted points, blank layers for teammates), that is no longer a cookbook problem: turn to Chapter 39 (The Troubleshooting Encyclopedia), which diagnoses the failure patterns behind these same forty tasks, and to Chapter 40 for any term this chapter used faster than it defined.

The Troubleshooting Encyclopedia

Something is broken, and you want it fixed. This chapter is organized so you can find your symptom, work down a short list of causes ranked from most to least likely, run the checks in order, and apply the fix. You do not need to read it front to back. You do need to internalize one idea first, because every diagnosis in this chapter depends on it.

The layered anatomy of every ArcGIS problem

Almost everything you see in ArcGIS is a stack of at least four layers, each saved as a separate item with its own settings:

1. **The data** — the actual rows and geometries, usually stored in a hosted feature layer, a layer whose data ArcGIS stores and serves for you (see Chapter 10).
2. **The layer item** — the catalog entry that points at the data and carries sharing, editing, and metadata settings.
3. **The web map** — a saved recipe that says which layers to draw, how to style them, and what the pop-ups say (see Chapter 6).

4. **The app** — a Dashboard, StoryMap, Instant App, or Experience Builder page that displays the map and adds its own configuration on top (see Volume F).

When something looks wrong in an app, the defect can live in any of those four places, and the single most common troubleshooting mistake is fixing the wrong layer of the stack. So the universal first move for any symptom is: **isolate the layer**. Open the data table directly. If the data is right, open the web map in Map Viewer. If the map is right, the problem is in the app. Work bottom-up, and you will never waste an hour restyling a map when the real problem was a stale filter in a dashboard.

> **Tip:** Keep a browser bookmark to your content page (`Content` in the top navigation of your ArcGIS Online site). Nearly every fix in this chapter starts from an item page — the settings hub for a single layer, map, or app.

The second universal move: **test in a private or incognito browser window**. Browsers cache map tiles, layer definitions, and login sessions aggressively. A private window gives you a clean slate and instantly answers the question "is this broken, or is my browser remembering an old version?"

Display problems: the map draws wrong or not at all

Symptom: a layer simply does not appear

Likely causes, in order: the layer is turned off or filtered, you are zoomed outside its visible range, the layer failed to load, or the data is somewhere else on the planet.

Checks, in order:

1. Open the map in Map Viewer and look at the `Layers` panel. Is the layer's visibility toggle on? Layers grouped inside a group layer inherit the group's toggle, so check the parent too.
2. Look for a small warning message on the layer entry such as "not visible at this scale." Layers can have a **visible range** — a zoom window outside which they hide themselves. Zoom in or out, or adjust the range in the layer's properties.
3. Check for a **filter** on the layer. A filter is a saved condition like "STATUS = Open" that hides every feature that fails the test. If someone set a filter against a field

value that no longer exists in the data, the layer draws nothing and reports no error. Remove the filter and see if features return.

4. Select the layer and use the option to zoom to it. If the map flies to the middle of the ocean or another continent, you have a coordinate problem, covered next.
5. If the layer entry shows an error icon or "layer failed to load," the layer item may have been deleted, unshared, or moved. Open its item page from the layer's options menu. If you get a permissions error instead of an item page, jump to the sharing section of this chapter.

Symptom: features draw in the wrong place

The classic version: your points appear off the coast of West Africa, stacked at latitude zero, longitude zero. GIS people call this "Null Island" — it is where features land when their coordinates are blank or zero. The other classic: everything is offset by a consistent distance, or your data lands in the wrong country entirely.

Ranked causes:

WHAT YOU SEE	MOST LIKELY CAUSE
Everything at one ocean point near Africa	Blank or zeroed coordinate fields at import time
Data in the wrong hemisphere or mirrored	Latitude and longitude columns swapped during import
Data a consistent small distance off	Wrong datum or transformation (see Chapter 3)
Data wildly far away, absurdly tiny or huge	Coordinate system misdeclared — projected values read as degrees, or vice versa

Checks: open the layer's attribute table — its raw data grid — and read the coordinate values. Latitude must fall between -90 and 90; longitude between -180 and 180. If your "latitude" column holds values like -122, the columns are swapped — re-import and assign them correctly. If the values are six- or seven-digit numbers, they are projected coordinates (meters or feet in some regional system), not degrees, and you must tell

the import tool the correct coordinate system rather than letting it assume global latitude-longitude. Chapter 3 (Coordinate Systems) explains why, and Chapter 11 (Creating Data) covers the import screens where you declare it.

> **Watch out:** A small, consistent offset — features sitting one street over from where they belong — is almost never a data-entry error. It is a datum mismatch, where two coordinate systems disagree slightly about the shape of the Earth. Do not "fix" it by dragging features to look right against the basemap; declare the correct datum transformation instead, or you will corrupt good data.

Symptom: the whole map is blank

If nothing draws, not even the basemap, the problem is above the layer level: your network blocks the tile servers (common on corporate networks — ask IT about firewall rules for Esri domains), your session expired (reload and sign in again), or the browser cache is corrupt (try the private window test). If the basemap draws but every operational layer — the data layers you added, as opposed to the basemap — fails, and those layers live in ArcGIS Enterprise rather than ArcGIS Online, the Enterprise server may be down or unreachable from your network — see Chapter 35.

Upload and import failures

Symptom: "unable to add" or an import that dies partway

You tried to publish a file as a hosted layer and got an error, or the upload spun forever. Ranked causes: wrong packaging, wrong field contents, file too large, or a name collision.

Checks, in order:

1. **Packaging.** A shapefile — the venerable multi-file vector format — must be uploaded as a single zip file containing all of its sibling files (`.shp`, `.dbf`, `.shx`, `.prj`, and friends) at the top level of the zip, not nested inside a folder. A file geodatabase must likewise be zipped. Re-zip correctly and retry; this alone resolves a large share of upload failures.

2. **Field name hygiene.** Column names with spaces, punctuation, leading digits, or reserved words (like `date` or `order`) can break publishing. Rename offending columns to plain letters, digits, and underscores before uploading.
3. **CSV type inference.** When you upload a CSV — a plain text spreadsheet — ArcGIS guesses each column's type by sampling early rows. If the first rows of a column look numeric but later rows contain text, the import can fail or silently null out the text values. The fix is to review and override the field types on the import screen rather than accepting the guesses, or to clean the column so its contents are consistent. Chapter 12 (Schema Design) explains why types matter so much.
4. **Size.** Very large files can time out in a browser upload. Split the file, or better, publish from ArcGIS Pro, which handles large data far more gracefully (see Chapter 10).
5. **Name collision.** You cannot create two hosted layers with the same name in the same folder. If the error mentions the service name already existing, rename or delete the older item — but check what depends on it first. ArcGIS does not reliably list which maps and apps use a layer, so search your organization's maps and ask around before deleting anything shared.

Symptom: the upload succeeded but the table looks wrong

Dates imported as text, numbers with leading zeros truncated (ZIP codes are the eternal victim), accented characters mangled. All three are import-time decisions: date format recognition, type inference, and text encoding. The fix is always to fix the source or the import settings and re-import — editing thousands of damaged rows after the fact is misery. Save ZIP codes as text on purpose; a ZIP code is a label, not a quantity.

Geocoding surprises

Geocoding converts addresses into map points (Chapter 11 covers the workflow). Its failure modes are distinctive.

Symptom: points land in the wrong place, or all in one spot

Geocoders return the best match they can at whatever precision the input allows. A full street address pins a rooftop; a bare city name pins the city's center point. If a whole batch stacks on one point, your address column was probably mapped to the wrong

input field (for example, only the city column got used). Re-run and check the field mapping screen carefully. If addresses land in the wrong country, set the geocoder's country hint — without one, an ambiguous name like "Paris" can match a Paris in Texas instead of the one in France.

Symptom: some addresses fail to match at all

Ranked causes: typos and abbreviations the geocoder cannot resolve, addresses that are genuinely new (recent construction), and post office boxes (which are not physical places and cannot be geocoded to a rooftop). Review the unmatched list, correct obvious typos, and rematch. Most geocoding workflows include an interactive rematch step where you fix stragglers one at a time.

> **Watch out:** Batch geocoding consumes credits — the ArcGIS Online service currency (see Chapter 34) — in proportion to how many addresses you process. A test run of ten rows costs almost nothing; accidentally geocoding a hundred-thousand-row file, twice, because the first run had swapped columns, is a real bill. Always test with a small sample first.

Editing blockers

Symptom: you cannot edit a layer — the tools are missing or grayed out

Editing requires four separate switches to align, and any one of them being off produces the same symptom. Check in this order:

1. **Layer-level editing.** On the layer's item page, under settings, there is a master toggle enabling editing on the hosted feature layer, plus options controlling *what* editors may do (add only, update only, full control). If the layer was published read-only, nobody can edit it anywhere.
2. **View-level restrictions.** If you are working with a **hosted feature layer view** — a filtered window onto another layer (Chapter 10) — the view has its own editing settings that can be stricter than the source. Confirm which item your map actually uses; the item page states whether it is a view and names its source.
3. **Your privileges.** Your organization role must include editing privileges. If a colleague can edit the same layer and you cannot, this is the likely gap — ask your

administrator (Chapter 34).

4. **The map and app.** A web map can disable editing per layer even when the layer allows it, and apps often hide editing tools unless explicitly configured to show them. If you can edit in Map Viewer but not in the app, the app configuration is your culprit.

Symptom: edits appear to save, then vanish

Two usual suspects. First, you edited a **view** or a copy rather than the layer everyone else is looking at — check item IDs (the long code in an item page's web address), not just names, because organizations accumulate identically named layers. Second, someone else's sync or edit overwrote yours; see the sync section below. A third, sneakier cause: server-side validation — an attribute rule or similar logic attached to the layer — rejected the edit while the client displayed it optimistically. If edits vanish on refresh, check whether the layer has validation logic attached (Chapter 14).

Sharing and permission errors

Symptom: "You do not have permissions to access this resource or perform this operation"

This is the most common error message in all of ArcGIS Online, and it almost always means one of three things: the item is not shared with you, you are signed into the wrong account, or the item no longer exists. Checks:

1. Confirm which account you are signed in with (click your profile picture). People with a personal account and a work account trip on this constantly. Sign out, sign into the right one, retry.
2. Ask the item's owner how it is shared. Sharing has levels — owner only, specific groups, the whole organization, or everyone (public) — and you must be inside the circle.
3. If the owner insists it is shared and you still cannot see it, the link may point at a deleted or replaced item.

Symptom: the app works for you but not for anyone else

This is the **nested sharing problem**, and it deserves its own diagram in your head. An app contains a map, and the map contains layers. Each of those is a separate item with

separate sharing. Sharing the app publicly does *not* share the map or the layers. Your audience needs access to every item in the chain; you always have access to your own items, which is why it works for you and fails for them.

The fix: open the app's item page and use the sharing controls — when you share an item, ArcGIS typically prompts you to update the sharing of contained items you own so they match. It cannot do that for items owned by someone else; those owners must share their pieces themselves. Then verify like your audience: open the app's link in a private browser window while signed out. If it loads there, it loads for everyone it is shared with.

> **Tip:** The signed-out private-window test is the only sharing verification that counts. Never announce a public app until you have watched it load in a browser that has never heard of your account.

Symptom: sharing a layer publicly is blocked

Some organizations restrict who may share content publicly, and layers with editing enabled warn loudly before going public (a public editable layer means strangers can modify your data — usually a mistake). If the share control is disabled for you, this is an administrator setting, not a bug.

Performance degradation: everything is slow

Slowness is a symptom with many owners. Isolate first: is one layer slow, one map, one app, or everything?

One layer is slow. Almost always: too many features drawn at once, too much geometry detail, or a drawing style that forces the client to fetch every record. The remedies, in order of payoff: set visible ranges so detailed layers only draw when zoomed in; simplify or generalize very detailed polygons; reduce the fields the layer returns; and check whether the layer has grown enormous attachments. Chapter 10 covers layer performance tuning in depth, including when to rebuild a layer for speed.

One map is slow. Count the layers. Maps accumulate layers the way drawers accumulate cables, and every visible layer costs load time. Remove what the map does not need, group and toggle the rest, and be suspicious of any layer that draws

thousands of overlapping features at full extent when a clustered or aggregated view (Chapter 7) would communicate better anyway.

One app is slow. Dashboards with many data-driven elements each query the layers independently; a dashboard with a dozen charts against a huge layer issues a dozen heavy queries on every refresh. Reduce elements, add filters that narrow the data, and lengthen any auto-refresh interval.

Everything is slow. Your network, your machine, or a service outage. Check Esri's service health dashboard (search "ArcGIS Online health dashboard") before blaming your own work.

> **Tip:** Before guessing at a slow map, bisect it: turn every layer off, confirm the map is fast, then re-enable layers one at a time until the slowdown returns. The layer that brings the lag back names itself.

Sync conflicts and field app failures

Field Maps and its siblings (Chapter 30) work offline by downloading a **map area**, letting crews edit locally, and syncing changes back. The failure modes cluster.

Symptom: the map cannot be taken offline at all

Every layer in the map must be offline-capable, and the offending layer is usually a basemap or a reference layer nobody thought about. Checks: the layer item's settings must have **sync** enabled (a capability of hosted feature layers); the map's item page has an offline section that lists which layers block offline use, by name. Fix the listed layers or remove them from the field map.

Symptom: sync fails or hangs in the field

Ranked causes: poor connectivity mid-sync, very large pending edits (especially photo attachments), and a schema change made to the layer *after* the crew downloaded their offline area. That last one is the killer: if you add or delete fields, change a field type, or swap out a layer while offline copies exist, those copies may no longer be able to reconcile. The safe procedure is: have all field crews sync and remove their offline areas, make the schema change, then have them re-download. If a device is stuck with

unsyncable edits, do not delete the offline area in a panic — that discards the crew's work. There are documented recovery paths for retrieving unsynced edits from a device; exhaust those, and Esri support if needed, before deleting anything.

Symptom: two people edited the same feature and one edit disappeared

By default, the last sync to arrive wins. If crews genuinely overlap on the same features, that is a workflow problem more than a software problem: partition territory, or move to editing patterns where each crew adds new records rather than updating shared ones. Chapter 37 walks through designing a field operation so conflicts rarely happen in the first place.

> **Watch out:** Never change a layer's schema — fields, types, domains — while field devices hold offline copies against it. It is the single most destructive routine mistake in field operations, because the damage surfaces days later on someone else's device.

Analysis tool failures

Symptom: the tool errors out immediately

Ranked causes: an input layer the tool cannot use (wrong geometry type — a tool expecting polygons was handed points), an empty input (a filter upstream removed every feature), or missing privileges for the analysis service. Read the error message fully; geoprocessing errors — geoprocessing being Esri's word for running analysis tools — are terse but usually name the offending parameter. Chapter 22 explains how to read them in Pro, where messages are far more detailed than in the browser.

Symptom: the tool runs but returns empty or nonsense results

This one has a champion cause: the **extent setting**. Browser-based analysis tools often include an option like "use current map extent," which silently clips the analysis to whatever you happened to be zoomed to. Run an overlay while zoomed into one county and you will get results for one county, with no warning. Check the extent option before every run.

Other causes in rank order: coordinate system mismatch between inputs (features that never actually intersect because one layer is in the wrong place — revisit the display section above); unit confusion (a buffer of "5" in decimal degrees instead of miles is a very different circle); and joins that failed because key fields had mismatched types or stray whitespace (Chapter 16 and Chapter 12 respectively).

Symptom: the analysis worked but cost more credits than expected

Browser analysis tools charge credits scaled by the number of features processed. Most tools display an estimate before you confirm — read it. Running exploratory analysis on a full multi-hundred-thousand-record layer, repeatedly, while iterating on parameters is the classic burn. Iterate on a small filtered subset, then run the full job once. Heavy repeated analysis may belong in ArcGIS Pro instead, where most geoprocessing runs on your own machine and consumes no credits at all (see Chapter 22).

Credit surprises

Credits (Chapter 34) are consumed by a specific, knowable list of actions. When an organization's balance drops unexpectedly, the causes rank like this:

1. **Hosted feature layer storage** — the quiet one. Storage charges accrue continuously based on how much data you store, so a huge layer someone uploaded and forgot costs credits every day while everyone sleeps. Check the administrative credit report for storage line items and delete or archive dead weight.
2. **Batch geocoding** — the spiky one. One big job shows up as a cliff on the usage chart.
3. **Spatial analysis and GeoEnrichment** — enrichment (appending demographic data to your features) is among the most credit-intensive operations per record.
4. **Scheduled or automated tasks** — a notebook or automation someone set to run nightly, doing any of the above.

The administrator's credit dashboard breaks usage down by user and by capability, which turns "where did the credits go" from a mystery into a lookup. Viewing maps, drawing layers, and editing data cost essentially nothing — the meter runs on storage and on services that do computational work for you.

Login and license tangles

Symptom: you sign in successfully but your content is missing

You are in the wrong organization or the wrong account. This is overwhelmingly the answer. Check the account name and the organization name after signing in. If your organization uses corporate single sign-on, make sure you chose the organization's sign-in page rather than typing a username into the generic ArcGIS login — they can be different identities that both "work."

Symptom: ArcGIS Pro refuses to start, citing a license

Pro typically licenses through your online account (a **named user** license — one tied to your sign-in rather than your machine). Ranked causes: signed into an account that has no Pro license assigned (an administrator assigns licenses per user — Chapter 34); the license is checked out to another machine in offline mode and must be checked back in from that machine; or your organization uses a concurrent-license server that is unreachable from your network. If you work somewhere disconnected, learn Pro's offline license option *before* the field trip, not during it.

Symptom: a specific tool or extension is missing in Pro

Extensions (Network Analyst, Spatial Analyst, and the rest) are licensed separately from Pro itself. If a tool documented in Volume D or E is absent or disabled for you, check which extensions your account has been assigned, and whether the extension is enabled in Pro's licensing settings.

App, map, and layer disagree with each other

The final symptom family: the same data looks different depending on where you view it.

Symptom: you changed the map, but the app still shows the old version

Checks, in order: did you actually save the web map (unsaved Map Viewer changes exist only in your browser tab)? Is the app pointed at the map you edited, or at a copy — check the map's item ID in the app's configuration against the item you edited. Is your browser caching the old app — do the private-window test. Some app builders also take a snapshot of certain map properties at configuration time rather than reading them live, so after significant map changes it is worth reopening the app's configuration and re-saving.

Symptom: pop-ups or styling differ between the map and the app

Pop-up and style configuration normally lives in the web map (Chapters 7 and 9), and apps inherit it — but several app builders can *override* pop-ups, visible fields, or styling per layer within the app's own settings. When the two disagree, the app's override is winning. Decide where the truth should live (usually the map, so every app inherits consistently) and clear the override in the app.

Symptom: the attribute table shows data the map refuses to draw

A filter or an equivalent saved query condition is hiding features at the map level while the raw table shows everything. Also check whether the map has **time animation** enabled on the layer — a time-enabled layer with the time slider parked on one date draws only that date's features, which looks exactly like missing data (see Chapter 20).

When nothing in this chapter matches

A short escalation ritual for the genuinely weird:

1. Reproduce it in a private browser window, signed in, then signed out. Note the difference — it is diagnostic gold.
2. Reproduce it at each layer of the stack: data table, layer item, web map, app. Name the lowest layer where the symptom appears.
3. Check Esri's service health dashboard and your organization's own announcements.
4. Search the exact error message text in quotes; Esri's community forums and support knowledge base have seen nearly everything.
5. If you file a support ticket or forum post, include: the item type, what you expected, what happened instead, the exact error text, and the lowest layer of the stack where it reproduces. That single sentence — "the layer draws correctly in Map Viewer but not in the dashboard" — saves days of back-and-forth.

Most ArcGIS problems are configuration disagreements between layers of a stack, not bugs. Work bottom-up, change one thing at a time, and retest in a clean browser window after each change. The platform is large, but its failure modes are finite — and now you have met most of them.

Mega-Glossary and Resource Directory

Every field builds a private language, and GIS built two: the vocabulary of geography itself, and the vocabulary of Esri's software. When a colleague says "the hosted view inherits the renderer but you can override the definition query," they are speaking both dialects at once. This chapter is your decoder ring. The first half is a glossary of more than 150 terms in plain English, each defined the way a patient friend would define it, with pointers to the chapter that treats it in depth. The second half is a resource directory: where to find official documentation, free training, living communities of practice, and — maybe most valuable of all — how to phrase a search query so the internet actually helps you.

Use this chapter two ways. When a term stops you mid-sentence anywhere else in the Compendium, flip here for a fast answer, then follow the chapter reference if you need the full story. And when you finish the book, skim the glossary once end to end; the terms you can define without reading are yours now.

> **Tip:** Definitions here are deliberately short. Chapter references point to the real treatment — a glossary entry is a handshake, not a friendship.

The Glossary

A

Alias (field alias) — a friendly display name for a field, so `POP_EST_20` can appear as "Estimated Population" (Chapter 9).

Annotation — map text stored as fixed graphics with their own positions, rather than labels the software places dynamically (Chapter 23).

API (application programming interface) — the published rules by which one program talks to another. Every ArcGIS service exposes one (Chapter 32).

Arcade — Esri's lightweight expression language for calculating values on the fly in pop-ups, labels, and styling. It travels with the map, so it works everywhere the map is drawn (Chapter 8).

ArcGIS API for Python — a Python library for working with your web GIS: managing users and content, running analysis against services (Chapter 31). Not the same thing as arcpy.

ArcGIS Enterprise — the version of the platform you install and run on your own servers instead of Esri's cloud (Chapter 35).

ArcGIS Online — Esri's cloud-hosted platform: maps, layers, apps, and sharing, all in the browser, run on Esri's infrastructure (Chapter 2).

ArcGIS Pro — Esri's flagship desktop application for advanced mapping, editing, analysis, and print cartography (Chapter 21).

ArcMap — ArcGIS Pro's retired predecessor. You will still meet it constantly in older tutorials and forum answers; its workflows and menus do not match Pro.

arcpy — the Python package built into ArcGIS Pro that automates geoprocessing tools and desktop workflows (Chapter 31).

Area of interest (AOI) — the geographic extent you actually care about for an analysis, export, or offline map.

Aspect — the compass direction a slope faces, computed from an elevation model (Chapter 19).

Attachment — a file, usually a photo, stored with an individual feature — the inspection picture clipped to the hydrant record.

Attribute — a fact recorded about a feature: the name of the road, the year the parcel sold. Stored in fields.

Attribute table — the spreadsheet-like grid showing every feature as a row and every attribute as a column.

Authoritative — a badge an organization can put on an item marking it as the official, curated source (Chapter 5).

B

Band — one layer of measurements inside a raster. A color photo has red, green, and blue bands; satellite imagery often carries more (Chapter 15).

Basemap — the reference map that sits under your data: streets, imagery, terrain, or a muted canvas.

Blend mode — a styling control that changes how a layer's colors combine with what is beneath it, like multiply or overlay in a photo editor (Chapter 7).

Bookmark — a saved map extent you can jump back to with one click.

Bounding box — the smallest rectangle that fully contains a feature or dataset.

Breaks (class breaks) — the cut points that divide a numeric range into classes for styling: where "medium" ends and "high" begins (Chapter 7).

Buffer — a polygon containing everything within a chosen distance of a feature. The workhorse of "what's near this?" analysis (Chapter 16).

C

Cardinality — how many records on each side of a relationship: one-to-one, one-to-many, many-to-many (Chapter 12).

Cartography — the craft of designing maps people can actually read (Chapter 4).

Cell — one square in a raster's grid, holding a single value. Also called a pixel (Chapter 1).

Choropleth — a map that shades areas by a value. Honest ones show rates, not raw counts (Chapters 4 and 7).

Classification — grouping numeric values into a small number of classes for display: natural breaks, quantile, equal interval, and friends (Chapter 7).

Clip — the cookie-cutter operation: keep only the data that falls inside a boundary (Chapter 16).

Clustering — drawing many nearby points as one aggregate symbol when zoomed out, so dense data stays legible (Chapter 7).

Coded value domain — a pick list of allowed values for a field, storing a compact code but showing a readable description (Chapter 12).

Contour — a line connecting points of equal value, classically elevation.

Coordinate — a pair (or triple) of numbers that pins down a location.

Coordinate system — the framework that gives coordinates meaning: which model of the Earth, which units, which flattening scheme (Chapter 3).

Credits — the usage currency of ArcGIS Online, consumed by storage and certain premium operations like geocoding and routing (Chapter 34).

CSV (comma-separated values) — a plain-text table format; one of the most common ways data enters ArcGIS (Chapter 11).

D

Dashboard — an app that pairs a map with charts, gauges, counters, and lists for at-a-glance monitoring (Chapter 28).

Data model — the way a GIS represents the world: vector for discrete things, raster for continuous surfaces (Chapter 1).

Datum — the mathematical model of the Earth's shape that anchors a coordinate system. Mixing datums shifts your data (Chapter 3).

Definition query — an expression that hides every feature not matching a condition. The data is untouched; the layer just shows a subset.

DEM (digital elevation model) — a raster in which every cell holds a ground height (Chapter 19).

Digitizing — creating features by tracing them over imagery or a scanned map by hand (Chapter 11).

Dissolve — merging adjacent features that share an attribute value into single larger features (Chapter 16).

Domain — a rule attached to a field restricting what values it will accept, either a pick list or a numeric range (Chapter 12).

Dot density — a style that scatters dots inside polygons, one dot per so-many things counted (Chapter 7).

E

Editing — changing feature geometry or attributes, in the browser or in Pro (Chapter 13).

Elevation surface — the 3D ground reference that features drape onto in a scene.

Enterprise geodatabase — a multi-user geodatabase living inside a database server such as PostgreSQL or SQL Server (Chapters 12 and 35).

Esri — the company behind ArcGIS; the name began as Environmental Systems Research Institute, founded in 1969.

Experience Builder — Esri's flexible drag-and-drop builder for custom web apps, from single pages to multi-page sites (Chapter 29).

Export — pulling data out of the platform into a file you can hand to someone else.

Extent — the geographic rectangle a map currently shows, or that a dataset covers.

F

Feature — one real-world thing represented as a point, line, or polygon plus its attributes. One hydrant, one road, one parcel.

Feature class — a collection of features sharing one geometry type and one schema, stored in a geodatabase (Chapter 12).

Feature layer — features served over the web so maps can draw, query, and (if allowed) edit them (Chapter 10).

Feature service — the web endpoint behind a feature layer; the machinery a browser actually talks to (Chapter 32).

Feature template — a preset bundle of default attributes and drawing tool used when creating new features during editing (Chapter 13).

Field — one column in a table; one kind of fact recorded for every feature.

Field Maps — Esri's mobile app for viewing maps and collecting or editing data in the field (Chapter 30).

File geodatabase — Esri's folder-based local database format, the `.gdb` you see on disk (Chapter 12).

Filter — Map Viewer's name for a definition query: show only features matching an expression (Chapter 6).

G

Geocoding — converting addresses or place names into map coordinates (Chapter 11).

Geodatabase — Esri's container format for feature classes, tables, relationships, and data rules (Chapter 12).

Geodesic — measured along the Earth's curved surface rather than across the flat map; the honest way to compute long distances (Chapter 3).

GeoJSON — an open, human-readable text format for vector data, beloved by web developers.

Geometry — the shape half of a feature: its coordinates and how they connect.

Geoprocessing — running tools that take data in and produce transformed data out; the verb form of analysis (Chapter 22).

GIS (geographic information system) — software, data, and methods for capturing, managing, analyzing, and displaying anything tied to a location.

GNSS / GPS — satellite positioning. GNSS is the umbrella term; GPS is the American constellation under it (Chapter 30).

Graduated symbols — styling where bigger values get bigger symbols (Chapter 7).

Group — a shared workspace in an organization: a set of members and the items they collaborate on (Chapter 34).

Group layer — layers nested under one heading in the layer list so they can be toggled together.

H

Heat map — a smooth color surface rendered from point density; hot where points crowd (Chapter 7).

Hexbin — aggregating points into a honeycomb of hexagonal cells, each colored by count or statistic (Chapter 17).

Hillshade — a shaded-relief rendering that lights terrain from an imaginary sun so ridges and valleys pop (Chapter 19).

Hosted feature layer — a feature layer whose data physically lives in ArcGIS Online rather than on your own server (Chapter 10).

Hub (ArcGIS Hub) — Esri's product for public-facing engagement and open data sites (Chapter 2).

I

Imagery — pictures of the Earth from satellites, aircraft, or drones, handled as rasters (Chapter 15).

Instant Apps — Esri's gallery of no-code, configurable app templates: pick one, point it at a map, fill in options (Chapter 26).

Interpolation — estimating values between sample points to build a continuous surface, such as rainfall between gauges (Chapter 19).

Intersect — the overlay that keeps only where all inputs overlap, combining their attributes (Chapter 16).

Item — anything stored in your content: a map, a layer, an app, a file. Everything in ArcGIS Online is an item.

Item page — the details-and-settings page every item has, where sharing, descriptions, and configuration live.

J–K

Join — attaching columns from one table to another through a shared key field. When location is the key instead, see spatial join (Chapter 16).

JSON — the plain-text data format web services use to exchange information; the wire language of web GIS (Chapter 32).

Kernel density — a statistical surface that spreads each point's influence over a neighborhood, producing smoother density maps than raw counts (Chapter 17).

KML / KMZ — the geographic file format popularized by Google Earth, now an OGC standard; ArcGIS can read and export it.

L

Label — dynamic text the software draws next to features, generated from attributes and placed automatically (Chapter 9).

Layer — one dataset drawn on a map, carrying its own style, pop-ups, and settings.

Layout — the print-oriented page composition in Pro: map frames, legends, scale bars, titles (Chapter 24).

LiDAR — laser scanning from aircraft or drones that yields dense 3D point clouds of terrain and structures (Chapters 15 and 19).

Line (polyline) — the geometry type for linear things: roads, rivers, pipes.

Living Atlas — Esri's curated collection of ready-to-use layers, maps, and apps covering the world (Chapter 5).

Locator — the engine and reference data behind geocoding; what actually turns "742 Evergreen Terrace" into a point.

M

Map image layer — a layer served as pictures drawn by the server, rather than as raw features; common with ArcGIS Enterprise (Chapters 32 and 35).

Map Viewer — ArcGIS Online's browser-based environment for assembling, styling, and saving maps (Chapter 6). Its predecessor lives on as Map Viewer Classic in older documentation.

Member — a named account inside an organization (Chapter 34).

Metadata — data about data: who made it, when, how, from what source, and who to ask (Chapter 5).

ModelBuilder — Pro's visual canvas for chaining geoprocessing tools into repeatable diagrams (Chapter 25).

Mosaic dataset — a catalog that manages many rasters and serves them as one seamless image (Chapter 15).

Multipart feature — one feature made of several disconnected shapes: Hawaii stored as a single state record.

N

Named user — Esri's licensing identity: one person, one login, with a user type that governs capabilities (Chapter 34).

Network dataset — streets (or rails, or pipes) modeled with connectivity, costs, and rules so routing works (Chapter 18).

NoData — raster cells holding no measurement at all. Crucially not the same as zero (Chapter 15).

Normalization — dividing a count by area or population so places of different sizes compare fairly (Chapter 7).

Null — an empty attribute value meaning "unknown." Distinct from zero and from an empty string, and it bites people constantly.

O

Offline map — a defined map area packaged onto a device so field work continues without a signal (Chapter 30).

OGC (Open Geospatial Consortium) — the standards body behind vendor-neutral service formats like WMS and WFS (Chapter 32).

Organization (org) — your subscription's shared space: its members, content, groups, and settings (Chapter 34).

Orthophoto — an aerial image geometrically corrected so it measures true like a map (Chapter 15).

Overlay — the family of operations that combine layers so their geometry and attributes merge: intersect, union, clip, and kin (Chapter 16).

P

Point — the geometry type for single locations.

Polygon — the closed-shape geometry type for areas: parcels, lakes, districts.

Pop-up — the information window that opens when you click a feature (Chapter 9).

Portal — the general term for a web GIS's front door; also the specific component of ArcGIS Enterprise that plays the ArcGIS Online role on-premises (Chapter 35).

Projection — the mathematics of flattening the curved Earth onto a plane. Every projection distorts something; the craft is choosing what (Chapter 3).

Publishing — turning data you have into a live web layer others can use (Chapter 10).

Python — the scripting language of ArcGIS automation, via arcpy and the ArcGIS API for Python (Chapter 31).

Q

Quantile — a classification method placing an equal number of features in each class (Chapter 7).

Query — a question asked of data in SQL-like syntax: `STATUS = 'Open' AND PRIORITY > 2`.

QuickCapture — Esri's big-button mobile app for rapid, one-tap field capture at speed (Chapter 30).

R

Raster — the grid-of-cells data model, ideal for imagery and continuous surfaces (Chapter 1).

Relationship class — a formal, managed link between tables or feature classes in a geodatabase (Chapter 12).

Renderer — the rule set that decides how features draw: which symbol, which color, driven by which attribute (Chapter 7).

Resolution — the ground size of one raster cell. Smaller cells, finer detail, bigger files.

REST — the web convention ArcGIS services speak; every layer has a REST endpoint you can inspect in a browser (Chapter 32).

Route — the best path between stops through a network, by time or distance (Chapter 18).

S

Scale — the ratio of map distance to ground distance. Counterintuitively, "large scale" means zoomed in (Chapter 4).

Scene — a 3D map, viewed in perspective with terrain and optionally extruded or modeled features.

Schema — the design of a dataset's tables: which fields, which types, which rules (Chapter 12).

Service area — everything reachable within a set time or distance along a network from a point — a drive-time polygon (Chapter 18).

Shapefile — the venerable multi-file vector format from the 1990s. Universally supported, but saddled with short field names and other dated limits (Chapter 11).

Slope — steepness computed from an elevation model, in degrees or percent (Chapter 19).

Smart mapping — Map Viewer's data-aware styling assistant, which reads your data and suggests sensible defaults (Chapter 7).

Snapping — cursor magnetism while editing, pulling your clicks onto existing vertices and edges so geometry connects cleanly (Chapter 13).

Spatial join — attaching attributes from one layer to another based on location rather than a shared key (Chapter 16).

Spatial reference — the coordinate system stamped on a dataset; what makes its numbers locatable (Chapter 3).

SQL (Structured Query Language) — the database query language whose WHERE-clause style powers filters and definition queries throughout ArcGIS.

StoryMaps — Esri's app for narrative: scrolling text, media, and maps woven into an article-like experience (Chapter 27).

Survey123 — Esri's form-centric field data collection app: smart forms first, map second (Chapter 30).

Symbology — the visual clothing of a layer: colors, sizes, shapes, and the logic assigning them (Chapters 7 and 23).

T

Table — rows and columns. A feature class is a table where each row also carries a shape.

Tile layer — a layer served as pre-drawn image tiles for speed; fast to display, not queryable like features (Chapter 10).

Time-enabled layer — a layer with date fields configured so maps can filter and animate it through time (Chapter 20).

TIN (triangulated irregular network) — a terrain surface built from triangles between elevation points; the vector cousin of a DEM (Chapter 19).

Topology — rules about how features share space: parcels must not overlap, roads must connect at endpoints (Chapter 14).

Transformation (datum transformation) — the extra mathematical step that converts coordinates between datums correctly (Chapter 3).

U

Union — the overlay that keeps everything from all inputs, splitting geometry wherever boundaries cross (Chapter 16).

Unique values — styling that gives each category its own symbol: one color per land-use type (Chapter 7).

UTM (Universal Transverse Mercator) — a family of projected coordinate systems dividing the world into narrow zones, each accurate within its own strip (Chapter 3).

V

Vector — the points-lines-polygons data model for discrete things (Chapter 1).

Vector tile layer — tiles that carry drawable geometry instead of pictures, so the client can restyle them and text stays crisp (Chapter 10).

Versioning — the mechanism letting many editors work on an enterprise geodatabase simultaneously in isolated versions, reconciled later (Chapters 13 and 35).

Vertex — one coordinate along a line or polygon boundary; the plural is vertices.

View (feature layer view) — a separate, filterable, re-permissionable window onto a hosted feature layer, sharing the same underlying data (Chapter 10).

Viewshed — the area visible from a point, given the terrain in between (Chapter 19).

W

Web map — the saved recipe — layers, styles, pop-ups, extent — that Map Viewer authors and every app opens. It references layers rather than containing their data (Chapter 6).

Web scene — the 3D counterpart of a web map.

WGS 84 — the global datum and coordinate system GPS uses; the default assumption for latitude-longitude data (Chapter 3).

Widget — a functional building block you drop into an app in Experience Builder and similar builders: a legend, a search box, a chart (Chapter 29).

WKID (well-known ID) — the numeric code that names a coordinate system unambiguously; 4326 means WGS 84 (Chapter 3).

WMS / WFS — OGC standards for serving maps and features across vendor boundaries (Chapter 32).

X–Z

XY data — a table of coordinate columns you convert into a point layer (Chapter 11).

Z-value — the third coordinate: elevation or height attached to a vertex.

Zonal statistics — summarizing raster values within zones: average elevation per watershed, total rainfall per county (Chapter 19).

Zoom level — the fixed ladder of scales web maps and tile layers use; each level doubles the detail of the one before.

> **Watch out:** Esri renames things. Map Viewer was "Map Viewer Beta" and its predecessor is now "Map Viewer Classic"; the community forums were "GeoNet"; app builders have merged and split. When a term in an older article does not match what you see, suspect a rename before you suspect yourself.

The Resource Directory

No single book — not even a forty-chapter one — stays current with a platform that updates several times a year. What follows is where to go when this book's answer is not enough, ordered roughly from official to communal.

Official Esri Documentation

Esri's documentation is genuinely good, which is not something you can say about every large software vendor. The trick is knowing which door to enter.

The ArcGIS Online product documentation (reachable from `doc.arcgis.com` or by searching "ArcGIS Online help") covers everything browser-based: Map Viewer, hosted layers, sharing, administration. It is organized by task — "create maps," "manage data," "administer" — and it is updated with each platform release.

The ArcGIS Pro help is a separate, larger corpus. Its crown jewel is the tool reference: every geoprocessing tool has a page describing parameters, behavior, environment settings, and usage notes. When a tool misbehaves, read its page before anything else — the answer is usually in the usage notes everyone skips.

The ArcGIS Developers site (`developers.arcgis.com`) houses the REST API reference, the documentation for Esri's SDKs (software development kits — the code libraries developers use to build custom apps), and the complete Arcade function reference. Even if you never write code, the Arcade reference is where you will end up once Chapter 8 hooks you.

The ArcGIS Blog, on Esri's main site, is where product teams announce releases and publish deep-dive tutorials. Release announcements are worth skimming quarterly; they are how you learn that the feature you wished for last year quietly shipped.

Learn ArcGIS

Learn ArcGIS (`learn.arcgis.com`) is Esri's free library of guided, scenario-based lessons: map a hurricane's path, site a new store, analyze urban heat. Each lesson includes the data, step-by-step instructions, and an estimated completion time, and you can filter the catalog by product and topic. This is the single best answer to "how do I get hands-on practice?" — the lessons are real workflows with real data, not toy exercises.

> **Tip:** Many Learn ArcGIS lessons run fine on a free trial or a personal-use account, and each lesson states its requirements up front. Check the requirements line before you start rather than discovering a licensing wall at step thirty.

Esri Community

Esri Community (`community.esri.com`, formerly GeoNet) is the official forum: question-and-answer spaces organized by product, plus blogs from Esri staff and an idea exchange where feature requests are logged and occasionally answered. Esri employees genuinely participate, and the search archive reaches back many years — which is both its power and its hazard.

Watch out: Always check the date and product on a forum answer before following it. A highly upvoted 2014 answer about ArcMap, or a 2019 answer about Map Viewer Classic, can send you hunting for menus that no longer exist. Sort by recent when the topic touches anything with a user interface.

The Wider Community

Outside Esri's walls, a few venues have earned their reputations. GIS Stack Exchange (`gis.stackexchange.com`) is the rigor-first question-and-answer site; excellent for precise technical questions, especially anything involving code, projections, or open-source tools alongside ArcGIS. The r/gis community on Reddit is looser and better for career questions, workflow opinions, and "is it just me?" sanity checks. Community-run Slack and Discord workspaces — The Spatial Community is the long-running example — offer real-time conversation with practitioners.

For ongoing learning rather than emergency help: John Nelson's cartography posts on the ArcGIS Blog are a masterclass in making beautiful maps with the tools you already have; Kenneth Field writes and speaks prolifically on cartographic judgment; and the MapScaping podcast interviews practitioners across the whole geospatial industry, Esri and beyond. A handful of practitioner-run email newsletters round up geospatial news and job postings; they change hands and names often enough that the reliable way to find the current good ones is to ask in the communities above. None of these require any particular skill level — they are how you absorb the culture of the field.

Training Paths and Certification

Esri Academy, Esri's training site, is the formal catalog: free web courses, paid instructor-led classes, and learning plans that sequence courses toward a role like analyst or administrator. Your organization's license often includes access to a substantial slice of the paid catalog — ask your administrator before paying out of pocket.

Esri also periodically runs free MOOCs — massive open online courses — on subjects like cartography and spatial analysis. They run as cohorts a few times a year, take a few hours a week for several weeks, and award a certificate of completion. The cartography

MOOC in particular has a devoted following and pairs beautifully with Chapters 4 and 23.

Esri Technical Certification is the formal credential track: proctored (supervised) exams organized by product and role, at escalating levels from foundational to professional. Conceptually, certifications matter most in two settings — Esri partner firms, where staff certifications affect the firm's standing, and government hiring, where they check a box procurement understands. For most other employers, a portfolio of actual maps and apps speaks louder than a certificate. If you pursue one, work from the current exam's published skills list rather than third-party guesswork.

> **Watch out:** Certification exams are versioned against specific software releases. Confirm which release the current exam targets before you study, or you may drill on an interface the exam no longer uses.

How to Search for GIS Help Effectively

Most GIS frustration is not a missing answer; it is a search query that cannot find the answer that exists. A few habits fix this.

Name the product, precisely. "ArcGIS" alone matches thirty years of software. Say `ArcGIS Pro`, `ArcGIS Online Map Viewer`, `ArcGIS Enterprise`, `Field Maps`. If the problem involves the older generation, say `ArcMap` or `Map Viewer Classic` explicitly — and if it does not, adding your product name filters out a decade of stale results.

Name the version when it matters. Desktop behavior changes between Pro releases. Find yours under `Project > About` in Pro and include it when behavior seems to contradict the documentation.

Quote the error verbatim. Paste the exact error text, in quotation marks, with the product name. Error numbers and phrasing are the most searchable strings Esri produces; paraphrasing them throws away your best evidence.

Use the field's vocabulary. Search "definition query" rather than "hide some rows," "spatial join" rather than "combine layers by location." This glossary exists partly so your searches speak the same language the documentation does.

Distinguish the two Pythons. arcpy questions and ArcGIS API for Python questions look similar and have disjoint answers. Name the library.

When you ask rather than search, give the trinity: what you did (product, version, data type, steps), what you expected, and what happened instead — plus the exact error if there is one. Questions shaped this way get answered in hours; "my map is broken, please help" gets silence.

> **Tip:** Before asking anywhere, spend five minutes on the layer's REST endpoint or the tool's documentation page (Chapters 32 and 22 show you how to read each). A striking fraction of "bugs" are documented behavior, and finding that yourself is faster than any forum.

Where the Compendium Ends

You now hold the vocabulary in this glossary, the map of resources beyond this book, and — from the thirty-nine chapters before this one — the working knowledge to use both. The platform will keep changing under you; that is not a threat but the terms of the craft. When a menu moves, you know how to find where it went. When a workflow breaks, Chapter 39 taught you how to reason about why. And when you meet a term nobody defined for you, you know exactly the kind of plain-English sentence to demand until it makes sense. That habit, more than any single definition here, is what turns someone who has built one map into someone who can build whatever the next problem needs.
