Airtable Extensions user interface guidelines

Introduction

Airtable provides a toolkit of components that users can assemble to create their own software. By composing tables, views, fields, and records, users can create the ideal structure, logic, and interfaces for their workflows. You can extend this kit by building blocks to let users visualize, collaborate, and analyze their information in new ways.

Extensions are installed inside Airtable bases, and each extension can be installed multiple times. Each extension installation has its own settings, perhaps for different collaborators or distinct workflows. Each extension can read and write data in all tables in the base it’s installed in, but typically extensions rely on a single source table and view. This allows users to specify which slice of their data they want to display or update. Each extension defines its own UI based on what it does — for the most part, these fall into one of a few common Archetypes. Extension installations are displayed in a dashboard, which can either be viewed alongside a table in a sidebar or viewed in fullscreen mode.

The goal of these guidelines is to convey the core design principles for extensions, along with specific practical UI and UX recommendations you can implement for better consistency and cohesion with the Airtable platform. By following these patterns, your extension will feel more familiar to Airtable users. They will be able to learn how to use your extension more quickly and effectively.

Principles

Extensions follow three core product principles — they should be composable, flexible, and collaborative. Use these themes to make design decisions and inform the ideal behavior of your extension.

COMPOSABLE

Extensions are modular extensions that compose with the rest of the base and with one another. Whenever possible, an extension should reuse configuration defined elsewhere in the base, such as view settings or field options. This allows users to define something once and use it in multiple places — this reduces the effort needed to set up an extension and makes it easier to maintain and change the configuration in the future. Generally, extensions should treat the base as the source of truth for application data. By fetching data from and saving data back to the base, each extension installation can automatically stay in sync with one another and operations can “flow” between them. For example, a user could modify a table with the batch update extension and instantly see those changes reflected in a summary extension pointing to the same table. You don’t need to build charting into your extension if users can install the existing chart extension alongside your extension.

FLEXIBLE

Airtable is used for a wide range of different use cases across industries. Even within a single use case, bases can look totally different depending on the specific team or workflow. As a result, extensions should be flexible and integrate into a variety of bases. An extension should not make too many assumptions about the schema of the base it’s installed in. The more requirements an extension has, the harder it is for an end user to set up. However, this doesn’t mean that an extension shouldn’t make any assumptions — often, extensions rely on the existence of certain fields for their core functionality. For example, the timeline and Gantt extensions both require start and end date fields. Extensions should have customization options to support different use cases and user preferences. These options can be unique to the extension or inherited from a source view (e.g. record coloring).

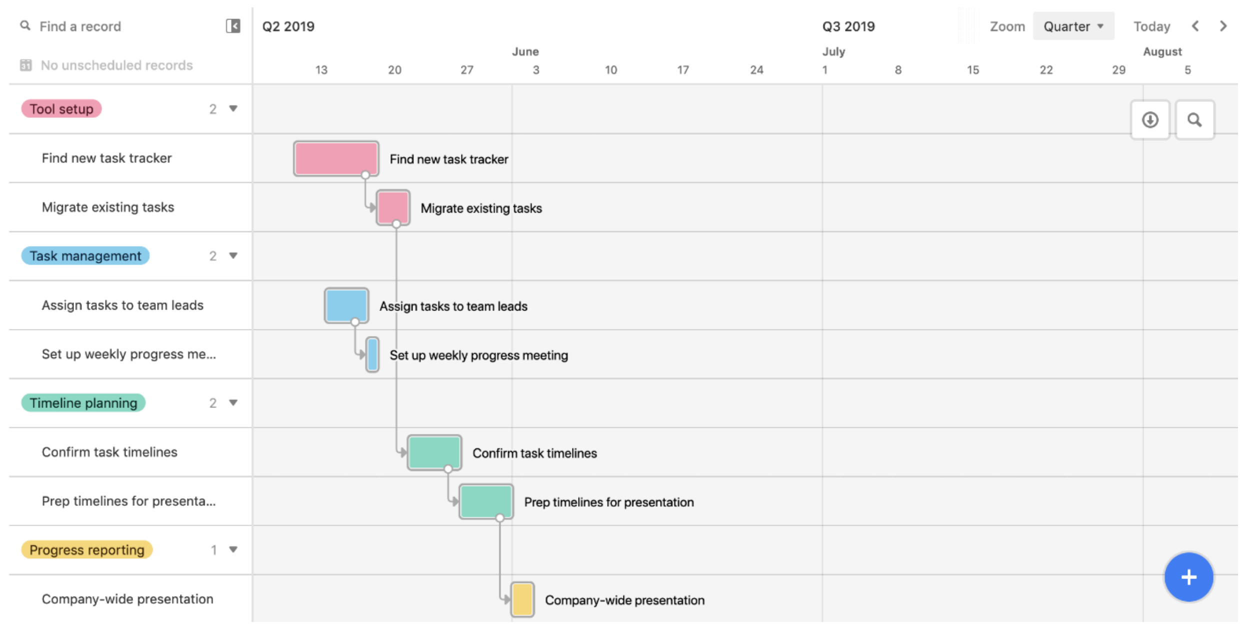
COLLABORATIVE

Extensions are used in a real-time, collaborative environment where it's possible for multiple users to access or modify some of the same data simultaneously. Generally speaking, an extension should use the most up-to-date information from the base it's installed in, so that users always see current, accurate data and have a more responsive multi-user experience. In certain scenarios, an extension might need to break out of this constraint and "freeze" data in local state. This is usually the case for actions or procedures that might take an extended period of time, such as mapping fields for an import extension or editing code in the scripting extension. Occasionally, the table, view, or fields that an extension installation relies on might get reconfigured or deleted entirely. Other times, the user of an extension might not have the necessary permissions to perform a particular action. Extensions should be robust enough to handle these states gracefully.

Archetypes

The extensions that have been released so far mostly fall into one of the following archetypes, each with its own architectural patterns. If the extension you're building fits into one of these categories, we recommend that you look at the relevant examples for reference. Not every extension maps neatly to one of the archetypes and we anticipate new archetypes emerging as the Extensions platform grows.

VISUALIZATION



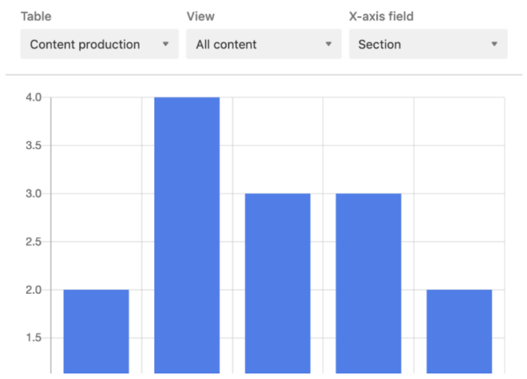
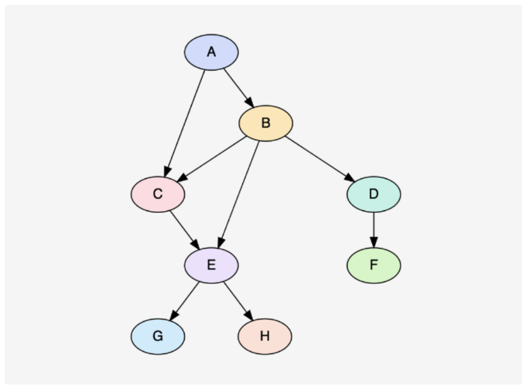


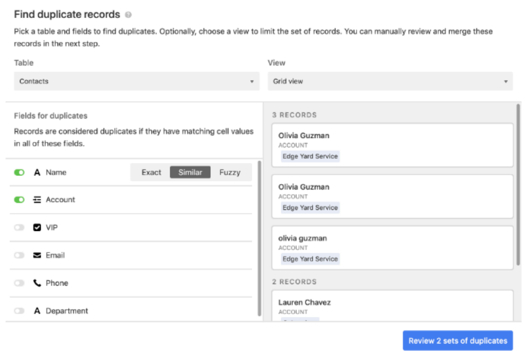
Table: Content production
View: All content
X-axis field: Section



Examples: Gantt, simple chart, flowchart

Visualization extensions take lists of records and display them in a particular UI. This can be a static visualization like a bar chart or an interactive visualization like a Gantt chart. Typically, these extensions consist of a settings sidebar, where the user can pick the source table, view, and fields and customize the visualization itself. These extensions should always display up-to-date data from the base and should offer an easy way to navigate to expanded records for full details and convenient editing. For improved ergonomics, the most important types of edits for that visualization can be implemented in the extension as a first-class feature, like drag and drop for changing categories in matrix extension or rescheduling in Gantt extension.

WIZARD



Find duplicate records

Pick a table and fields to find duplicates. Optionally, choose a view to limit the set of records. You can manually review and merge these records in the next step.

Table: Contacts

View: Grid view

Fields for duplicates

Records are considered duplicates if they have matching cell values in all of these fields.

● Name	Exact [✓ Similar] Fuzzy
● Account	
○ VIP	
○ Email	
○ Phone	
○ Department	

3 RECORDS

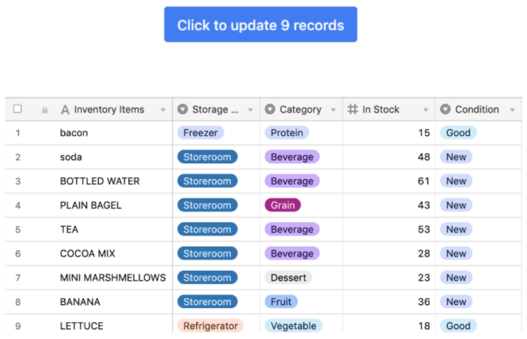
Olivia Guzman
ACCOUNT
Edge Yard Service

Olivia Guzman
ACCOUNT
Edge Yard Service

olivia guzman
ACCOUNT
Edge Yard Service

2 RECORDS

Lauren Chavez
ACCOUNT



	Inventory Items	Storage ...	Category	In Stock	Condition
1	bacon	Freezer	Protein	15	Good
2	soda	Storeroom	Beverage	48	New
3	BOTTLED WATER	Storeroom	Beverage	61	New
4	PLAIN BAGEL	Storeroom	Grain	43	New
5	TEA	Storeroom	Beverage	53	New
6	COCOA MIX	Storeroom	Beverage	28	New
7	MINI MARSHMELLOWS	Storeroom	Dessert	23	New
8	BANANA	Storeroom	Fruit	36	New
9	LETTUCE	Refrigerator	Vegetable	18	Good

Examples: Dedupe, update records

Wizard extensions perform actions or guide users through structured flows. These tend to either be "push button" interfaces or linear procedures. Usually, these extensions are set up once and are not regularly reconfigured — it's more likely that these extensions will be installed multiple times for different flows. For these extensions, descriptive headings and instructional help text should be included for each step of the process. Moreover, it's important to include some sort of confirmation step or the ability to revert updates if there's a possibility for destructive changes to the base.

IMPORT/SYNC/ENRICHMENT

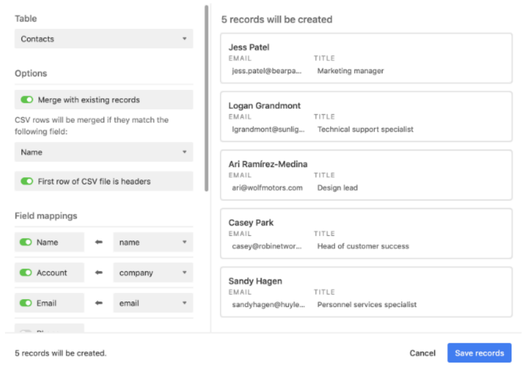


Table: Contacts

Options

● Merge with existing records

CSV rows will be merged if they match the following field:

Name

● First row of CSV file is headers

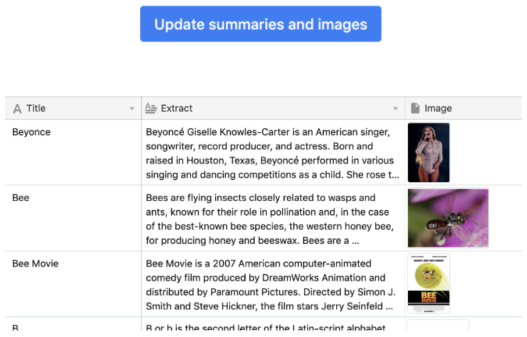
Field mappings

● Name	⬅	name
● Account	⬅	company
● Email	⬅	email

5 records will be created

Name	EMAIL	TITLE
Jess Patel	jess.patel@bearpa...	Marketing manager
Logan Grandmont	lgrandmont@sunlig...	Technical support specialist
Ari Ramírez-Medina	ari@wolfmotors.com	Design lead
Casey Park	casey@robinetwor...	Head of customer success
Sandy Hagen	sandyhagen@huyle...	Personnel services specialist

5 records will be created.

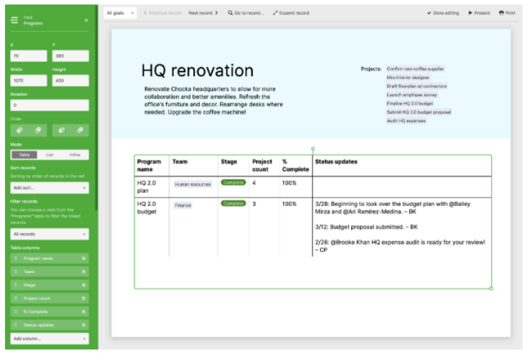


Title	Extract	Image
Beyonce	Beyoncé Giselle Knowles-Carter is an American singer, songwriter, record producer, and actress. Born and raised in Houston, Texas, Beyoncé performed in various singing and dancing competitions as a child. She rose t...	
Bee	Bees are flying insects closely related to wasps and ants, known for their role in pollination and, in the case of the best-known bee species, the western honey bee, for producing honey and beeswax. Bees are a ...	
Bee Movie	Bee Movie is a 2007 American computer-animated comedy film produced by DreamWorks Animation and distributed by Paramount Pictures. Directed by Simon J. Smith and Steve Hickner, the film stars Jerry Seinfeld ...	
B	B or b is the second letter of the Latin-script alphabet	

Examples: CSV import, wikipedia enrichment

Import, sync, and enrichment extensions are a particular type of wizard extensions that take data from an external source (either a file or a third-party API) and save them to an Airtable base. In the file import case, the initial screen of the extension will usually be a file upload/dropper UI. In the third-party API case, the extension will typically feature more detailed onboarding to guide the user through the process of creating an API key for the service and adding it to the extension. Both kinds usually involve a field mapping step, where the user defines how the source data should be saved in Airtable.

EDITOR ENVIRONMENT



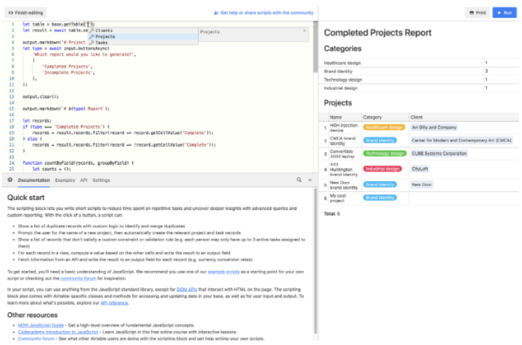
70 380 1070 400 0

HQ renovation

Renovate Chocka headquarters to allow for more collaboration and better amenities. Refresh the office's furniture and decor. Rearrange desks where needed. Upgrade the coffee machine!

Projects: Confirm new coffee supplier; Hire interior designer; Draft floorplan w/ contractors; Launch employee survey; Finalize HQ 2.0 budget; Submit HQ 2.0 budget proposal; Audit HQ expenses

Program name	Team	Stage	Project count	% Complete	Status updates
HQ 2.0 plan	Human resources	Complete	4	100%	
HQ 2.0 budget	Finance	Complete	3	100%	3/28: Beginning to look over the budget plan with @Bailey Mirza and @Ari Ramírez-Medina. – BK 3/12: Budget proposal submitted. – BK 2/26: @Brooke Khan HQ expense audit is ready for your review! – CP



```
let table = base.getTable('');
let result = await table.se

output.markdown('# Project
let type = await input.buttonsAsync(
    'Which report would you like to generate?',
    [
        'Completed Projects',
        'Incomplete Projects',
    ],
);

output.clear();

output.markdown(`# ${type} Report`);

let records;
if (type === 'Completed Projects') {
    records = result.records.filter(record => record.getCellValue('Complete'));
} else {
    records = result.records.filter(record => !record.getCellValue('Complete'));
}

function countByField(records, groupByField) {
    let counts = {};
```

Quick start

The scripting block lets you write short scripts to reduce time spent on repetitive tasks and uncover deeper insights with advanced queries and custom reporting. With the click of a button, a script can:

- Show a list of duplicate records with custom logic to identify and merge duplicates
- Prompt the user for the name of a new project, then automatically create the relevant project and task records
- Show a list of records that don't satisfy a custom constraint or validation rule (e.g. each person may only have up to 3 active tasks assigned to them)
- For each record in a view, compute a value based on the other cells and write the result to an output field
- Fetch information from an API and write the result to an output field for each record (e.g. currency conversion rates)

To get started, you'll need a basic understanding of JavaScript. We recommend you use one of our example scripts as a starting point for your own script or checking out the community forum for inspiration.

In your script, you can use anything from the JavaScript standard library, except for DOM APIs that interact with HTML on the page. The scripting block also comes with Airtable-specific classes and methods for accessing and updating data in your base, as well as for user input and output. To learn more about what's possible, explore our API reference.

Other resources

- MDN JavaScript Guide - Get a high-level overview of fundamental JavaScript concepts.
- Codecademy Introduction to JavaScript - Learn JavaScript in this free online course with interactive lessons.
- Community forum - See what other Airtable users are doing with the scripting block and get help writing your own scripts.

Completed Projects Report

Categories

Healthcare design	1
Brand identity	3
Technology design	1
Industrial design	1

Projects

	Name	Category	Client
1	HGH injection device	Healthcare design	Avi Gilly and Company
2	CMCA brand identity	Brand identity	Center for Modern and Contemporary Art (CMCA)
3	Convertible 3000 laptop	Technology design	CUBE Systems Corporation
4	443 Huntington brand identity	Industrial design	CityLoft
5	New Door brand identity	Brand identity	New Door
6	My cool project	Brand identity	

Total: 6

Examples: Page designer, scripting

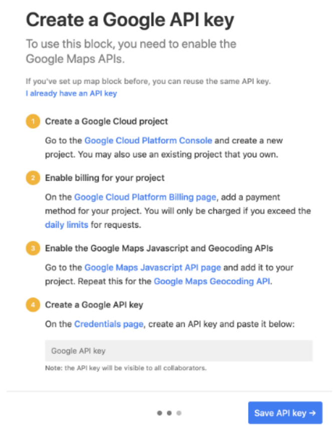
Editor environment extensions have distinct "edit" and "view" modes. Typically the extension will be set up and edited by a smaller set of creators, while other collaborators mostly consume or use the end product. These extensions are unique in that much of their content is stored in global config, even though extensions usually save data back to a table. For example, the page designer extension saves the layout information and the scripting extension saves the script code in global config, since this data does not directly describe records in the base. This content is specific to a particular extension installation, so it does not need to be shared with other extension installations in the base.

Patterns

For more information, also refer to the Polishing your extension guide.

ARCHITECTURE

Onboarding



Create a Google API key

To use this block, you need to enable the Google Maps APIs.

If you've set up map block before, you can reuse the same API key.
I already have an API key

1 Create a Google Cloud project

Go to the Google Cloud Platform Console and create a new project. You may also use an existing project that you own.

2 Enable billing for your project

On the Google Cloud Platform Billing page, add a payment method for your project. You will only be charged if you exceed the daily limits for requests.

3 Enable the Google Maps Javascript and Geocoding APIs

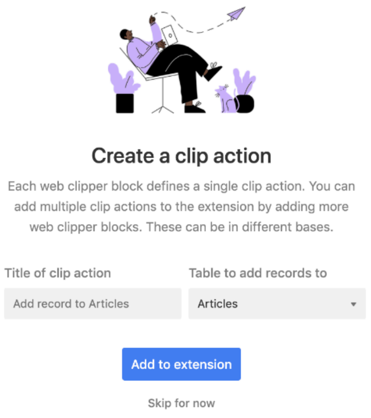
Go to the Google Maps Javascript API page and add it to your project. Repeat this for the Google Maps Geocoding API.

4 Create a Google API key

On the Credentials page, create an API key and paste it below:

Google API key

Note: the API key will be visible to all collaborators.



Create a clip action

Each web clipper block defines a single clip action. You can add multiple clip actions to the extension by adding more web clipper blocks. These can be in different bases.

Title of clip action	Table to add records to
Add record to Articles	Articles

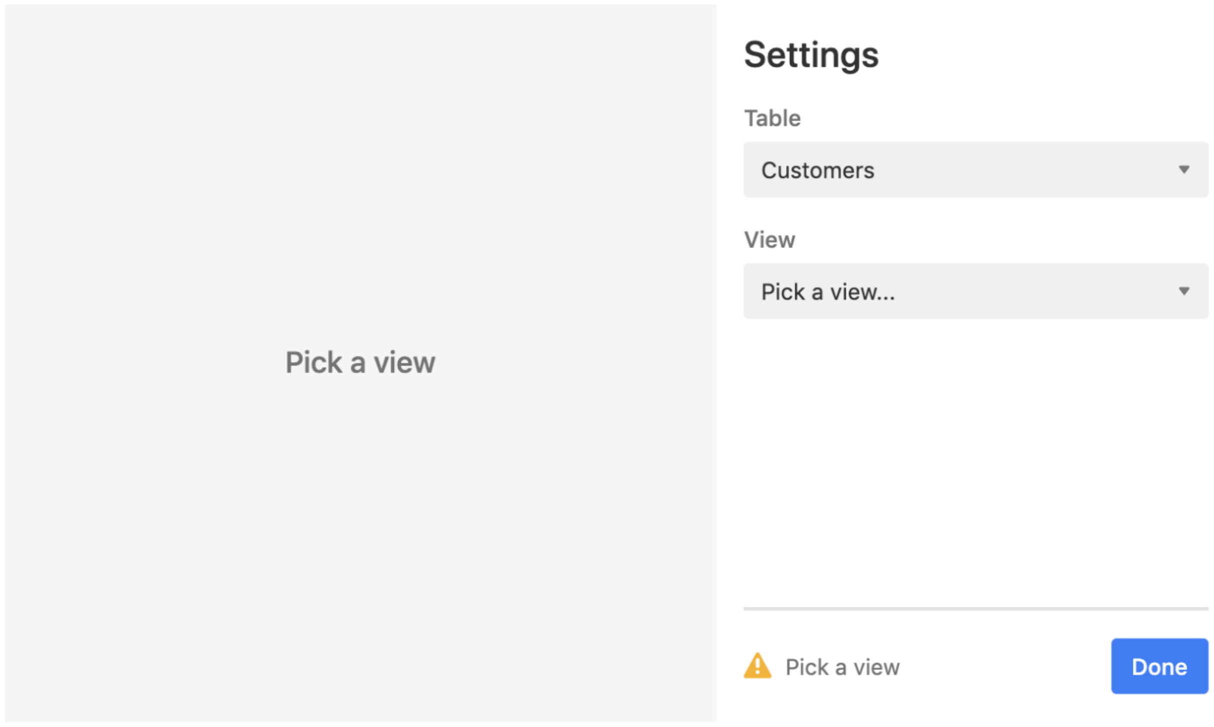
Determine if your extension requires a dedicated onboarding flow for first-time users. If your extension requires involved setup or user input (typically Wizard extensions), consider adding an introductory sequence with those instructions. For simpler extensions, you can start directly on the settings screen or rely on clear, descriptive help text to assist the user.

Settings

The first time your extension is run, pre-populate the extension settings with sensible defaults and “best guesses”. For example, the timeline extension automatically uses the active table and view for source data, the first date field for the start date, and the first select field for record coloring. This optimizes for having something actually show up when a user initially installs the extension.

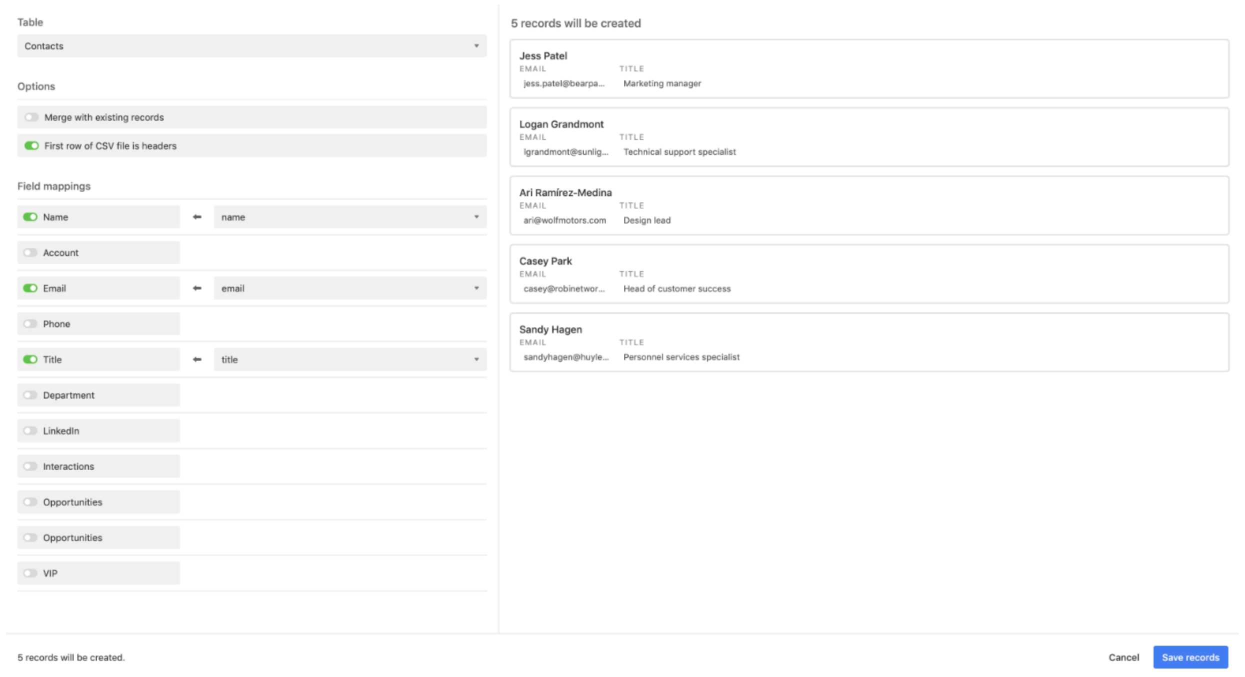
In most cases, the settings for your extension should live within a dedicated settings screen or sidebar. These settings should be ordered in terms of importance — picking a source table, view, and fields should be at the top, followed by optional fields or other customization options. For Visualization extensions, use a settings sidebar and reflect changes in real-time, so that the user can fine-tune options as they go without having to switch back and forth between screens.

If there is an invalid setting or a required setting is missing, your extension should call this out to the user with an actionable validation error. One common UI pattern is to add a persistent bottom bar to your settings screen or sidebar:



You can also call out these configuration issues on the main screen of the extension, either by using a bottom bar or an empty state.

In some Wizard extensions, certain settings like field mappings might actually change with every session — these can be lifted out of the settings screen and put on the main screen. As an example, the CSV import extension doesn’t have a separate settings screen that the user toggles on/off. Instead, each time a user uploads a file, they can change the settings as part of the import flow.



Table

Contacts

Options

Merge with existing records

First row of CSV file is headers

Field mappings

Name	⬅	name
Account		
Email	⬅	email
Phone		
Title	⬅	title
Department		
LinkedIn		
Interactions		
Opportunities		
Opportunities		
VIP		

5 records will be created

Jess Patel

EMAIL	TITLE
jess.patel@bearpa...	Marketing manager

Logan Grandmont

EMAIL	TITLE
lgrandmont@sunlig...	Technical support specialist

Ari Ramírez-Medina

EMAIL	TITLE
ari@wolfmotors.com	Design lead

Casey Park

EMAIL	TITLE
casey@robinetwor...	Head of customer success

Sandy Hagen

EMAIL	TITLE
sandyhagen@huyle...	Personnel services specialist

5 records will be created.

Infrequently changed configuration settings like API keys should still be kept on a separate settings screen.

COMMUNICATION

All screens should have a heading, unless it's the main screen/canvas of a Visualization extension. Optionally, you can add help text underneath the heading for more detail. For all UI copy, use sentence case rather than title case (e.g. "Choose a date field" rather than "Choose a Date Field").

On every screen, there should be at most one primary action button (or danger button, if the primary action is dangerous). Buttons that only have an icon without a text label should be used sparingly — they are best used in a toolbar of buttons of the same variant. Buttons should be labeled with simple verbs (e.g. "Update" or "Publish") that describe the action that will be performed. One notable exception to this rule is the "Done" button.

For actions that a user cannot perform, it is better to disable controls rather than conditionally hide them. For example, if your extension requires editor permissions to run, you could disable the primary action button for read-only collaborators and wrap it in a tooltip that explains the necessary permissions.

When referring to a table, view, field, or record from the base, use the model's name (or primary cell value, in the case of records) followed by the model type. You can wrap the model name in quotation marks or make the text strong to distinguish it from the rest of the UI copy. For example:

- **Good:**
  Are you sure you want to delete the "Table" record?
  Are you sure you want to delete the **Table** record?
- **Bad:**
  Are you sure you want to delete this record?
  Are you sure you want to delete Table?

In some cases, you may need to truncate long strings to one line — you can do this by adding ellipses at the end of the displayed string.

DATA PERSISTENCE

In general, your extension should use the base as the source of truth and write back to the base when making updates. Besides the base, there are two other options for data persistence: global config and local storage.

Global config is a key-value store for each extension installation and should be used for storing extension-specific settings like the IDs of the source table and view. For example, the todo list extension stores the IDs of a tasks table, a tasks view, and a “Done” field in global config. This data is shared between all collaborators and most extensions automatically update when a value in global config changes.

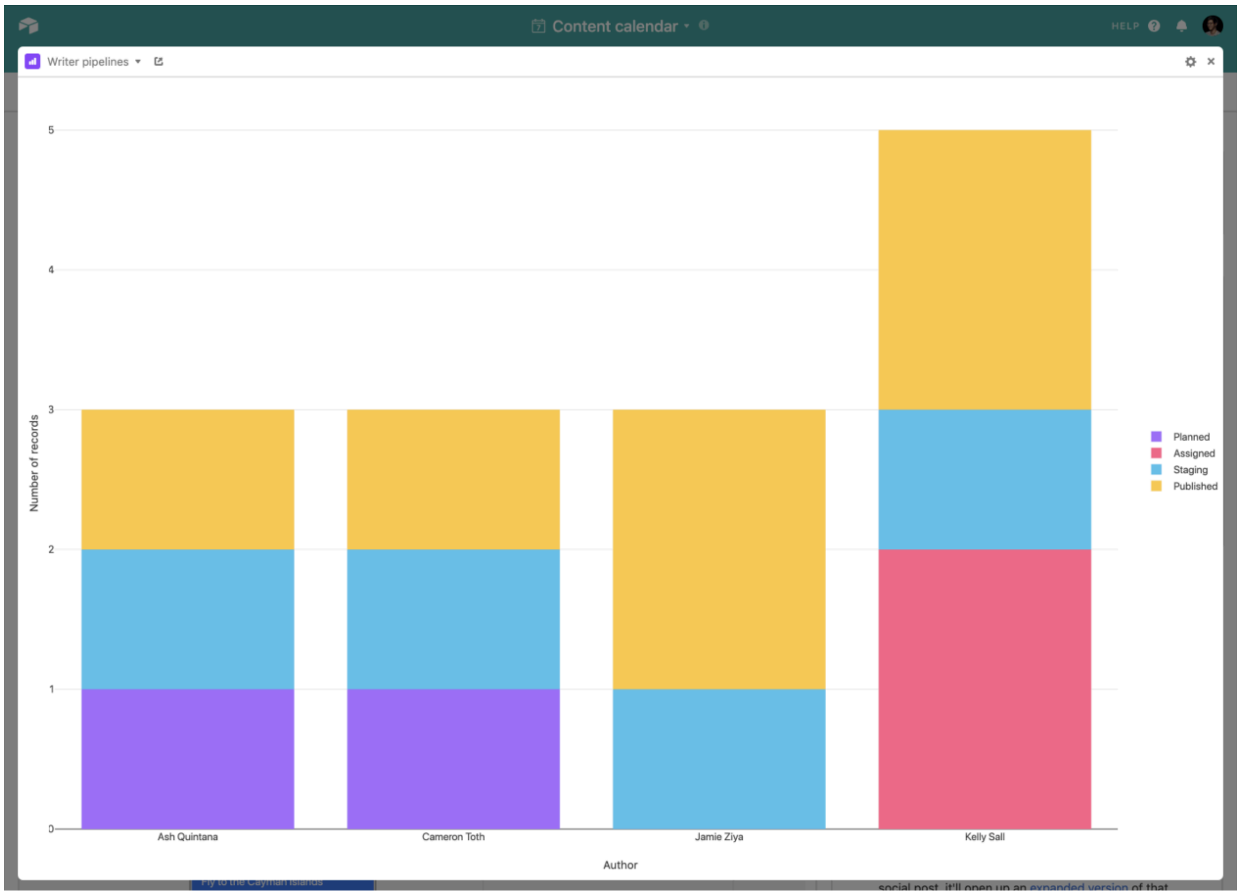
Occasionally, extensions will not only store its configuration in global config, but also its content. As an example, the description extension stores markdown in global config. It doesn’t really make sense to save this back to a table, since it isn’t an attribute of particular record or set of records in the base. One caveat is that global config can only store up to 100kB of data.

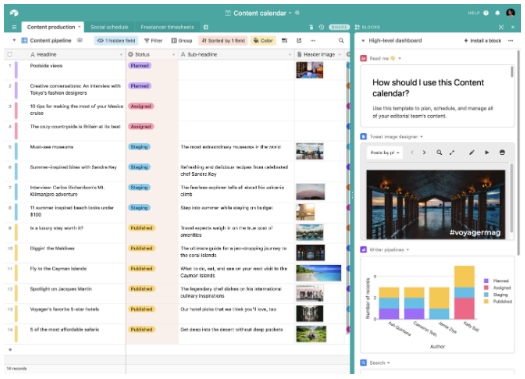
In cases where the source table may change (e.g. field mappings), you may want to consider storing the configuration per table, so that switching tables doesn’t clear out previous settings. You don’t need to clean up global config if models get deleted — this way, the configuration is retained if the user undoes the deletion or restores the deleted item from trash.

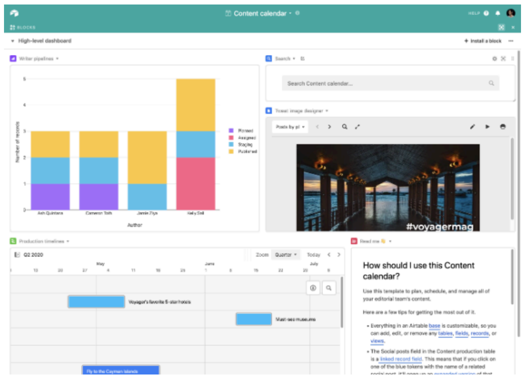
Local storage is for per-user (per-browser, to be precise) configuration and should be used sparingly. This is most useful for more transient UI state, like zoom level or whether a certain panel is collapsed or expanded.

VIEWPORTS

By default, an extension installation runs in fullscreen mode the first time it runs. In future sessions, the extension installation can be run in one of three viewports: fullscreen, in the sidebar dashboard, or in a fullscreen dashboard.







Within each of these viewports, the extension frame can be resized. Your extension can gracefully handle different viewport sizes by conditionally showing different UI. You can use responsive web design techniques like stacking your layout vertically or changing the settings sidebar to a side sheet or full settings screen. You could also consider displaying a summary or single primary action button when the extension is small.

As a last resort, the extension can define a minimum viewport size, which means that installations of the extension will be disabled when they’re smaller than a certain size. For some interactions, you may want to force the extension installation to enter fullscreen mode — for example, when displaying UI that requires significant screen real estate, like the scripting extension code editor or the dedupe extension merge interface.

Your extension can also define a maximum viewport size when it’s in fullscreen mode. This can be useful for screens in your extension that don’t have much content or screens where you want finer-grained control over the sizing of the UI, like during the introductory sequence of your onboarding flow.

FAQ

WHAT IS THE DIFFERENCE BETWEEN AN EXTENSION AND A SCRIPT?

With the launch of the scripting extension, we introduced a new component to the Airtable toolkit. Scripts are small programs that users can run by clicking a button. They are ideal for reducing time spent on repetitive tasks or creating guided user flows. By contrast, every extension is a full React app. Building an extension requires knowledge of React and a local development environment. Scripting works in the browser and only requires basic familiarity with JavaScript. As a result, the extensions SDK has a much wider surface area than the scripting extension APIs.

Wizard extensions like dedupe or batch update are similar to scripts: they perform actions when a user clicks a button or progresses through a sequence of steps. Unlike scripts, these extensions often involve rich interactive UIs, such as the merge interface in the dedupe extension or the action configuration in the batch update extension. Moreover, these extensions have user-configured settings that are persisted between sessions, whereas scripts have to either hardcode those settings or ask for them each time the script is run.

When deciding whether to build an extension or a script, consider the following:

- Do you need custom UIs beyond basic form inputs?
- Do you need real-time interaction between multiple users?
- Do you need to persist settings between sessions?
- Do you need to display information or visuals without user interaction?

If you answered “yes” to any of these questions, it’s likely that you should be building an extension instead of a script.