AppSheet Reference Manual

Contents

AppSheet Reference Manual..... 3

Basic Concepts....3
- General Knowledge....3
- Get Started -- 0 to 60!....6

App Definition: Data....16
- Data concepts....16
- Effective spreadsheet design....24
- Data definition in the app editor....34
- Expressions....41
- Controlling Data Inputs (Column Constraints)....52
- Tutorial Content....55

App Definition: User Experience....65
- Presenting information....65
- Capturing information....73
- Working with Media Types....79
- Branding and styling the app....82
- Tutorial Content....90

App Definition: Advanced Features....95
- Rich data definition....95
- Controlling system behaviors....99
- Working with your data....102
- Integrating with other services....106
- Tutorial Content....108

Security and Reliability....113
- Data Security....113
- Reliability....115
- Access Control....117
- Tutorial Content....120

Managing Apps....125
- Preparing for Deployment....125
- Deploying and Sharing Your Apps....129
- Once You Have Users....140
- Application Lifecycle....141

Troubleshooting....144
- Getting and giving help....144
- General....145
- Mobile device issues....148
- Issues with specific providers....149

Frequent App Design Questions....151
- Common App Usage Scenarios....151

AppSheet Reference Manual

Basic Concepts

General Knowledge

Who should use AppSheet and what kind of apps can it create? – AppSheet

AppSheet allows anyone to create elegant, powerful, and useful applications for business, professional, or personal needs.

If you are computer savvy and use online tools to manage your life and work, AppSheet is meant for you, and hundreds of millions of others like you.

It generally takes complex programming to build useful applications. However, AppSheet enables a limited but important subset of apps-- those that follow a simple data-driven pattern. The behavior of an AppSheet app is almost completely determined by the structure of your data. That's why you can build an app without writing any code.

You can build apps for project management, field operations, customer service, team collaboration, task management, staffing, and so much more.

AppSheet apps are designed to work offline-- you can control and enable various aspects of online behavior.

Check out some of our samples.

Related articles

- *Extracting structure from cloud spreadsheets*
- *AppSheet is NOT...*
- *Create an app*
- *How should I use multiple sheets in my app?*
- *Traditional vs. instant app deployment*

Extracting structure from cloud spreadsheets – AppSheet

Most interesting apps show and capture data. Traditionally, apps built by software developers use proprietary mechanisms to store data in structured databases. AppSheet avoids this approach for two reasons: (1) **our app creators are not software developers**, (2) **we want our app creators to be in full control of their data**.

Instead AppSheet apps use data from spreadsheets. Spreadsheets allow almost anyone to organize and store data. In later sections, we'll describe how you can also access data from other sources, but for now, let's focus on spreadsheets.

AppSheet cannot access data from a spreadsheet on your PC, but it can access data from a spreadsheet stored in a cloud file system like Google Drive or Dropbox. To explain it really simply, AppSheet makes your cloud-hosted spreadsheet behave like a simple database. The power of using a cloud-hosted spreadsheet is that there are no copies of your data created, nor is the data 'uploaded' into AppSheet servers. Instead, AppSheet reads and writes directly to your spreadsheet.

AppSheet automatically infers the structure of your data from the spreadsheet you provide. AppSheet models a data set as a 'table'-- i.e. a collection of rows with uniformly structured columns. Each spreadsheet is treated as a single 'table'.

For each column, AppSheet infers a data type, such as text, number, date, etc. There are more than 20 data types recognized. AppSheet also automatically groups related columns to form composite columns-- eg: [FirstName and LastName], or [Street, City, State]. These inferences happen automatically but of course, they may not always make the right decision, so you can always override and fine tune these choices.

Related articles

- *Mobile apps 'driven' by the structure of data*
- *Make a spreadsheet*
- *AppSheet is NOT...*
- *Who should use AppSheet and what kind of apps can it create?*
- *Mobile-specific features in my app*

Mobile apps 'driven' by the structure of data – AppSheet

A mobile app created by AppSheet is completely driven by the structure of the data.

- The app can show various pre-defined views of the data. As the app creator, you *specify and customize the views shown*.
- The individual data values are *presented based on their type*. For example, an address is automatically geocoded and shown on a map. A phone number can be dialed or texted when selected.
- If the app permits data to be added or updated, it automatically generates an *input form based on the structure of the data*. Based on the column type, the input mechanism is automatically chosen. For example, an image column automatically launches a camera.

So in short, the app creator defines the structure of data and configures the presentation of the data, but all of the detailed behaviors of the app are automatically driven by the data.

Related articles

- *Traditional vs. instant app deployment*
- *AppSheet is NOT...*
- *Mobile-specific features in my app*
- *Make a spreadsheet*
- *Extracting structure from cloud spreadsheets*

Traditional vs. instant app deployment – AppSheet

We do NOT follow the traditional app deployment model because it is too cumbersome.

The traditional 'heavyweight' app deployment model is to create and submit each app separately to an app store (iTunes Store for iOS and Google Play Store for Android). And then your users find and download the app from the app store. This is mechanically complex-- a software developer needs to package code together and submit it. It is procedurally complex-- there are forms to fill out and an approval process. It is time consuming, especially as this is the process every time you make a small change to your app.

These processes make sense if you are building a unique app for consumers or a broad audience. In the future, we may offer a 'whitelabel' service for this scenario, but it does not make sense if you are building an app for your team or employees. That's why we have created an instant app deployment mechanism for AppSheet. Deploying an AppSheet app is as simple as clicking on a link in an email and following a couple of instructions.

AppSheet's deployment model

AppSheet runs on iOS and Android (4.2 and above) devices.

Instant app deployment is possible because all AppSheet apps (the apps you build) are "hosted" by the AppSheet mobile app.

The essential idea behind AppSheet is that your apps are driven by data and configuration settings. The only unique aspect of each app is its data and its configuration. And everything else that is _common_ across apps is collected together into a single 'hosting' environment called the AppSheet mobile app.

The AppSheet app already available for download in each mobile appstore. When you deploy and run your app, it will appear to run on its own, but is actually 'hosted' by the AppSheet app. While this is not a perfectly accurate analogy, it may be useful to think of your app hosted in the AppSheet app in just the same way that web pages are hosted in a web browser.

You can distribute your app *by sending your users an install link by email*. When they click on the link from a mobile device, they are asked to install AppSheet on their device (AppSheet is already available in the app stores) and then to install the app icon on their homescreen. Clicking on the icon launches the app.

Your app is hosted and rendered on the mobile device using AppSheet as a 'runtime', just the same way different web pages are hosted in a browser.

Beyond the obvious simplicity of this approach, there are a few other benefits:

- Your apps are instantly available-- there is no delay between app creation and app deployment.
- Changes to your apps are instantly available too.
- Any system-wide changes (new features we provide, issues we fix, performance improvements) automatically appear in all apps.

In some cases, you might really want to build a traditional app like a software developer. White-label is a possibility for AppSheet apps, we are currently working on a solution for stand-alone apps that go in the store but the option is currently unavailable. While this choice is not common, you might consider it if:

- The discovery and distribution model of the appstore is important in your use case.
- You are aiming for large scale consumer adoption of your app.

When you choose to submit an app to the iTunes or Google Play app stores, you need to ensure that it conforms to the policies of those app stores, and the approval process that goes along with it.

We will update this entry with more details about White-Label in the future.

Related articles

- *Mobile-specific features in my app*
- *AppSheet is NOT...*
- *Who should use AppSheet and what kind of apps can it create?*
- *Advanced app customizations*
- *Create an app*

Mobile-specific features in my app – AppSheet

You should not expect to see a spreadsheet in your app. Your spreadsheet is only a data source. You are creating an app optimized for a mobile device in four ways: data presentation, data capture, offline use, and branding.

1. Mobile devices have limited screen sizes, so each data row is presented using mobile-optimized patterns.

- Most data views are summary views that expand to show the full details when an entry is clicked/tapped.
- In summary views, we always show the key value and as many other columns as possible.
- Actionable values (that launch actions when you tap them) like phone numbers and emails are preferentially shown.
- Columns on the left of the spreadsheet are considered more important than columns on the right.
- Mobile-appropriate input controls are provided for different column data types.
- - If there is an Image column, it is populated by taking a photo with the camera.
  - Signatures can be captured on the touch screen. *See how the signature feature works in our Delivery Tracking sample app.*
  - Enumerated types (defined by data validation lists in the spreadsheet) get a dropdown list to choose from.
- Each app can be branded in a variety of ways, including with a logo. When you distribute your app to your users, you will control the brand experience for your users.
- Apps that require connectivity and constantly utilize the network significantly reduce battery life. AppSheet apps are designed for offline access with only occasional synchronization over the network. The data used by the app therefore is cached locally on the device. This does constrain the volume of data that can be used in an app-- for this reason, in the current version of AppSheet, avoid extremely large spreadsheets. All updates on the device are made locally and pushed to the backend spreadsheet only during synchronization.
-

Related articles

- *AppSheet is NOT...*
- *Create an app*
- *Make a spreadsheet*
- *Extracting structure from cloud spreadsheets*
- *Controlling data updates*

AppSheet is NOT... – AppSheet

AppSheet is not an app discovery mechanism. It is not a place where your users can "discover" the apps that you create. That is the traditional role of an app store.

Instead, we leave it to you to distribute the apps to your users. Most of our app creators are building apps for their colleagues, their employees, or their existing customers. In those cases there is already an obvious mechanism to distribute the app to the desired users. Because all you need is an installation link, it becomes trivial to distribute this link via email or any other communication mechanism.

AppSheet is also not an app marketplace. It is not a place where users buy apps or app creators sell apps. Again, that would be the role of an app store, but that is not the role of AppSheet. We are not involved in any commerce related to the apps themselves. Instead, we are providing a platform service for users to create, deploy, and run apps.

Related articles

- *Who should use AppSheet and what kind of apps can it create?*
- *Make a spreadsheet*
- *Mobile-specific features in my app*
- *Advanced app customizations*
- *Sign in*

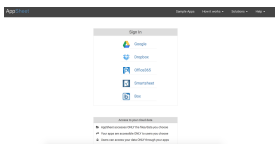
Get Started -- 0 to 60!

Sign in – AppSheet

The first step to creating apps is choosing the cloud provider with which you'll connect to AppSheet. We support Google Drive, Box, Dropbox, Office365, and Smartsheet.

There are two ways to get started. The easiest way is to use our *add-ons for Google Apps* if you're coming from Google Forms or Google Sheets. Or, you can click Sign in at the top right of the homepage. In both cases, click the appropriate cloud provider and sign into AppSheet with that account's details.

Note: AppSheet only uses your cloud provider's authentication process, and does not store any of your data on its servers. *You can find out more about the sign-in process here.*



Once you're signed in, you'll want to connect a spreadsheet that holds some data to AppSheet in order to begin creating your first app. There are a few ways to do this. If you're using Google Apps, you can use AppSheet's convenient *Google Sheets and Google Forms add-ons*. Otherwise, you can either make a new app or copy one of our samples.

In the next section we'll walk you through *how to set up your spreadsheet to be AppSheet-friendly.*

Related articles

- *Make a spreadsheet*

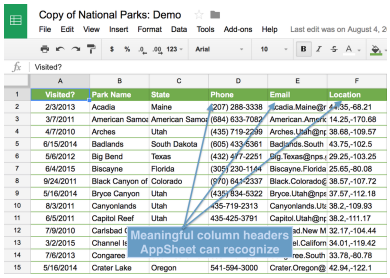
- *Create an app*
- *Advanced app customizations*
- *Deploy your app*
- *How can I receive (or send) an email alert when data changes?*

Make a spreadsheet – AppSheet

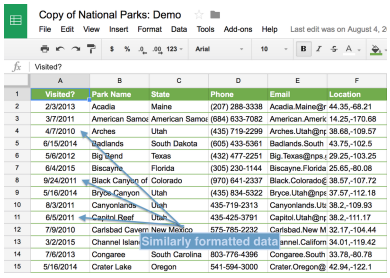
AppSheet builds an app based on the table structure of data in your spreadsheet. AppSheet will attempt to understand the structure of your spreadsheet and *there are simple ways to help with that*. The more complex your structure is, the more difficult it is for AppSheet to translate it into a working mobile app. For example, pivot tables don't work well with AppSheet's system.

First, you'll want to make sure your spreadsheet is neat and tidy. Make sure your columns are clear and if possible, add a couple of rows of data before you publish. Your spreadsheet should also have a defined set of columns and corresponding headers. AppSheet will read the table in the **first worksheet** of your spreadsheet to translate it into a mobile app. You'll have the chance to utilize other worksheets in the workbook later.

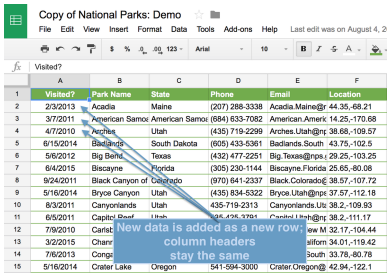
Assign meaningful headers to the columns in your spreadsheet-- this will help AppSheet identify the values in those columns. Try to use common titles in English (we plan to support more languages in future releases). For example, if you are adding a column with telephone numbers, it helps AppSheet identify the values in that column as telephones if the title (header) of the column is called Telephone, or Telephones, or Phone Number.



AppSheet will more quickly recognize the columns if the format of each cell in the column is the same. For example, if you have a column with dates, make sure that all of the dates in the column share the same format.



Finally, new data should be added as new rows, not as columns.



In the next section, we'll show you the different ways you can begin *creating apps.*

Related articles

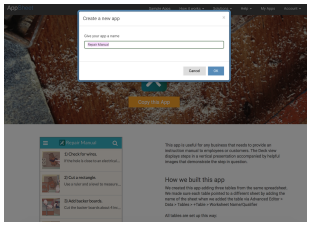
- *Create an app*
- *Effective use of column headers*
- *Basic app customizations*
- *Extracting structure from cloud spreadsheets*
- *Sign in*

Create an app – AppSheet

There are a few ways to create an app.

1. Start with a sample

You can always start by copying a sample app from our *template gallery*. Just open any app and click Copy at the top. You'll get the option to rename the app.



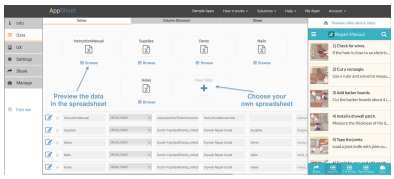
Here's what AppSheet actually does when you click the OK button:

- Creates a folder for this app at [cloud provider]/AppSheet/data/[appName].
- Copies the spreadsheet(s) used by the app into this folder.
- Generates a mobile app automatically from the spreadsheet(s).

You now have full access to the spreadsheet that was used to create this app. You can either replace it with your own data or switch to another spreadsheet. *Note: when you copy an app, you do not get a copy of the icons and logos-- you will need to add those later.*

Once you've clicked Ok, you're now taken to AppSheet's Editor tool where you'll customize your app. AppSheet has a Basic and Advanced Editor-- some apps can use both, and others are too complex for the Basic Editor. We will expand more on that later, but for now, we've landed in the Basic Editor.

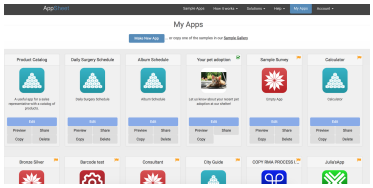
From here, play around with the Basic Editor. You can preview the data in the spreadsheet, navigate to the spreadsheet itself, or switch to a new spreadsheet.



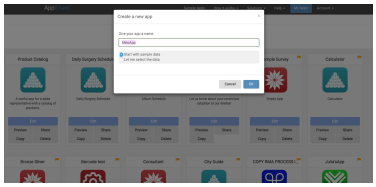
Note: if you switch to your own spreadsheet, you will need to click Refresh column structure so AppSheet expects the column structure of the new spreadsheet.

2. Make a new app

To make a new app, click the My apps tab at the top. Then click Make New App.



You'll see a window where you can name your new app and decide whether to use sample data or to select your own. Starting with sample data might be a good idea if you're not sure how to structure your spreadsheet or if you'd like to see how data in a pre-existing spreadsheet shows in an app. If you choose to select your own, you will see a popup window which will allow you to choose the spreadsheet from your cloud provider. Either way, you always have the option in the app Editors to *switch to another spreadsheet*.

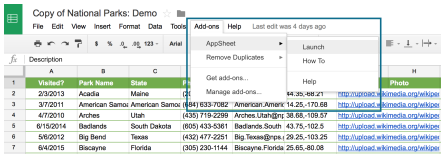


3. Create apps with AppSheet's Google Apps add-ons

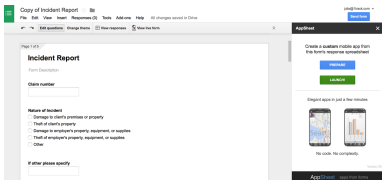
Google Sheets

You can use *AppSheet's add-on for Google Sheets* to turn your Sheets into mobile apps. This is the easiest way to create apps if you're using Google Sheets.

1. It's best to start with a Google Sheet that already has some data. AppSheet works best when it has at least a couple of rows of well-structured data to work with.
2. From your Google Sheet, click Add-ons -> Get Add-ons -> then install AppSheet (it's free). Then click Add-ons again --> AppSheet.



3. A pane will appear to the right. Click "Go"-- AppSheet will automatically generate a mobile app from your data.



4. You are now taken to a browser tab on www.appsheet.com where you can edit and refine your app. Data in the app and the spreadsheet always stay in sync.

Google Forms

The *Google Forms AppSheet add-on* works much the same way as Google Sheets, with one exception. In order to interpret the Form data and allow you to see responses, Google produces a 'response' Google Sheet for each Form. *This* spreadsheet is what AppSheet uses to generate a mobile app. This is the easiest way to create apps if you're using Google Forms.

1. Start with a Google Form where you've added a few form questions.
2. From your Google Sheet, click Add-ons -> Get Add-ons -> then install AppSheet (it's free). Then click Add-ons again --> AppSheet.
3. Click 'Prepare' in the add-on pane. After a few seconds, the 'Launch' button will be enabled. Click on 'Launch' -- AppSheet will automatically create a mobile app that has a similar mobile form.
4. You are now taken to the app Editor on appsheet.com where you can customize your app. *Note: If you make changes to your form after already having created an app, you need to go back through the Prepare and Launch steps in the add-on pane so that AppSheet can recognize the changes.*

Google Docs

The Google Docs AppSheet add-on is the fastest way to turn your paper-based forms into mobile apps because you can instantly create an app using a Google Doc that contains placeholders.

The add-on will let you build a Google Doc template with input placeholders that you can update via a mobile app. Additionally, every time a response is submitted, you get a copy of the completed template via Email and as a PDF.

1. Create a document with placeholders for input fields. Any word that has around it is an input placeholder. AppSheet will turn those placeholders into columns of a response sheet and create a mobile app where you can insert the values for the placeholder, which will then be sent to you via email and as a PDF.
2. Launch the AppSheet add-on from the add-ons menu.
3. Click "Prepare" an then "Launch" from within the AppSheet sidebar.
4. You are now taken to the app Editor on appsheet.com where you can customize your app.

4. Create apps with AppSheet's Smartsheet and Office 365 integrations

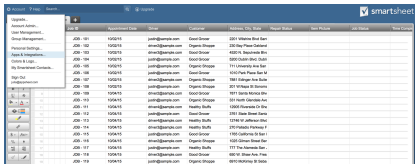
Smartsheet

Smartsheet's integration with AppSheet allows you to easily navigate from your Smartsheet to AppSheet, but isn't a formal add-on and requires you to navigate the "make a new app" process as described above in step 2.

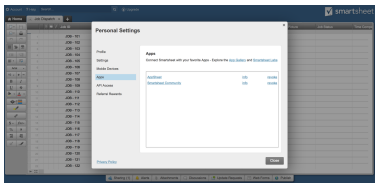
You can either:

A. sign into AppSheet directly with your Smartsheet account, and then select data from that account; or,

B. install AppSheet's integration from Smartsheet's App Gallery. Once you install AppSheet (by logging into AppSheet with your Smartsheet account from the browser popup), from there you can create apps directly from your Smartsheets by clicking the Account menu at the top, then "Apps and Integrations".



You'll see AppSheet:

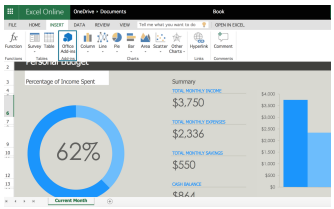


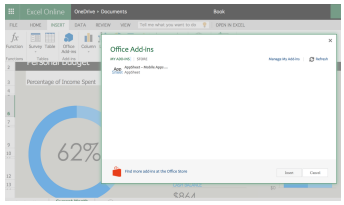
Clicking the link will take you to www.appsheet.com where you can then log in via Smartsheet and upload your Smartsheet data.

Office 365

AppSheet's add-in for Office 365 works much the same as the Google Forms, Google Docs, and Google Sheets add-ons. If you're working from an Excel file, you can add AppSheet's add-in from the Office 365 store.

1. From the "Insert" menu, click "Office Add-ins". To find AppSheet, search the store. If you've already added AppSheet, just double click the AppSheet icon. The add-in pane will appear to the right.





2. When you're ready to make an app, click "Go" on the pane.
3. You are now taken to the app Editor on appsheet.com where you can customize your app.

What's next?

Whether you've chosen sample data, made a new app from your own data, or used AppSheet's Google, Smartsheet, or Office 365 integrations, you will be taken to a page where you can use the Basic or Advanced Editors to customize your app as well as view your live app in action.

In the next section, we'll dive into the Basic and Advanced Editors, where you can make rich customizations to your app.

Related articles

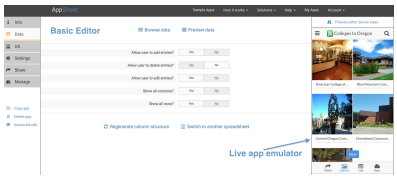
- *Basic app customizations*
- *Make a spreadsheet*
- *Advanced app customizations*
- *Sign in*
- *Deploy your app*

Basic app customizations – AppSheet

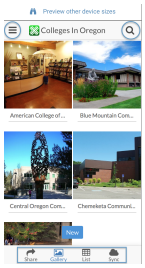
The "editor" is the AppSheet web page that lets you customize the app you just created.

Every new app creator starts with the **Basic Editor**-- this lets you edit simpler apps and shows only the most important app features. It prioritizes simplicity and ease of use. There is also an Advanced Editor that exposes the full functionality of the AppSheet model-- more about that later. For now, let's focus on the Basic Editor.

- There is a **fully interactive emulator** on the left that previews the mobile app.
- The editor tool (app definition) is shown on the right. It has three main sections-- **Data, UX, and Settings**.



First, play with the app in the **live emulator window**.

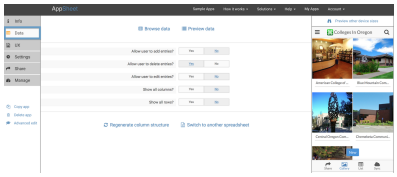


- Click on images and other content in the app-- the app reacts appropriately.
- At the bottom of the app are the 'view controls'. These allow the user to switch between different data views.
- The four buttons at the corners of the app are part of every app
- - The **Sync** button at bottom right synchronizes changes in the app with changes to the backend spreadsheet.
  - The **Share** button at bottom left lets the user send an app install link to others.
  - The **Search** button at top right allows the user to search the data in the current view.
  - The **Menu** (hamburger) button at top left provides advanced options and can be used for more views.

Explore the Data section.

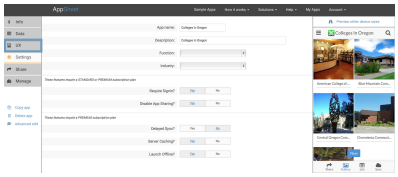
- You can see the spreadsheet data either as a preview or in a separate browser window.
- Modify and save the data (rows) in the spreadsheet, and click on the Sync button in the app emulator to see these changes in the app.

- You can decide whether the app allows the user to make data changes. Changing this option automatically refreshes the app in the emulator.
- In the app emulator, make changes to the data and click the Sync button. The changes will be pushed back to your spreadsheet.
- Modify the structure (columns) of the spreadsheet, and click on the 'Regenerate column structure' button to reflect these changes in your app.
- To use a different spreadsheet instead, click on the 'Switch to another spreadsheet' link.



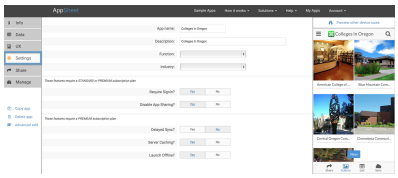
Explore the UX section.

- Some of the options here allow you to change the look and feel of the application. Try to change the app logo by selecting an image file from your cloud storage.
- Other options let you control the view controls that appear at the bottom of your app.



Explore the Settings section.

- Name your app, indicate the industry, and the app's function.
- Decide your app's level of security by choosing whether to require sign-in. If you require sign-in, you'll also get the opportunity to manage the user whitelist.
- Choose whether you'd like to run your app offline.



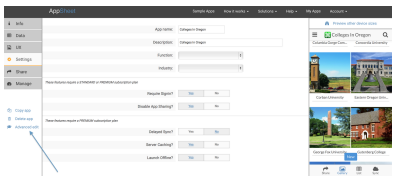
In the next section, we'll walk you through how to use the *Advanced Editor*.

Related articles

- *Advanced app customizations*
- *Create an app*
- *Make a spreadsheet*
- *Displaying images and documents*
- *Tables*

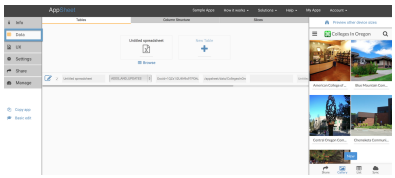
Advanced app customizations – AppSheet

For richer apps or for more advanced app features, the Advanced Editor comes in handy. You can get to it by clicking the Advanced Editor link at the top of the Basic Editor.



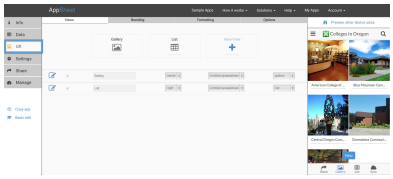
The Advanced Editor keeps the three **Data, UX, and Settings** sections-- each one has at least three subsections.

Data



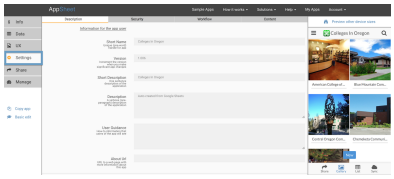
The data section contains the three subsections **Tables, Column Structure, and Slices**. You can add *multiple spreadsheets* to your app, *adjust editing permissions*, and more via the Tables tab. The Column Structure tab allows you to *make changes to the structure of your spreadsheet data*. The Slices tab is where you specify *slices* (or filters) on your data.

UX



The UX tab contains four subsections, **Views, Branding, Formatting, and Options**. Controls is where you can *choose the different views for your apps.* Branding allows you to *add your own application icons and background screens.* The Style tab enables you to choose different customizations such as *font and color themes.* Last, Rules is where you specify *change alerts/workflow rules.*

Settings



Settings contains the four subsections **Description, Security, Workflow, and Content**. Description is where you can add various informational aspects to your app. On the Security tab you can make *authentication changes to your app.* The Content tab allows you to decide various miscellaneous aspects of your app, like whether you want to be able to *launch offline or whether you'd like saved changes to automatically sync to the backend spreadsheet.*

A few other things you can do with the Advanced Editor

With the Advanced Editor, you can make much more specialized customizations to your app.

- You can add multiple spreadsheets (i.e. multiple tables) to your app. To see an example of this feature, look at the *Order Capture sample.*
- You can add multiple table slices to your app. To see an example of this feature, look at the *Marketing Event sample*.
- You can examine and modify the column structure of your data. To see an example of this feature, look at the *Consultant sample*.
- You can define extra data views for your app. To see an example of this feature, look at the *Contact Directory sample*.

- You can modify various advanced settings.

Unlike the Basic Editor, the Advanced Editor does not automatically save changes. You have to explicitly click the Save button.

In the next section, we'll show you how to *install and launch your apps.*

Related articles

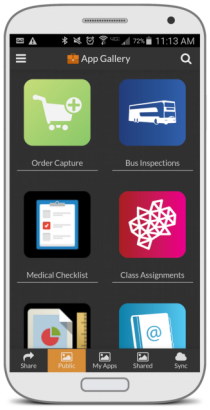
- *Basic app customizations*
- *Using multiple spreadsheets*
- *Slices*
- *Change alerts and workflows*
- *Reflecting your brand*

Deploy your app – AppSheet

In order to get your apps onto your device, you must first download the AppSheet app from your device's app store. Any subsequent apps you create essentially run through the AppSheet app. You can read more about this deployment model *here*.

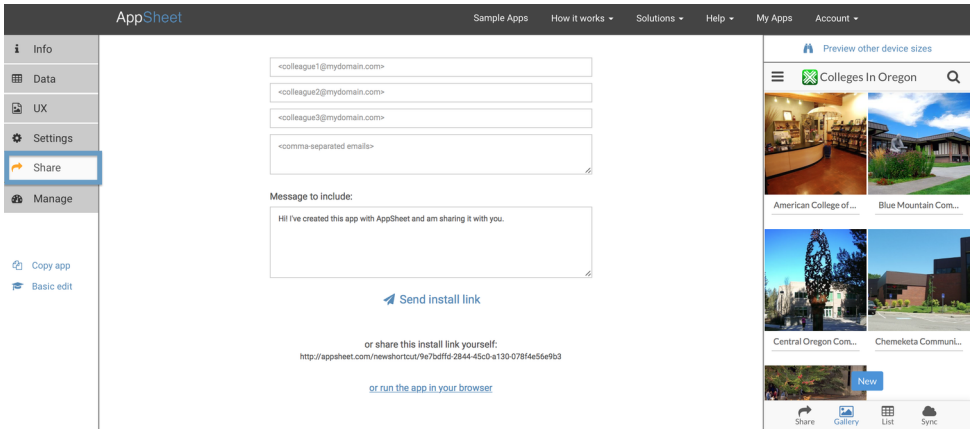
If you are an app creator, it is a good idea to install the AppSheet mobile app on your mobile device even if you won't be a user of the app. You may want to test your app out on your own device before launching to your users. You can find AppSheet's app by searching for AppSheet in the iTunes store (for iOS devices) or Google Play store (for Android devices).

When you launch the AppSheet app, it asks you to sign in, after which it starts an *App Gallery* (itself an app created by AppSheet!). The App Gallery lets you browse and run your apps, apps that have been shared with you, or public samples on your mobile device.

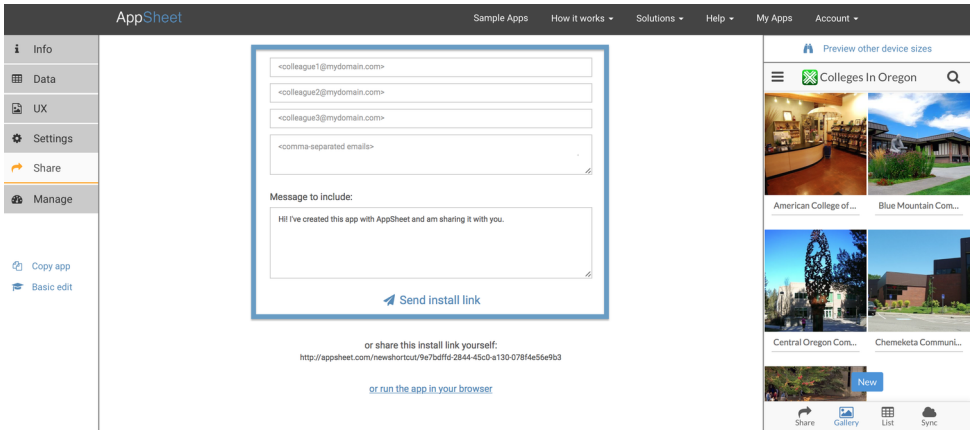


This becomes a convenient mechanism for app creators to test their apps on their own mobile device in the process of building and modifying the app definition, without having to actually install the apps themselves to the homescreen.

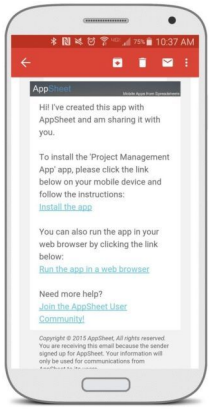
You can distribute your app by sending your users (or yourself) an install link by email. Just click the Share button at the left of the app Editor.



Then you'll be able to enter in the email addresses (or full domain) of those you'd like to share the app with.



This is what the email looks like:



When your users click on the link from a mobile device, they are asked to install AppSheet on their device (AppSheet is already available in the app stores).

Installation is easy. Find out more about installation *here*.

Related articles

- *Deployment from an install link*
- *Reflecting your brand*
- *Create an app*

- *Basic app customizations*
- *Sign in*

App Definition: Data

Data concepts

Tables – AppSheet

The first step for your data to become an AppSheet app is to be added as a table to the template of the app. It means the data is still going to be in your spreadsheet, the table is simply a "model" of the rows and columns in it. This is the first step in creating an App Definition.

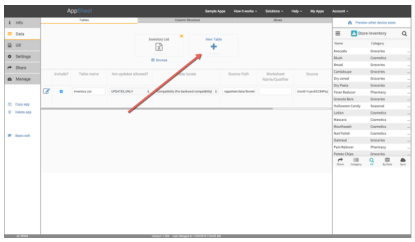
This is how we see columns and rows in AppSheet:

Columns: AppSheet reads each column header to define the Column Structure of the app. Every time you change the columns in the spreadsheet, then you need to regenerate the column structure in the app or AppSheet won't know how to locate the columns to read/write data. Column headers are essential to your app, you need column headers for each column where you want to collect data

Rows: Every time you add new data, a new row will be added to the spreadsheet. This is very important to understand. You want the columns in your table to never or rarely change while the rows can change as needed by adding, deleting or updating new data.

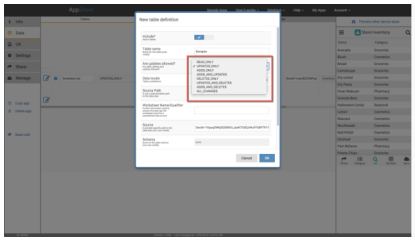
Adding Multiple Tables

To add multiple tables after you created your app, go to Advanced Edit > Data > Tables and add a new table:



AppSheet will ask you to select a new file, if you have multiple data sources, it will prompt you to select a new data source; it can be the same spreadsheet or a completely new one from a different data source. Great for data mashups!

When you add a new table - or when you are editing an existing table - you can choose how data is accessed in the app. You can allow people to Add, Update, Delete or any combination of those three.



You can also choose a specific tab inside the table by adding its exact name in the Worksheet Qualifier field.

Tables are the first step for a solid AppSheet app. And you can bring a lot of richness with multiple tables and data access levels.

A few tips for tables:

- You can add the same spreadsheet any times you need. This is useful if you want to use one table for read only (show data) and the same data table to capture data in a different view.

- Even though AppSheet can see the row number in a spreadsheet, it doesn't locate the row by it's number but rather by a data point in one of the columns. That's where the concept of *Keys* become essential in an AppSheet app
- Using keys as a way to locate rows in the spreadsheet also allows for multiple users adding data at the same time. AppSheet will "serialize" the data going back to the spreadsheet from the app, letting the last person to write in a cell to "win".

Related articles

- *Column structure*
- *References between tables*
- *Column types*
- *Slices*
- *Make a spreadsheet*

Column structure – AppSheet

Every table has a column structure which is a list of column definitions. All the rows of the table conform to this structure.

Each column definition has these attributes:

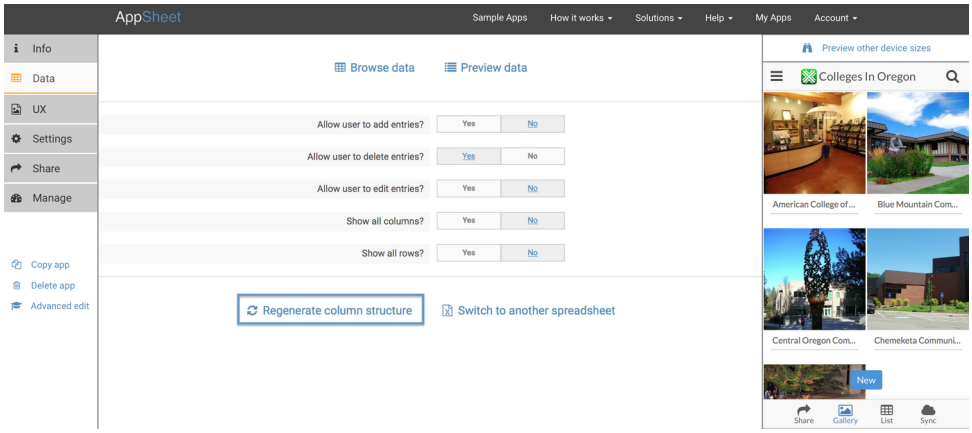
- Name: this correlates to the column header in the spreadsheet
- Description: a verbose description suitable for the application end-user
- Type: one of the column types supported by AppSheet (for example, Number or Text)
- Type Qualifier: optional further information to specialize the type
- App Formula: an expression used to compute the value of the column
- Column behaviors:
  - Is it part of a key?
  - Is it a hidden or visible in the app?
  - Is it read-only in the app?
  - Is it searchable in the app?
  - Is it scannable in the app?
  - What default value should be assigned to this column in a new row?

The type is the most important component of the column definition and has a significant impact on the behavior of the app.

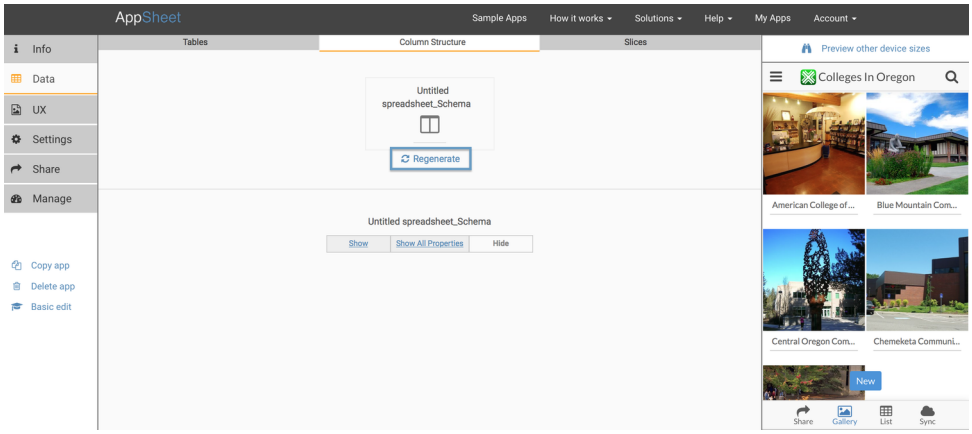
Regenerating the column structure

If you have already created an app from your spreadsheet and you go back and make changes to the spreadsheet's column structure, you will need to regenerate the column structure in the AppSheet Editor so AppSheet can recognize the new structure. You can do this in both the Basic and Advanced Editors.

In the **Basic Editor**, click the Data tab. Click "Regenerate column structure" at the bottom.



In the **Advanced Editor**, click the Data>Column Structure tab. If you have multiple spreadsheets, find the one you're looking for and click "Regenerate".



Related articles

- *Column types*
- *Modifying column structure in the editor*
- *Tables*
- *Slices*
- *Making an "AppSheet-friendl"y spreadsheet*

Column types – AppSheet

Here is a list of available column/data types. We expect this list to grow over time...

Text types

- **Text**: models a short piece of text (a few words) shown on single line.
- **Name**: special case of a Text type that represents the name of a person or place.
- **LongText**: models longer text content shown across several lines.

Numeric types - in the app, values of these types can be graphed.

- **Number**: models an integer value.
- **Decimal**: models a number with decimal precision.
- **Price**: models currency values. The TypeQualifier field can be used to indicate a currency symbol ($ is the default).
- **Percent**: represents percentage values.

Temporal types - in the app, these values are shown utilizing the timezone and presentation format of the client device.

- **Date**: models a day (day-month-year).
- **Time**: models a time within a day (hour-minute-second).
- **DateTime**: models the day and the time.
- **Duration**: models a period of time.

Change types - in some apps, it is important to record a timestamp or increment a counter automatically in a row when changes are made to other columns, and even _values_ within the columns. The change types provide this functionality. When the Type Qualifier is empty, these change types automatically update when _any_ other column value changes. However, they can be constrained to react to changes only on specific other columns by providing a Type Qualifier of the form: {"ChangeColumns":["Column1", "Column2"]}. Change types can be constrained to react to changes on certain values using this format in the Type Qualifier: {ChangeColumns:[Column1], ChangeValues: [failed,error,urgent]}.

- **ChangeTimestamp**: shows when an entry was last edited. See how this works along with a Type Qualifier in the *Store Inventory sample.*
- **ChangeCounter**: shows how many times an entry has been edited. See how this works without a Type Qualifier in the *Interview Feedback sample.*
- **ChangeLocation**: will automatically populate with the current GPS location (where the change was made).

Enumerated types - in the app, fields of these types are constrained to having one of a fixed list of allowed values.

- **Yes/No**: also known as a 'Boolean' type, these values display as Y or N in the app and have a 'slider' choice mechanism in input forms.
- **Enum**: models a text value that must belong to a specific list. The TypeQualifier contains the list of allowed values. This is typically generated automatically from data validation rules in your spreadsheet. You also have the ability to allow users to manually input data into dropdown menus instead of only having the option to choose from the dropdown list. To do this, choose the Enum type, and in the type qualifier, for example: {"EnumValues": ["One","Two","Three"],"AllowOtherValues":true} When you have an enum that allows for the option "Other", typing in the box will allow you to choose from a set of values in order to autocomplete the entry. This allows users to choose from common entries that have already been submitted, as well as to ensure all entries are submitted the same way, avoiding typos.
- **EnumList**: allows the user to select multiple answers on a question. The TypeQualifier contains the list of allowed values. This is typically generated automatically from data validation rules in your spreadsheet.
- **Ref**: an advanced feature that models a value that must be the key of another table or slice. The TypeQualifier contains the name of the referenced table or slice. See how table references work in the *Order Capture sample.*
- **Color**: color code entries in your app with a subset of 6 standard colors: Red, Yellow, Green, Orange, Purple, and Blue. See how color coding works in the *Project Plan sample.*
- **Progress**: show the progress of an entry by utilizing a âÄòHarvey Ball' ideogram. See how the progress function works in the *Project Plan sample.*

Communication types - in the app, values of these types can be tapped to launch communication.

- **Phone**: models a phone number -- gives you the option to both call and SMS text through the app. At present, AppSheet only automatically recognizes phone numbers in North American format. We hope to improve this in the future to recognize international phone numbers as well. You can overcome this limitation by using the Advanced editor and manually setting the column type to Phone. Once you do this, AppSheet will treat the column as a phone number, and the column type setting will be preserved across schema regeneration.
- **Email**: models an email address -- gives you the ability to send emails by clicking the email address.

Geographic types - in the app, values of these types can be seen on a map.

- **Address**: models a fully-specified postal address. You can store address information in AppSheet by creating adjacent columns in your worksheet and naming them appropriately. When you do this, AppSheet automatically recognized that the adjacent columns form an address. For example, you can create adjacent columns in your worksheet, and name them Street, City, State, Zip. In this case, AppSheet will recognize that taken together these columns represent an address and it will create a "Computed Address field that concatenates the values

in these address fields. AppSheet can also handle multi-line Street addresses, so you can name your columns Street1, Street2, Street3, City, State, Zip, if your application requires multiple lines of street information. This can be useful if you need to extend the address information with apartment numbers, unit numbers, attention, care of, or other such, address information. AppSheet is also capable of handling two or more addresses in the same worksheet. For example, your worksheet might contain both a Home Address and a Work Address. AppSheet recognizes the two addresses based upon a combination of naming and adjacency. Keep the Home Address columns adjacent to one another and likewise for the Work Address columns. Then name the columns to help AppSheet group them appropriately. For example, you might name the "Home Address" columns Home Street, Home City, Home State, Home Zip. You might name the "Work Address" columns Work Street1, Work Street2, Work City, Work State, Work Zip. You can use this approach to include three or more addresses in your worksheet, if necessary.

- **LatLong**: models a latitude and a longitude (eg: '48.5564, -122.3421'). Form fields for this data type can fill in the current location with a single click. See how location capture works in the *Site Inspection sample.*

Content types - in the app, values of these types are shown as inline content, or open in an external content viewer.

- **Drawing**: creates a drawing pad in the app.
- **Image**: models .jpg, .png and .gif images. The values may be image urls or names of files in the source file system of the spreadsheet. Please reference the section describing how to use files as images. Images are captured on the device using the camera or from the local camera roll.
- **Thumbnail**: also models images, but instructs the app to expect small icons and thumbnails. Thumbnails are captured just like images.
- **Signature**: models user signatures. These are captured using a touch-based signature pad and are stored as small inline images in the spreadsheet.
- **File**: models any file content that can be viewed in a browser (typically used for PDF documents). There is no capture mechanism for files in the app.
- **URL**: models any web address.
- **Show**: empty columns in your spreadsheet that serve the sole purpose of improving the presentation of **data capture forms**. There are six categories of show types:

  1. Page_Header: used to create a new page within the form
  2. Section_Header: used to create a new section within the same form page
  3. Text: used to show some descriptive text
  4. Url: used to show a clickable url
  5. Image: used to show a static image
  6. Video: used to show an MP4 video

System types - these are meant for internal use only and should not be explicitly assigned by AppSheet users in the current version.

- **MultiColumnKey**: a composite field representing the combination of multiple fields for the purpose of a key.
- **App**: represents an AppSheet app handle.

Related articles

- *App formulas*
- *Slices*
- *Multi-page forms with conditional branching*
- *Presentation types*
- *How do I control the order of rows displayed in my app?*

Keys – AppSheet

A 'key' **uniquely** identifies each individual row in the table.

The key may be a single column (such as Employee ID) or a composite column (such as FirstName and LastName).

When end-users change data on a mobile device, they are not actually interacting with the spreadsheet at that moment. They make the change to a local copy of the data and AppSheet needs to make sure the change can be applied to the spreadsheet sometime later (when the device is online and the user syncs the changes).

Without a key to uniquely identify the row changes, AppSheet would find it impossible to apply the change to the appropriate spreadsheet row.

Here's more about *keys and how to choose one*.

Related articles

- *How should I choose a key?*
- *Specifying a key*
- *Slices*
- *Concurrent usage with multiple users*
- *Customizing input forms*

Expressions – AppSheet

Please see the Expressions section for further details.

Expressions allow you to calculate new values from existing values. Most advanced apps utilize expressions to customize behavior. If you are familiar with spreadsheet formulas, you will find AppSheet expressions to be similar in syntax and meaning. At the moment, AppSheet supports a very small subset of the functions supported by Excel or Google Sheets. However, we are adding new functions every week.

Expressions are utilized in a variety of AppSheet features-- App Formulas, Default Values, Column Constraints, and Virtual Columns. The sample app *Rate Calculator* shows the use of various expressions in App Formulas.

AppSheet checks all expressions to ensure that they are correctly formed and are being used in an appropriate manner. For example, if an expression is being used to assign a default value to a column of type Number, AppSheet checks that the result of the expression is indeed a number.

Although not an exact 1:1 match to AppSheet Expressions, these *Google Functions* may provide assistance with syntax.

An expression is built by utilizing any combination of the following:

Constants

Use any of the following values as part of a formula:

- **Numbers**: You can use any number as part of the formula
- **Dates**: Use format MM/DD/YYYY
- **Words**: Use quotation marks around the word --> e.g. "Word"

Columns

Name any column using square brackets around the exact column name: **[Column]**

De-references

If a field in the column structure has a reference to a row in another table, you can use a DE-REFERENCE expression to get the value of another column in that row of the referenced table. The format to do this is **[Column With Reference].[Column in Referenced Table]**

Check out the *Order Capture* sample app to see how the Order Details table uses DE-REFERENCING to get the price of a product with the expression: **[Product].[Price]**

[Product] Is the name of the column that has a reference to the Products table.

[Price] Is the name of the column in the Products table that contains the price.

In the Order Capture app, there is also a formula that multiplies **[Product].[Price]** with another column called **[Quantity].** This is an example of a complex expression.

Built-in functions

AppSheet's list of built-in function expressions is growing every week:

- NOW() for the current DateTime
- TODAY() for the current Date
- TIMENOW() for the current Time
- HERE() for LatLong of the current Location
- USERNAME() for the Name of the current user
- USEREMAIL() for the Email of the current user
- UNIQUEID() to create unique Text values for keys
- ISBLANK(<expression>) to check if an expression is empty
- CONCATENATE(<text-expression1>, <text-expression2>, ...) to combine two or more text values
- LEN(<text-expression>) to get the length of a text value
- IF(<condition>,<then-expression>,<else-expression>)

Other Functions

- SECOND(duration) for the Second component of a Specific Time
- MINUTE(duration) for the Minute component of a Specific Time
- HOUR(duration) for the Hour component of a Specific Time
- DAY(date) for the Day of the Month
- WEEKDAY(date) for the Day Number from a Date
- WEEKNUM(date,[type]) for the Week Number from a Date
- MONTH(date) for the Month Number from a Date
- YEAR(date) for the Year from a Date

For backwards compatibility, we also support the function syntax below for a set of functions that have been supported from the earliest AppSheet release.

- @(_TODAY) for Date
- @(_TIMENOW) for Time
- @(_NOW) for DateTime
- @(_HERE) for LatLong
- @(_USERNAME) for Name
- @(_USEREMAIL) for Email
- @(_UNIQUE) for unique Text values

Arithmetic

Use any of the arithmetic operators below to build arithmetic expressions:

- **Add**: +
- **Subtract**: -
- **Multiply**: *
- **Divide**: /

Each of these operators is used with two numeric expression parameters: 3+2, 5.0/2.3, etc. Add and Subtract can also be used with dates and datetime types to add or subtract days:

- TODAY() + 1 adds a day to the current Date
- TIMENOW() - 1 subtracts a day from the current DateTime
- TODAY() - "12/12/2001" gets a Duration between the two dates
- TIMENOW() + "003:03:00" adds a 3 hours and 3 minutes to the current DateTime

Combine arithmetic expressions with parentheses to form complex expressions like: **([ColumnA]*2) + ([ColumnB]*3)** or **@(_TODAY)+([ColumnA]+1)**

Comparison conditions

AppSheet also supports comparison operators in expressions that return true or false.

- **Equals**: =
- **Not Equals**: <>
- **Greater Than**: >
- **Greater Than or Equals**: >=
- **Less Than**: <
- **Less Than or Equals**: <=

Each of these operators is used with two expression parameters that have comparable types. For example, 5 > 2 is valid, but 5 > "Hello" is not.

Composite conditions

These expression combine comparison expressions utilizing three composition operators:

- AND(expr1, expr2, ...) : returns true if all the sub-expressions are true
- OR(expr1, expr2, ...): : returns true if any of the sub-expressions are true
- NOT(expr): returns true if the sub-expression is false and vice-versa

Related articles

- *App formulas*
- *Controlling data inputs with column constraints*
- *Virtual columns*
- *Slices*
- *Modifying column structure in the editor*

Slices – AppSheet

A 'Slice' is a subset-- or filter-- of the rows and columns of a table. As the definition suggests, it has three components:

1. The table it is based on
2. The (optional) subset of columns it retains
3. The (optional) subset of rows it retains

A slice does not need to subset both the columns and the rows-- in fact, it is common to subset just the rows or just the columns as appropriate.

When a slice subsets the rows in a table, it uses a 'Filter Condition' to express the subset. A filter condition is a simple expression that can be evaluated on a single row and return true (yes, include in the slice) or false (no, exclude from the slice). For example, a filter condition 'Price > $1000' might be used to define a slice of high priced products.

The use of filter conditions makes a slice a very powerful and dynamic mechanism to model the data used in an app. While the slice definition stays fixed, the actual rows that participate in the slice change as the underlying table data changes.

Here's more about *how to implement slices*.

Related articles

- *Defining and using slices*
- *Advanced app customizations*
- *Expressions*
- *Spreadsheet formulas*
- *Keys*

Effective spreadsheet design

Making an "AppSheet-friendly" spreadsheet – AppSheet

You can control column structure by changing your spreadsheet.

- Use the first worksheet-- by default, AppSheet only uses the first worksheet of your spreadsheet file. And it only extracts one structured 'table' of data from that spreadsheet. So it is important that you put the appropriate data in the first worksheet. Note that AppSheet apps can utilize multiple spreadsheet files, a feature described in the Advanced Editor section. Also as described in that section, you can explicitly specify a worksheet other than the first worksheet, if necessary.
- Just header + data please-- you could put almost any content in a spreadsheet, including charts, pictures, free form text, etc. However, if the data isn't obviously tabular, AppSheet will probably fail to extract a tabular column structure. Ideally make the first row contain column headers and the rest of the rows contain data.
- Provide data for column type deduction-- if possible, provide at least 5-10 rows of actual data in the spreadsheet. This allows AppSheet to look at the data and deduce their types. For example, if every row of a column had an entry that looks like 'something.jpg', AppSheet can recognize that this is an Image. However, users sometimes start with a blank spreadsheet and that leads AppSheet to make poor guesses.
- Use data formats consistently, as described in *the next article*.
- Use spreadsheet data validation rules where appropriate, as described in *a later article*.
- Provide data for key identification, as described in *a separate article*.
- If AppSheet still doesn't quite get the column structure right, you can tweak it in the *Advanced Editor*.

Read a detailed blog post about structuring your spreadsheets.

Related articles

- *Specifying a key*
- *Advanced app customizations*
- *App formulas*
- *Make a spreadsheet*
- *How do I design a secure app?*

Effective use of column headers – AppSheet

AppSheet infers the types of columns from the column header names as well as from the content of the rows. Especially in cases where there are no existing rows, it is important to pay attention to the column header names.

There are special words in column headers that 'trigger' AppSheet to infer specific column types. For example, a column header name 'Web Site' suggests that the data is of the URL type. Currently, this only works with English but we intend to add similar capabilities for other languages in the future.

There are a few special cases to be aware of:

- A column header ending with a question mark is inferred as a Yes/No data type.
- A column header ending with an exclamation mark is inferred as an action Url
- A column header whose name is similar to another table already in the app may be inferred as a table Ref type. For example, if there is a table called Products and a new spreadsheet is added with a column called 'Product' or 'Products', AppSheet will infer that it is a Ref type.

If you are building an app from a Google Form and your question is marked as Text, a similar inference process occurs on the question title.

Here is a complete list of field types, what they do, and the trigger words that activate them:

- **Address**

What it does: Identifies the content of the field as a full postal address that can be mapped. The completeness of the address will increase the accuracy of the map location. Trigger words: "address", "where"

- **ChangeCounter**

What it does: Shows how many times an entry has been edited. **Trigger words:** No trigger words. Must be selected in the Advanced Editor.

- **ChangeLocation**

What it does: Automatically populates the GPS location where the change was made. **Trigger words:** No trigger words. Must be selected in the Advanced Editor.

- **ChangeTimestamp**

What it does: Shows when an entry was last edited. **Trigger words:** No trigger words. Must be selected in the Advanced Editor.

- **Color**

What it does: Displays a color entry in the app from a subset of 6 standard color names. Use IF Conditions in the spreadsheet to populate the color name based on a specific field. **Trigger words:** "blue", "green", "orange", "purple", "red", "yellow"

- **Date**

What it does: Models a day (day-month-year). **Trigger words:** "birthday", "dob", "day", "month", "year"

- **DateTime**

What it does: Models the date and the time (day-month-year hour-minute-second). **Trigger words:** "date", "dates", "day", "days", "month", "months", "year", "years", "timestamp"

- **Decimal**

What it does: Models a number with decimal precision. **Trigger words:** "altitude", "altitudes", "amount", "amounts", "amt", "amts", "age", "ages", "capacity", "capacities", "depth", "depths", "displacement", "displacements", "height", "heights", "hours", "latitude", "latitudes", "length", "lengths", "longitude", "longitudes", "magnitude", "magnitudes", "mass", "masses", "population", "populations", "pop", "qty", "qtys", "quantity", "quantities", "size", "sizes", "sum", "sums", "total", "totals", "units", "weight", "weights", "width", "widths", "volume", "volumes"

- **Drawing**

What it does: Enables a large field to make a simple drawing or capture notes. Choose from a set of seven colors. **Trigger words:** "depiction", "diagram", "drawing", "illustration", "layout", "likeness", "rendering", "sketch"

- **Duration**

What it does: Models a period of time. Data should be in the format HH:MM:SS. **Trigger words:** "duration", "timespan", "period", "elapsed"

- **Email**

What it does: Enables to send an email using the default email app in the mobile device. **Trigger words:** "email", "e-mail"

- **Enum**

What it does: Picks up data validation from the spreadsheet and displays the list in the app. The list is picked only the first time the app is created and is not dynamically refreshed. Trigger words: No trigger words. Automatic if there is a validation list in the column.

- **EnumList**

What it does: Allows multiple choice selection from a validation list. **Trigger words:** No trigger words. Must be selected in the Advanced Editor.

- **File**

What it does: You can reference files in your cloud drive (PDFs and similar documents) By adding a string indicating the location of the file in relation to the spreadsheet. For example. Reference/guide.pdf Ãü AppSheet will locate the file and add a link to it from the app. **Trigger words:** "file", "files"

- **Image**

What it does: Displays .jpg, .png, and .gif it also allows to capture images using the phone's camera. **Trigger words:** "image", "images", "picture", "pictures", "photograph", "photographs", "photo", "photos", "figure", "figures", "fig", "figs", "icon", "icons", "illustration", "illustrations", "snapshot", "snapshots"

- **LatLong**

What it does: Reads latitude and longitude data (e.g. 48.5564, -122.3421) It can place the location on a map or use the mobile device location to populate the field. **Trigger words:** "latlong", "geolocation"

- **LongText**

What it does: Shows a large text box. **Trigger words:** "notation", "notations", "note", "notes", "desc", "description", "descriptions", "comment", "comments"

- **Name**

What it does: Text type that represents the name of a person of place. Shown as Text. **Trigger words:** "first", "givenname", "given-name", "last", "middle", "mi", "name", "names", "surname", "surnames"

- **Number**

What it does: Models an integer value. **Trigger words:** "altitude", "altitudes", "amount", "amounts", "amt", "amts", "age", "ages", "capacity", "capacities", "depth", "depths", "displacement", "displacements", "height", "heights", "hours", "latitude", "latitudes", "length", "lengths", "longitude", "longitudes", "magnitude", "magnitudes", "mass", "masses", "population", "populations", "pop", "qty", "qtys", "quantity", "quantities", "size", "sizes", "sum", "sums", "total", "totals", "units", "weight", "weights", "width", "widths", "volume", "volumes"

- **Percent**

What it does: Shows percentage symbol. **Trigger words:** "%", "discount", "discounts", "interest", "percent", "percentage", "chance", "percentages", "probability", "probabilities", "quota", "quotas", "split"

- **Phone**

What it does: Enables calling or texting a phone number listed in the field directly from the app. **Trigger words:** "cell", "cellphone", "fax", "mobile", "pager", "phone", "phonenumber", "tdd", "tel", "telephone", "telex", "tty"

- **Postal Code**

What it does: This helps AppSheet detect an address when an address spans multiple columns. For example, a "Street" column, "City" column, and "Postal Code" column. **Trigger words:** "zip", "zipcode", "postalcode"

- **Price**

What it does: Shows a currency symbol. Use the **TypeQualifier** field to indicate anything different than **$. Trigger words:** "amount", "amounts", "amt", "amts", "balance", "balances", "bonus", "bonuses", "cash", "commission", "commissions", "compensation", "contribution", "contributions", "cost", "costs", "discount", "discounts", "dividend", "dividends", "donation", "donations", "dues", "duty", "duties", "earnings", "excise", "expense", "expenses", "fee", "fees", "fine", "fines", "gain", "gains", "gross", "indemnity", "income", "levy", "levies", "loss", "losses", "pay", "payment", "payments", "payoff", "pension", "pledge", "pledges", "premium", "premiums", "price", "prices", "proceeds", "profit", "profits","receipts", "reimbursement", "reimbursements", "remittance", "remittances", "remuneration", "remunerations", "rent", "rents", "revenue", "revenues", "royality", "royalities", "salary", "salaries", "sales", "stipend", "stipends", "subtotal", "subtotals", "tariff", "tariffs", "tax", "taxes", "tip", "tips", "tithe", "tithes", "toll", "tolls", "total", "totals","value", "values", "wage", "wages", "winnings"

- **Ref**

What it does: Models a list picked from another table. The **TypeQualifier** must contain the name of the referenced table. Tables can be added in the Advanced Editor>Data>Tables section. **Trigger words:** No trigger words. Must be selected in the Advanced Editor.

- **Signature**

What it does: Enables a signature capture field. The signature is saved in the app creator's Google Drive. **Trigger words:** "signature", "signatures"

- **Thumbnail**

What it does: Models images but instructs the app to expect small icons and thumbnails. They are captured just like images. **Trigger words:** "thumbnail", "thumbnails", "logo", "logos"

- **Time**

What it does: Models a time within a day (hour-minute-second). **Trigger words:** "hour", "hours", "mins", "minute", "minutes", "second", "seconds", "secs", "time", "times"

- **URL**

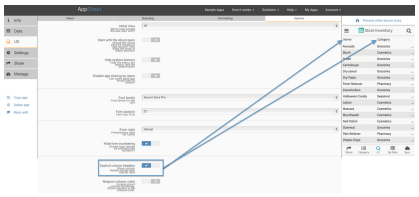
What it does: Displays a URL, when clicking on it, the default device browser will load the page. **Trigger words:** "site", "sites", "url", "urls", "web", "website", "websites"

- **Yes/No**

What it does: Also known as Boolean. It displays a Y or N or N/A option. **Trigger words:** Starts with "has" or "is" or ends with "?".

Displaying column headers in the app

You can choose to display your spreadsheet's column headers in your app using the Advanced Editor. Go to UX>Options and click the 'Explicit column headers' check box.



Related articles

- *Column types*
- *Specifying a key*
- *Using formats and data validation rules*
- *Make a spreadsheet*
- *Create an app*

Specifying a key – AppSheet

It is very important that AppSheet pick a good key for your dataset. AppSheet needs to be able to uniquely identify each row of data in the spreadsheet. The 'key' is a single column that provides this unique identity. This helps AppSheet synchronize changes made in the app back to the spreadsheet.

Many data sets naturally have a key, i.e. a column whose value does not change and that uniquely identifies the row. For a product catalog, the product ID or the product name are good keys. For a customer database, the customer name or ID makes a good key. Make your key column the leftmost column if possible. AppSheet processes the data left to right looking for a key column. Key columns should not have duplicate values. AppSheet checks the data to see if there are duplicates, and ignores some column types that are typically not good keys (e.g. a URL or a timestamp). If no obvious unique key is found, AppSheet tries to combine columns to form a 'computed' key. And if no other option exists, the rownumber is picked as the key, but this has various limitations so try to avoid this if possible.

Generating a random key

You have the option of generating a random text key for each entry to uniquely identify it. This is useful for forms inputs that need to generate a key field. To do this, use the Advanced Editor. Click the Data tab, and then the Column Structure tab. Scroll to the field name you would like to have a random text key, then scroll to the right, and in the Default Value field, type *UNIQUEID()*.

AppFormulas and keys

AppFormulas may be used in key fields provided the AppFormula always yields the same result over time. That ensures that the key value remains consistent over the life of the row. We examine the contents of the key field's AppFormula to ensure that it yields the same result over time. We display an error if it does not. For example, an AppFormula that includes the current date or time might yield different results over time, so it would be prohibited in a key field's AppFormula.

Using multi-column keys

You can also specify more than one column as the key in your app. This will combine the columns you choose, showing all chosen columns' values next to each other in the app. You would do this to create a more refined key. From the Advanced Editor, click the Data tab, then 'Column Structure.' Under 'Primary key?' check any number of boxes to form a multi-column key-- this will appear at the bottom as the '_ComputedKey'.

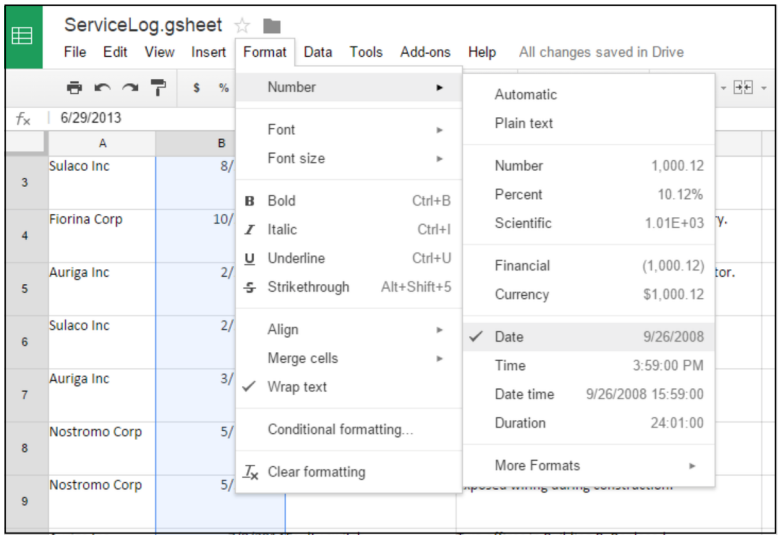
Related articles

- *Column types*
- *How do I control the order of columns displayed in the app?*
- *Keys*
- *Spreadsheet formulas*
- *How do I control the order of rows displayed in my app?*

Using formats and data validation rules – AppSheet

Spreadsheets have a built-in mechanism called 'formats' to indicate the type of data in a cell. Especially when using dates and times with custom formats, please assign spreadsheet formats.

Spreadsheets allow you to pick a format for each cell. However, AppSheet is deducing a data type for every column, not every cell. So make sure to apply the same format to all non-header cells in a column. This is the most normal use of formats anyway. In the case below I'm making sure to set my Date column to the Date format under the Format menu.



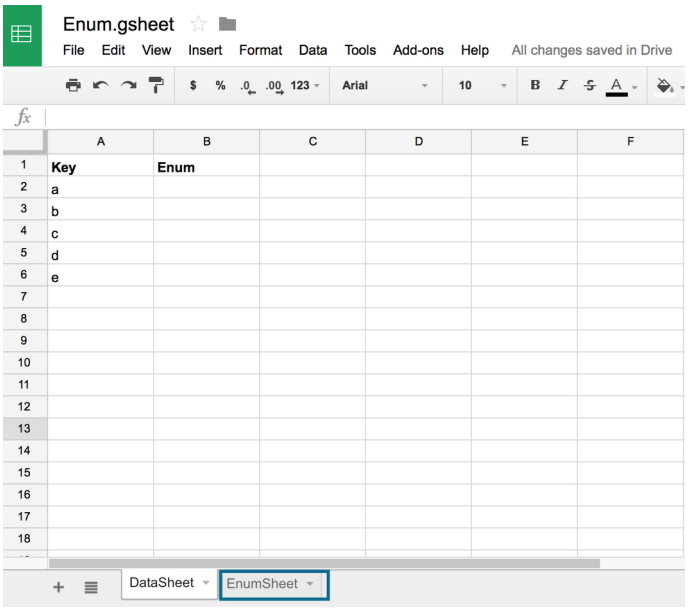
Specifying data validation rules (dropdown menus, or enums) in the spreadsheet

Spreadsheets have a built-in mechanism called data validation rules to constrain the allowed values in a cell. Utilize this mechanism if you want a column to have an 'enumeration' type, i.e. a dropdown list of allowed values. You can either do this by manually typing in the allowed values into the validation list (not recommended), or selecting a predefined set of cells that already contain the allowed values (recommended).

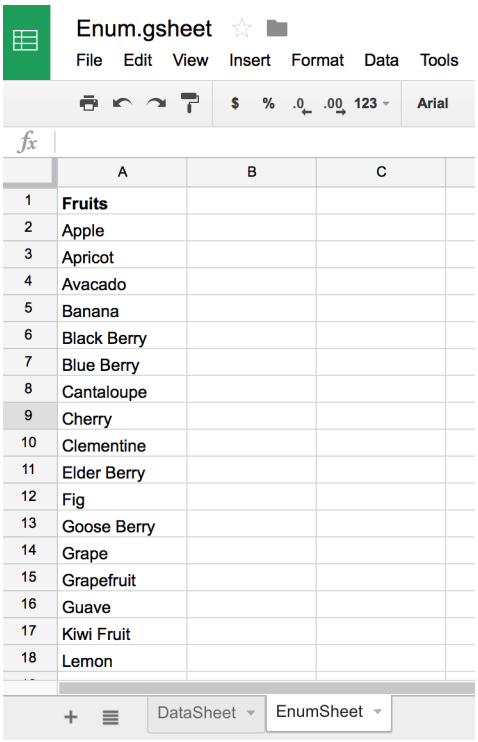
When you have lots of legal enum values, we suggest creating an additional worksheet in your workbook to contain all of these enum values. The alternative, manually typing in the allowed values into the validation list, imposes a limit of 256 characters for the entire list. For clarity, we will refer to the original worksheet containing your application data as the DataSheet. We will refer to the legal enum values worksheet as the EnumSheet.

Do the following:

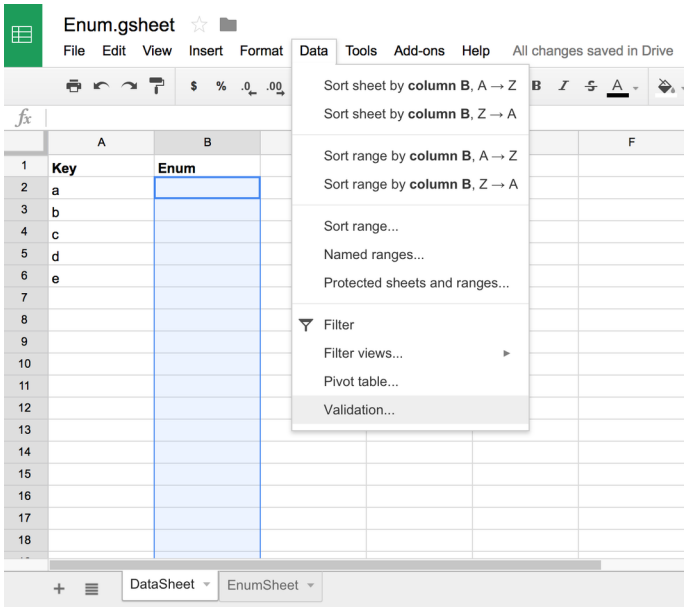
1. Add a new worksheet to your Google workbook to contain your legal enum values. I am referring to this as the EnumSheet.



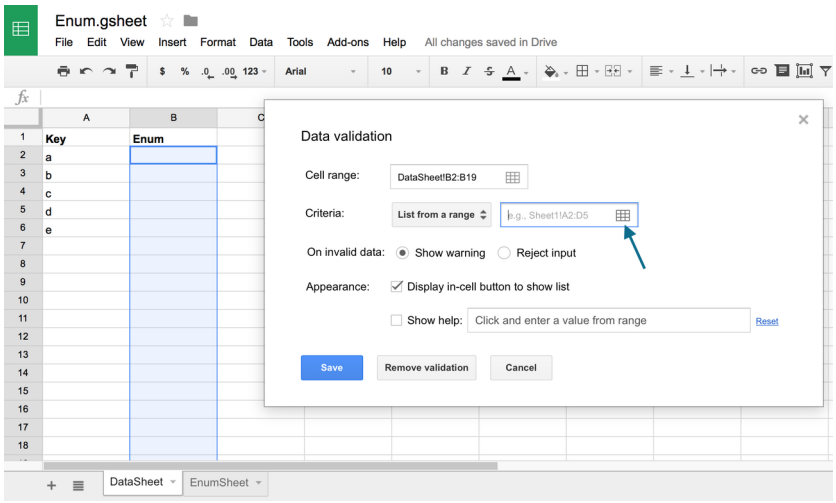
2. Open the EnumSheet and allocate a column for your first set of valid enum values.
3. Enter a column header value. In my case I entered Fruits in cell A1.
4. Enter all of the legal enum values into this column. In my case I entered Apple in cell A2, Apricot in cell A3, Avacado in cell A4, and so forth.

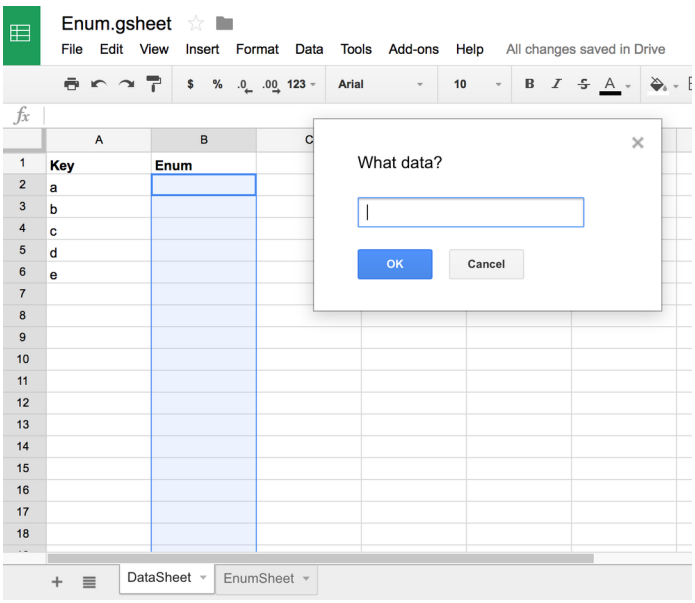


5. In the DataSheet go the column that will contain the enum values and select all of the cells in the column.



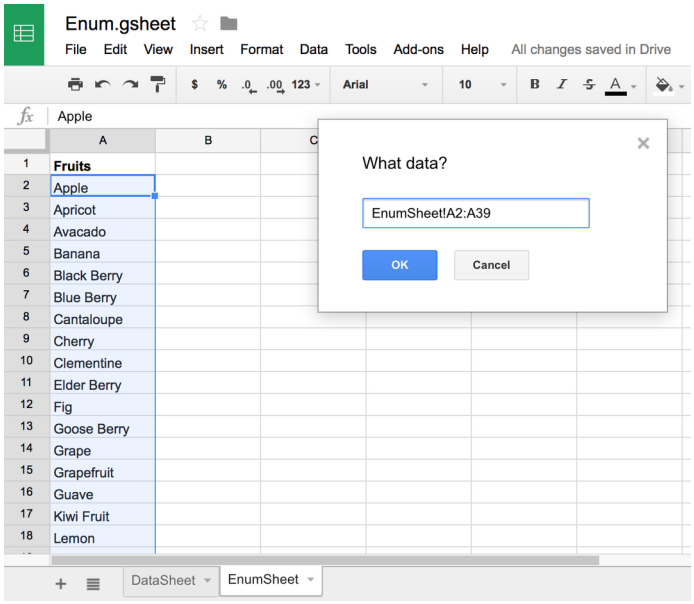
6. Go to the Data menu and select Validation.
7. The Data validation dialog window will appear. In the Criteria row select List from a range and then click the Select data range button just to the right of the drop down list.



8. 

The What data? dialog window will appear.

9. With the What data? dialog window still open, click the EnumSheet tab to open the EnumSheet.

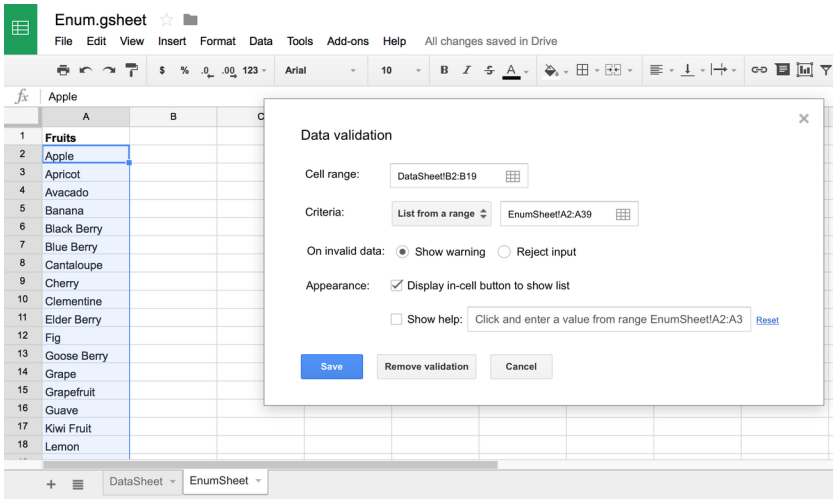
10. Select the entire range of cells in the column containing your legal enum values. In my case, I selected the cells 

containing Apple, Apricot, Avacado, etc.

11. The What data? dialog should now contain something like EnumSheet!A2:A15. The EnumSheet name appears to the left of the ! and the selected enum values cell range appears to the right.

12. Click OK in the What data? dialog.

13. The Data validation window will now reflect the EnumSheet name and cell range you just selected above. These are the enum data values that will be used for validation.



14. Hit Save in the Data validation dialog.
15. If you have multiple sets of enum values, you can add a column in the "EnumSheet" for each set of enum values. For example, you might add a column for "Vegetable" enum values, another for "Animal" enum values, and so forth. Repeat steps 2 through 14 for each of your enum values.
16. Ensure that at least one row in the "DataSheet" contains data values. For each enum column, choose one of the legal enum values as the data value.
17. Generate your AppSheet application from the workbook (use the add-on). AppSheet will read the "DataSheet" to get your data values. It will read the Enum Sheet to get your enum values.
18. Once you have created your app, open the Data>Column Structure tab to ensure all of the enum values have been picked up by AppSheet. They should appear in the Type Qualifier column for each of your Enum fields.

Related articles

- *Column types*
- *Spreadsheet formulas*
- *App formulas*
- *Conditional formatting*
- *Modifying column structure in the editor*

Spreadsheet formulas – AppSheet

Spreadsheets have very rich and expressive formula capabilities. These are well-documented by the spreadsheet providers like Excel and Google Sheets. When the mobile app reads data from the spreadsheet, the formula values are computed. Likewise, when data is updated and synced back to the spreadsheet, the appropriate formulas are computed again.

There are two kinds of spreadsheet formulas:

1. In-row formulas: these are formulas that only use values from other cells in the same row. For example, if the formula for cell C2 = A2 + B2
2. Multi-row formulas: these are formulas that use values from cells in other rows. For example, if the formula for cell C2 = C1 + 1

Columns with spreadsheet formulas are treated as Read-Only by AppSheet. This is because spreadsheet formulas cannot be evaluated in the mobile app. However, they are evaluated when data is sent to or from the backend spreadsheet.

Formula consistency

If all the cells in a column have the same formula, AppSheet recognizes that the intent is for all new rows to also have that formula. As a result, new rows inserted will include the formula.

On the other hand, if some cells in a column have a formula and other cells have either no formula or a different formula, AppSheet will issue a warning to the app creator. The column will be marked Read-Only but new rows will not be assigned a formula in this column (because it is not clear what the desired intent is).

Multi-row formulas with aggregates

A common use of multi-row formulas in a spreadsheet is to compute aggregate functions (totals, averages, etc). For example, a spreadsheet with individual order items followed by a row with the order total. AppSheet apps require that the rows in the same table be all of a similar nature --- i.e. all order items --- because clearly there are different app behaviors expected with order items and order totals.

AppSheet ignore rows with aggregate multi-row formulas after issuing a warning to the app creator. This applies to both "totals rows" and "sub-totals" rows.

Note: spreadsheet formulas are distinct from *App Formulas*. The two mechanisms are complementary and both add value to your app. App Formulas run while the user is updating data in the app, and will work even when the app is offline.

Related articles

- *App formulas*
- *Expressions*
- *Specifying a key*
- *Make a spreadsheet*
- *Slices*

Limits on data size – AppSheet

AppSheet is meant for mobile apps that are designed to work seamlessly despite intermittent connectivity or completely offline. As a result, all data used by the app must be cached locally on the mobile device. This is an important factor to consider when designing your app.

Ideally, you should make your data set as small as possible to achieve the desired functionality.

Actual limits

Do not build an AppSheet app against a huge data set. What is "huge"? For an AppSheet app, the compressed data size limit is 5MB or 10MB (depending on the device) for all the data in one app. It is difficult to translate this accurately into a specific number of rows or columns because compressed data size depends on how much repetition there is in the data. For example, a large spreadsheet with a lot of empty cells will probably compress better than a smaller spreadsheet with no empty cells. In general, the fewer cells in the sheet, the better.

In the future, we may increase the 5MB/10MB limit.

"External" data like images and documents are not included in this data size limit. You can definitely have applications with many rows and an image in every row. Images and documents are not cached locally on the device by default. If you do enable the option to cache images for offline access, they are stored in a different location on the device that does not have the same size limitations.

Performance concerns

In practice today, this is not a meaningful limit, because system performance degrades well before you reach the limit. This happens for three reasons:

1. Slow iterative development-- the data set is checked repeatedly during app development to ensure that the app is consistent with the data
2. Long sync times-- when data is synchronized between the device and the backend, the delay depends on the size of the data set.

3. Sluggish app behavior-- large data sets can make the app itself sluggish in its interactions like scrolling, search, etc.

Mitigations

Always check if your sheets have empty rows in them. In particular, some sheets have hundreds of empty rows at the bottom. Removing these rows will improve performance.

Also check if your sheets have many extra worksheets that are not being used by the app at all. Removing extra worksheets will improve performance.

In some circumstances though, the underlying dataset is large and just cannot be changed. If so, consider using either of these options:

1. Use a Security Filter to forcibly limit the data sent to the mobile app.
2. use a slice to subset the data and strictly use only the slice in your app. This allows AppSheet to send just that subset of data to the device.

It is often the case that even if an app requires large data sets, much of the data is read-only. In this situation, you can improve some aspects of app performance by separating the read-only data sets into a separate table or tables from those that are writeable/updateable.

Related articles

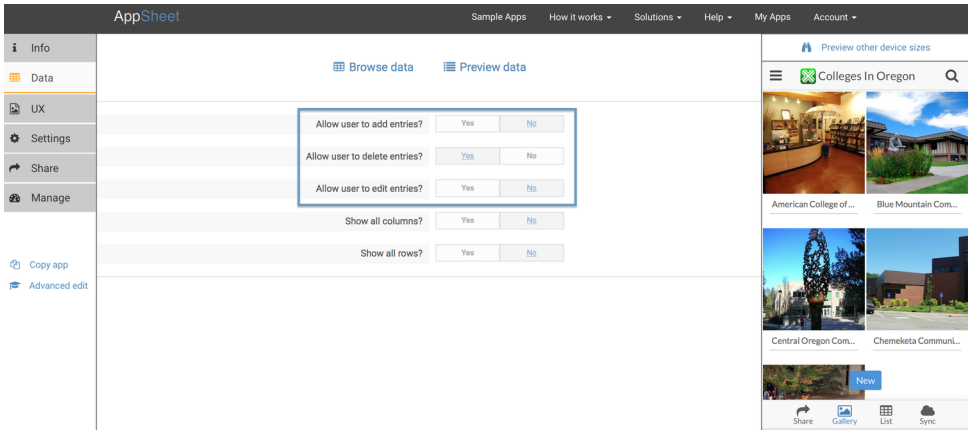
- *Expressions*
- *Slices*
- *Including PDF's in your application*
- *Virtual columns*
- *Conditional formatting*

Data definition in the app editor

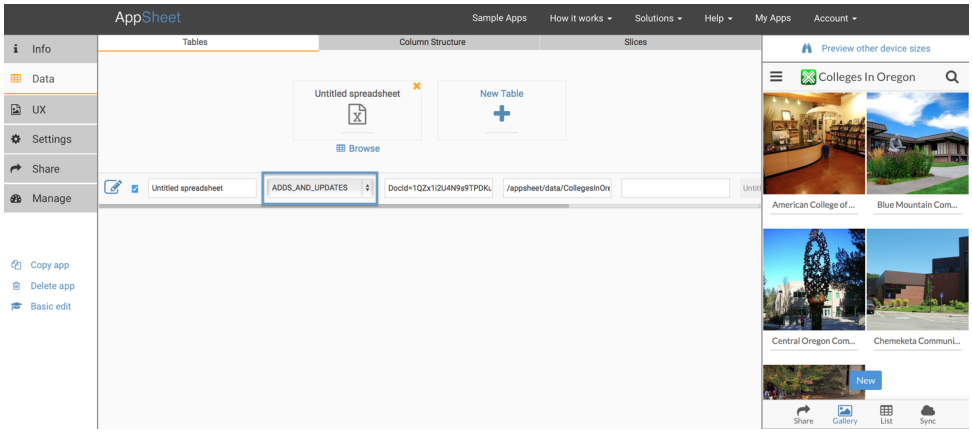
Controlling data updates – AppSheet

You can control whether the users of the app can update the data.

You can control the update modes of your app with a great degree of granularity. On the Data pane of the **Basic Editor**, you can independently specify whether users can **Add, Delete,** or **Edit** data. All combinations are allowed.



You can also do this on the Data>Tables pane of the **Advanced Editor**:



Related articles

- *Data access for different classes of users*
- *Modifying column structure in the editor*
- *Advanced app customizations*
- *Defining and using slices*
- *Controlling data inputs with column constraints*

Modifying column structure in the editor – AppSheet

You can tweak the schema generated by AppSheet. But please do so carefully!

In the Data pane of the Advanced Editor, if you choose the Column Structure tab, you will see the column structure that AppSheet has deduced for your data. As we have discussed, this is often not exactly what you want. The best way to change it is to change your spreadsheet, but if that is not possible, you may also decide to change the schema directly. These changes can include:

- **Column Name**There should rarely be a reason to change the field name. If the field name is not the same as the column header name in your spreadsheet, errors will occur. This capability is useful for advanced scenarios where the data comes from non-spreadsheet sources.
- **Column Type and Type Qualifier**There are approximately 20 different data types and we expect this to grow. (*You can find the official list here.*) The choice of data type affects the behavior of this field in the app. For example, you might want a particular text column to be a 'LONGTEXT', or a particular image column to be a 'THUMBNAIL'.
- **App Formula**App formulas are expressions that are automatically computed in the app to populate the value of a column. This is not to be confused with spreadsheet formulas, which are computed in the backend spreadsheet.
- **Primary Key**Keys should be chosen with care, as discussed in an earlier section. Please note that only one column can be a key. You will find composite columns at the bottom of the column list, and one of them may also be chosen as a key.
- **Read-Only, Hidden, and Required**Individual fields may be marked read-only to prevent updates in the app, hidden to prevent display, or required to mandate entry.
- **Column Description**The description, if provided, is shown to the user instead of the field name in forms. It is automatically populated from comments on the column header cells in the spreadsheet.
- **Default Value**: when a new entry is created in the app, the fields are populated with default values that are typically constants but can be any supported *expression* that matches the expected column type.
- **Searchable**Indicates that a column should participate in the search function. By default, most text-based columns do participate in search, but content columns (like images) do not.
- **Scannable**Indicates that a column can be filled by scanning a barcode or QRCode. By default, no column is scannable unless explicitly indicated.
- **Sheet Formula**Indicates a formula on the backend spreadsheet that is calculated (and the result shown in the app) once the user hits "Save" (or "Sync" if the app is set to "Delayed Sync").

Related articles

- *Column types*
- *App formulas*
- *Defining and using slices*
- *Specifying a key*
- *Multi-page forms with conditional branching*

Controlling data inputs with column constraints – AppSheet

Every column definition has a type (that specifies what values are allowed in the column) as well as flags that specify whether the column is hidden, whether it is required for input, etc. This is adequate for many apps, but sometimes a more dynamic or data-driven mechanism is needed. This is what Column Constraints provide-- a column constraint is an *expression* that is dynamically computed to determine the behavior of a specific column in an input form.

Four different column constraints can be defined for each column:

- Show_If
- Valid_If
- Required_If
- Editable_If

To add a column constraint, use the Column Structure tab of the Data pane in the *Advanced Editor* and click the "Edit" icon at the left of the corresponding column definition.

Expression components

These expression components utilize the following formatting:

- Name any column using square brackets around the exact column name: **[ColumnName]**. When combining a [ColumnName] with another value, put the expression in (parentheses).
- Use **[_THIS]** as a "virtual column name." It refers to the value of the current column (used in Valid_If, Show_If, Required_If conditions).
- Use **[_THISROW]** as a "virtual reference column." It refers to the current row. For example, use [_THISROW].[ColumnName].
- Highlight all values with "quotes," except for numeric values: **"Value"**.
- Numeric values are noted just as they are, e.g **10**.

Show_If

A Show_If column constraint is used when an input field should be shown or hidden depending on the values of one or more earlier field values in the form.

A Show_If constraint is a condition expression that determines whether or not an input for this column should be shown in a form. This is usually based on the values of other columns. For example, the condition expression ***[UserRating] = ""5*** will display this column if the value in the "UserRating" column is "5". The condition expression ***[Status] = "Gree"n*** will display this column if the value in the "Status" column is "Green"**.**

The special column name "_THIS" is used to refer to the current column being constrained. For example, a Show_If condition of ***ISBLANK([_THIS])*** can be used to show an input field only if the column itself is blank. The column will be hidden, once a data value has been entered and saved for the column.

Show_If conditions can also be defined on Page Header columns to *conditionally show or hide entire pages*.

Valid_If

Every input in a form is checked for validity based on its type. For example, a column of Number type will not accept "Hello" as an input. A Valid_If column constraint is used in situations where the validity of the input requires richer data-dependent logic.

A Valid_If constraint is a condition expression that determines whether or not the form input for this column is valid. For example, the *Quote Calculator sample* utilizes a Valid_If condition to ensure that the Cost Per Hour must be less than $20.

Here are examples of commonly used Valid_If constraints:

1. Does the value of the column satisfy a simple condition?
   - For example, comparing the value with a constant: ***[_THIS] > 5***
   - For example, comparing the value with another column: ***[_THIS] > [ColumnA]***
2. Is the value of the column in a list?
   - A list of constant values. For example: ***{100, 200, 300}***
   - A list of values from a column in another table. For example: ***LookupTable[ColumnC]*** . This is particularly useful if the list of allowed values should itself be allowed to change while the app is being used.
   - A list of values from specific rows in another table. For example: ***SELECT(LookupTable[ColumnC], [ReportDate] > Today() - 7)*** specifies that the rows in LookupTable should be filtered to find those where ReportDate is within the last week, and the corresponding values in ColumnC of LookupTable become the allowed list of valid values.

Whenever a list of allowed values is provided, it is actually a syntactic shortform for an expression that uses the IN function. For example, ***{100, 200, 300}*** is the same as ***IN([_THIS], {100, 200, 300})***. In input forms in the mobile app, columns with such Valid_If constraints are provided with dropdowns or enumeration selectors that reflect the list of choices.

Valid_If and Dependent Dropdowns

Dependent dropdowns are a common design pattern in apps that capture input. For example, consider an app like *this Lead Tracker sample* that asks for the 'Lead Region' (America, Asia, Europe) and then for a 'Country' within that region. This is actually requires relatively complex logic, but AppSheet tries to make it simple. Dependent dropdowns are driven by a separate lookup table.

In the sample, there is a separate 'Regions' lookup table with two columns: 'Region' and 'Country'. This acts as the lookup table for allowed combinations of regions and countries. *Here is the table data* used in the sample.

The 'Lead Region' column has a regular Valid_If constraint: ***Regions[Region]*** . Therefore, when a new entry is being added, the input for this column shows three choices: America, Asia, and Europe.

Likewise, the 'Country' column also specifies a similar Valid_If constraint: ***Regions[Country]*** . However, because it follows the 'Lead Region' column and because both specify columns from the same lookup table 'Regions', AppSheet recognizes the intent and implements a dependent dropdown.

Internally, AppSheet creates an expression to capture the allowed set of values for the 'Country' column. The expression must say (in English!):

- Look at the Regions table
- Filter the rows to make sure that the Region column of the table matches the value in the 'Lead Region' column of the row being edited in the form
- Now extract the 'Country' column from those filtered rows
- Eliminate any duplicates --- these are the allowed countries!
- Recompute this list each time the 'Lead Region' is changed

Strictly for an expression afficionado, here is the full underlying AppSheet expression: ***IN([_THIS], SELECT(Regions[Country], [_THISROW].[Lead Region] = [Region]))***

While most app creators will never need to express something this complicated, you could infact provide this expression as a Valid_If constraint. It is useful to know for advanced use cases. For example, instead of using an equality condition, an app creator could use inequality or richer expressions to build very expressive dynamic dropdowns.

Required_If

A 'required' input is one that must be filled in before the record can be saved. A Required_If column constraint is used when a field is 'required' depending on the values of earlier form inputs.

A Required_If constraint is a condition expression that indicates whether a specific column is 'required' in an input form. It is usually based on the values of other columns (e.g. ***[Country Of Birth] = "US"A***).

Editable_If

When an input column is 'editable' it can be modified by the user. The Editable_If column constraint is useful to lock down a certain column (prevent user edits) depending on the value already in the column or on the values of other columns.

An Editable_If constraint is a condition expression that indicates whether a specific column is 'editable' in an input form (e.g. ***ISBLANK([_THIS]***) allows edits if the current value is blank, but once there is a value, it cannot be edited). Another common use case is to give certain users the ability to edit a field but make it readonly for others (eg: ***USEREMAIL() = "manager@acme.co"m***)

Best practices

Column constraints give you the power to define very subtle or complex conditions, but end-users will only see the resulting behavior. As an app creator, it is important to provide adequate explanations for the columns affected by these expressions--particularly for Valid_If conditions, so that users will know how to proceed if they provide an invalid column value. The best way to do so is by providing meaningful column descriptions.

When these expressions reference other fields in the row (not just [_THIS]), it is best to ensure that they are always "backward" references to fields that the end-user has already seen (meaning columns that come *before* the column being considered in the spreadsheet and appear *above* the column being considered in the Column Structure tab). Conditions containing "forward" references may be confusing to end-users and may cause problems with multi-page forms.

Related articles

- *Expressions*
- *Multi-page forms with conditional branching*
- *App formulas*
- *Defining and using slices*
- *Column types*

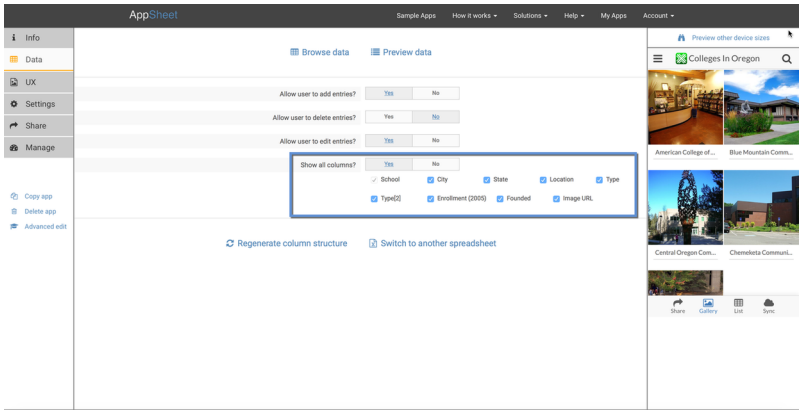
Defining and using slices – AppSheet

You do not need to send all the spreadsheet data to the app. Instead, you can 'slice' the data, choosing a subset of the columns and a subset of the rows.

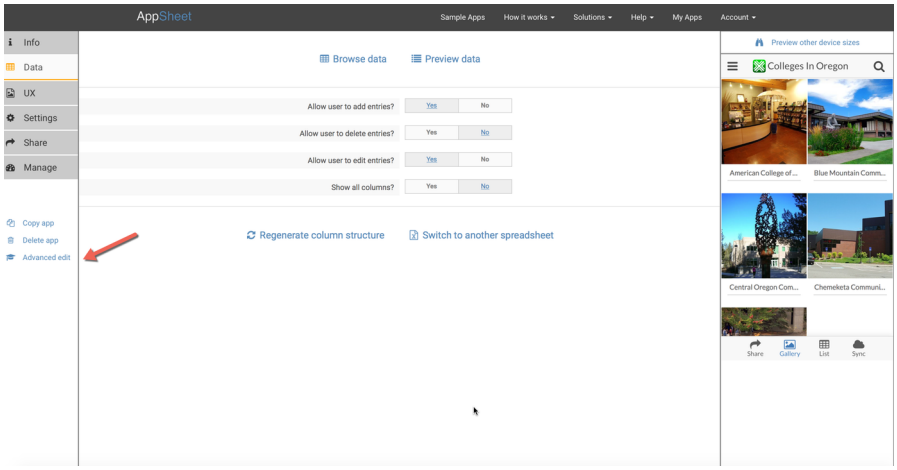
By default, the app utilizes all the columns and all the rows. But this is something you can change.

To slice the columns, pick a subset of the columns that will be included. You can do this in both editors.

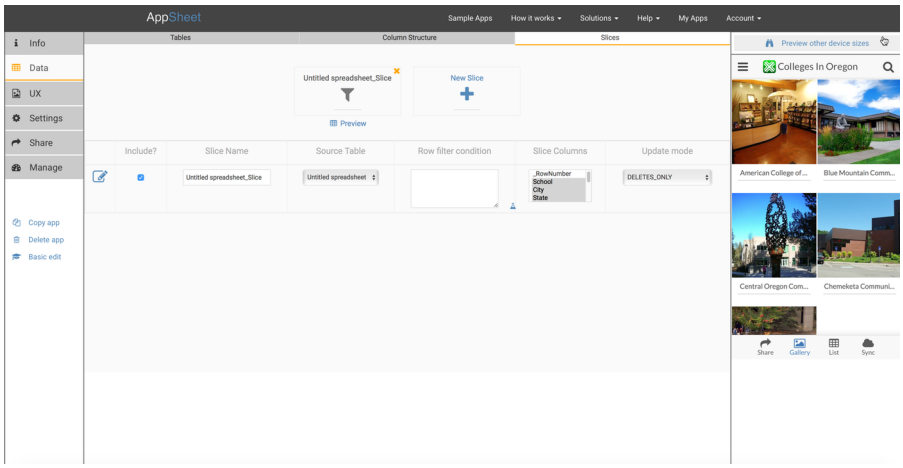
Basic Editor



To slice the rows, go to the **Advanced Editor**.



Advanced Editor In the Advanced Editor, Slices to filter rows and/or columns may be added and edited. See Advanced Editor>Data>Slices.



Rich filter conditions

Rich filter conditions give you the ability to express simple or complex criteria. Most commonly, you want to combine a couple of conditions with an AND or an OR. There are two ways to do this:

1. Insert an *AppSheet expression* in the Slice Row Filter Condition. This permits you to write slices with rich filter conditions. You may use any arbitrary AND/OR/NOT condition to define the row filter for the slice. For example,

to include rows with a DateColumn value in the next 7 days use the following formula: AND([_DateColumn] >= TODAY(), [_DateColumn] < TODAY()+7). To filter on an EmailColumn use this formula: [EmailColumn] = USEREMAIL().

2. Use spreadsheet formulas: Spreadsheets already have a wonderfully expressive formula language. So if you need complex expression logic, consider adding an additional spreadsheet column that is computed via a formula expression. Then use that column in a simple AppSheet slice filter condition. Please be aware that the spreadsheet formulas are only computed when you sync changes to the backend-- i.e. they do not run dynamically in your app.

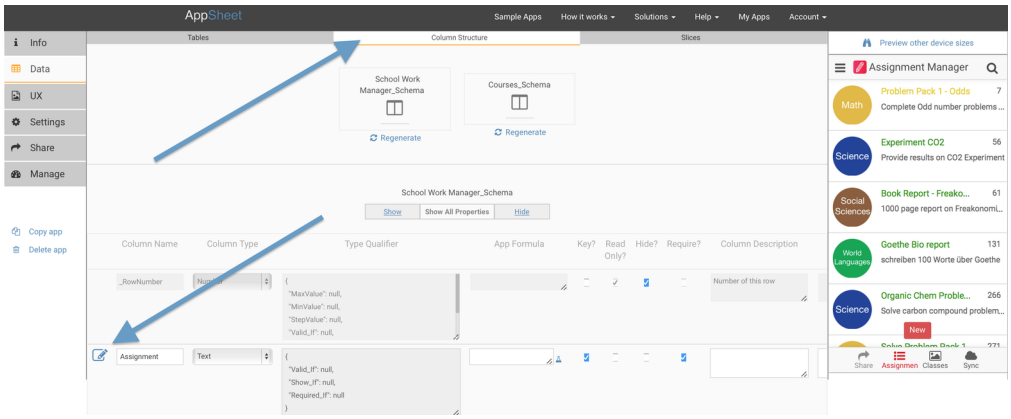
Related articles

- *Slices*
- *Using multiple slices*
- *App formulas*
- *Choosing and adding data views*
- *Column types*

App formulas – AppSheet

Sometimes, you want calculations to run in the app while the user is interacting with the data. To help you achieve that goal, you can specify an App Formula for every column. An App Formula is an expression that AppSheet evaluates whenever a user changes a value in a form (while creating a new record or editing an existing record). Each time such a change occurs, the App Formula of every column in the record is evaluated and the column is assigned the result of the formula evaluation. For example, if a record has a Price column and a Tax column, the Tax column might have an App Formula [Price]*0.05 to compute a 5% tax automatically.

In AppSheet's Advanced Editor, you can add App Formulas to any column when you go to Data>Column Structure.



Click on the edit icon (seen above) and navigate to the App Formula field to enter the formula you need for your app. The formula itself is any valid AppSheet expression that matches the type of the column. You can learn more about *the different kinds of expressions* supported by AppSheet.

App Formulas are distinct from *Spreadsheet Formulas*. The two mechanisms are complementary and both add value to your app. Your spreadsheet is a very powerful tool to run calculations. Those calculations run only when data syncs back to the spreadsheet.

App Formulas are commonly used to calculate the value of a column when the user makes updates to the app. When the changes are synced back to the spreadsheet, the computed value is saved to the corresponding spreadsheet cell.

App Formulas are also used to define *Virtual Columns*. These columns do not actually exist in the spreadsheet but are only computed within the context of the app itself.

Related articles

- *Spreadsheet formulas*

- *Column types*
- *Customizing input forms*
- *How do I control the order of columns displayed in the app?*
- *Keys*

Virtual columns – AppSheet

A virtual column, as the name suggests, is a column of a table in the app that doesn't have an actual column in the underlying spreadsheet. Instead, it is automatically computed via an *app formula expression*.

In order to define a virtual column, use the Advanced Editor>Data>Column Structure, and add a virtual column to any of the column structures. You can specify an appropriate formula expression and the type of the column is automatically detected by AppSheet. Typically, a virtual column utilizes the values of other columns in the same row.

Here are three common use cases for virtual columns:

1. **To combine other columns**: in an app that captures a FirstName and a LastName, a virtual column with the formula **CONCATENATE([LastName], ", ", [FirstName])** can be used to create a FullName.
2. **To construct conditional values**: in an app that captures contact information, a virtual column called PreferredPhoneNumber may be defined with the formula **IF([UseMobilePhone?],[MobilePhoneNumber], [PhoneNumber])**
3. **To construct complex yes/no values**: in an app that captures orders, a virtual column called Important? may be defined with the formula **OR([Amount] > 1000,[Quantity] > 100)**

The ability to create complex yes/no values is really important when they are used in other expressions like slice conditions, column constraints, etc. In short, any time there is a need for a complex condition in an app, it is best to create a virtual column to represent the complex condition, and then use the virtual column wherever needed.

Related articles

- *Expressions*
- *App formulas*
- *Controlling data inputs with column constraints*
- *Defining and using slices*
- *Multi-page forms with conditional branching*

Expressions

Yes/No expressions – AppSheet

Expressions may be used in various AppSheet features -- Initial Values, App Formulas, Virtual Columns and Column Constraints (Valid_If, Show_If, Required_If) -- to customize app behavior and provide your users with advanced functionality. Expressions in this article align with the **Yes/No Expressions** in the **Expression Builder** of the **Advanced Editor.** The Expression Builder can be found anywhere you are able to enter a formula/expression, noted by a little "flask" symbol next to it. Clicking on the flask will bring up the Expression Builder. The Expression Builder is "context-aware," i.e. it shows you expressions that are relevant to the specific table you are editing. Also included in the builder is an "instant" expression checker, to verify that the expression is valid.

Expression components

Use any of the following values as part of an expression:

Constants

- Words, Dates, Times. Highlight all values with **"quotes"**, except for numeric values, e.g. **"Value"**, **"01/01/2016"**, **"12:00:00" for time, "012:00:00" for duration.**
- Numeric values are noted just as they are, e.g **10**.

Column Names

- Name any column using square brackets around the exact column name: **[ColumnName]**. When combining a [ColumnName] with another value, put the expression in **(parentheses)**. May be used in any expression; however, when used in the Initial Value feature, it may only refer to a separate table.

Yes/No conditions

Yes/No Expressions utilize comparison operators that return a True or False result displayed as a Yes or No in AppSheet. These Yes/No expressions are composed of Comparison Operators, Composite Conditions, and Other Conditions. AppSheet conditions are not a 1:1 match with Google Sheets functions; however, in some cases the formatting similarity may help you construct your expressions. Alternatively, if the formatting is not similar, the Google Sheets function page may provide context for use of the function . If available, see the Google Sheets function link in parentheses next to applicable operators.

Comparison operators

AppSheet supports Comparison Conditions by using these comparison operators with two expression parameters that have comparable types. For example, 5 > 2 is valid, but 5 > "Hello" is not valid.

- Equals: = (*EQ*)
- Not Equals: <> (*NE*)
- Greater Than: > (*GT*)
- Greater Than or Equals: >= (*GTE*)
- Less Than: < (*LT*)
- Less Than or Equals: <= (*LTE*)

Composition operators

AppSheet supports Composite Conditions by using composition operators with comparison operators.

- AND({cond_1},..,{cond_n}): returns true if all the sub-expressions are true (*AND*)
- OR({cond_1},..,{cond_n}): returns true if any of the sub-expressions are true (*OR*)
- NOT({cond_1}): returns true if the sub-expression is false and vice-versa (*NOT*)

Other operators

AppSheet supports the following additional operators:

- ISBLANK({*}) returns true if an expression is empty (*ISBLANK*)
- CONTAINS({*},{*}): returns true if value contains
- IN({*},{List}): returns true if a value is in a list

Common and complex expressions

Common expressions

- AND([Color]="Green",[CompleteDate]>TODAY())
- CONTAINS([Fruit],"Oranges")

Complex expressions

- OR(([Price]*[Quantity])>$10,000.00,[Price]>$100.00)
- OR(CONTAINS([Fruit],"Oranges"),CONTAINS([Fruit],"Apples"),CONTAINS([Fruit],"Bananas"))

Yes/No expression patterns and examples

From the **Expression Builder,** follow the pattern below for a True/False, i.e. Yes/No result. See examples for further clarity.

Related articles

- *Math expressions*
- *Expressions*

- *Columns expressions*
- *Other expressions*
- *Time expressions*

Math expressions – AppSheet

Expressions may be used in various AppSheet features -- Initial Values, App Formulas, Virtual Columns and Column Constraints (Valid_If, Show_If, Required_If) -- to customize app behavior and provide your users with advanced functionality. Expressions in this article align with the **Math Expressions** in the **Expression Builder** of the **Advanced Editor**. The Expression Builder can be found anywhere you are able to enter a formula/expression, noted by a little "flask" symbol next to it. Clicking on the flask will bring up the Expression Builder. The Expression Builder is "context-aware," i.e. it shows you expressions that are relevant to the specific table you are editing. Also included in the builder is an "instant" expression checker, to verify that the expression is valid.

Expression c omponents

Use any of the following values as part of an expression:

Constants

- Words, Dates, Times. Highlight all values with **"quotes"**, except for numeric values, e.g. **"Value"**, **"01/01/2016", "12:00:00" for time, "012:00:00" for duration.**
- Numeric values are noted just as they are, e.g **10**.

Column Names

- Name any column using square brackets around the exact column name: **[ColumnName]**. When combining a [ColumnName] with another value, put the expression in **(parentheses)**. May be used in any expression; however, when used in the Initial Value feature, it may only refer to a separate table.

Math conditions

Math Expressions utilize arithmetic operators to return a numeric value. All operators, except Unary Minus, are used with two numeric expression parameters, e.g. (3+2), (5.0/2.3), etc. AppSheet conditions are not a 1:1 match with Google Sheets functions; however, in some cases the formatting similarity may help you construct your expressions. Alternatively, if the formatting is not similar, the Google Sheets function page may provide context for use of the function . If available, see the Google Sheets function link in parentheses next to applicable operators.

Math operators

Use any of the arithmetic operators below to build arithmetic expressions:

- Add: + (*ADD*)
- Subtract: - (*MINUS*)
- Multiply: * (*MULTIPLY*)
- Divide: / (*DIVIDE*)
- Mod: % (*MOD*)
- Unary Minus: - (*UMINUS*)

Common and complex expressions

Common expressions

- (1+[Number])
- ([Decimal]-4.5)
- ([Price]*[Quantity])
- 10/2

Complex expressions

- ([Number]*2) + ([Number]*3)

Math expression patterns and examples

From the **Expression Builder**, follow the pattern below for a numeric result. See examples for further clarity.

Pattern	Example
({value_1} + {value_2})	([Decimal] + 1.12)
({value_1} - {value_2})	([Percent] - 0.2)
({value_1} * {value_2})	([Price] * 19.99)
({value_1} / {value_2})	([Number] / 1)
({value_1} % {value_2})	([Percent] % 0.2)
(- {value_1})	(- [Decimal])

Related articles

- *Yes/No expressions*
- *Time expressions*
- *Expressions*
- *Columns expressions*
- *Lists expressions and Aggregates*

Time expressions – AppSheet

Expressions may be used in various AppSheet features -- Initial Values, App Formulas, Virtual Columns and Column Constraints (Valid_If, Show_If, Required_If) -- to customize app behavior and provide your users with advanced functionality. Expressions in this article align with the **Time Expressions** in the **Expression Builder** of the **Advanced Editor**. The Expression Builder can be found anywhere you are able to enter a formula/expression, noted by a little "flask" symbol next to it. Clicking on the flask will bring up the Expression Builder. The Expression Builder is "context-aware," i.e. it shows you expressions that are relevant to the specific table you are editing. Also included in the builder is an "instant" expression checker, to verify that the expression is valid.

Expression c omponents

Use any of the following values as part of an expression:

Constants

- Words, Dates, Times. Highlight all values with **"quotes"**, except for numeric values, e.g. **"Value"**, **"01/01/2016"**, **"12:00:00" for time, "012:00:00" for duration.**
- Numeric values are noted just as they are, e.g **10**.

Column Names

- Name any column using square brackets around the exact column name: **[ColumnName]**. When combining a [ColumnName] with another value, put the expression in **(parentheses)**. May be used in any expression; however, when used in the Initial Value feature, it may only refer to a separate table.

Time conditions

Time Expressions utilize Dates or Durations to return a DateTime, Date, Time, Duration or Number value. AppSheet conditions are not a 1:1 match with Google Sheets functions; however, in some cases the formatting similarity may help you construct your expressions. Alternatively, if the formatting is not similar, the Google Sheets function page may provide context for use of the function . If available, see the Google Sheets function link in parentheses next to applicable operators.

Current operators

- NOW() for the current DateTime (*NOW*)

- TODAY() for the current Date (*TODAY*)
- TIMENOW() for the current Time (*TIME*)
- HOUR({Duration}) for the Hour component of a Specific Time (*HOUR*)
- MINUTE({Duration}) for the Minute component of a Specific Time (*MINUTE*)
- SECOND({Duration}) for the Second component of a Specific Time (*SECOND*)
- DAY({Date}) for the Day of the Month (*DAY*)
- MONTH({Date}) for the Month Number from a Date (*MONTH*)
- YEAR({Date}) for the Year from a Date (*YEAR*)
- WEEKDAY({Date}) for the Day Number from a Date (*WEEKDAY*)
- WEEKNUM({Date}) for the Week Number from a Date (*WEEKNUM*)

Legacy operators

For backwards compatibility, we also support the function syntax below for a set of functions that have been supported from the earliest AppSheet release.

- @(_NOW) for the current DateTime (*NOW*)
- @(_TODAY) for the current Date (*TODAY*)
- @(_TIMENOW) for the current Time (*TIME*)

Common and complex expressions

Common expressions

- TODAY() + 1: adds 1 day to the current Date
- NOW() - 1: subtracts 1 day from the current DateTime
- TODAY() - "12/12/2001": returns a Duration between the two dates
- TIMENOW() + "03:03:00": adds 3 hours and 3 minutes to the current Time

Complex expressions

- @(_TODAY)>([TargetDate]+1)

Time expression patterns, results and examples

From the **Expression Builder**, follow the pattern below for a DateTime, Date, Time, Duration or Number result. See the results and examples for further clarity.

Pattern	**Result**	**Example**
NOW()	DateTime	NOW()
TODAY()	Date	TODAY()
TIMENOW()	Time	TIMENOW()
HOUR({Duration})	Number	HOUR([Duration])
MINUTE({Duration})	Number	MINUTE([Duration])
SECOND({Duration})	Number	SECOND([Duration])
DAY({Date})	Number	DAY([Date])
MONTH({Date})	Number	MONTH([Date])
YEAR({Date})	Number	YEAR([Date])
WEEKDAY({Date})	Number	WEEKDAY([Date])
WEEKNUM({Date})	Number	WEEKNUM([Date])
{value_1} + {number}	Date	[Date] + 1

{value_1} - {value_2}	Duration	[Date] - (TODAY() + 1)
{value_1} - {duration}	Date	[Date] - "002:00:00"
{value_1} - {number}	DateTime	[DateTime] - 1
{value_1} - {value_2}	Duration	[DateTime] - (NOW() + 1)
{value_1} + {duration}	DateTime	[DateTime] + "002:00:00"
{value_1} + {number}	Duration	[Duration] + 1
{value_1} - {value_2}	Duration	[Duration] - (0:00:00 + 1)
{value_1} - {duration}	Duration	[Duration] - "002:00:00"
{value_1} - {number}	Time	[Time] - 1
{value_1} - {value_2}	Duration	[Time] - (TIMENOW() + 1)
{value_1} + {duration}	Time	[Time] + "002:00:00"

Related articles

- *Columns expressions*
- *Expressions*
- *Math expressions*
- *Lists expressions and Aggregates*
- *Yes/No expressions*

Columns expressions – AppSheet

Expressions may be used in various AppSheet features -- Initial Values, App Formulas, Virtual Columns and Column Constraints (Valid_If, Show_If, Required_If) -- to customize app behavior and provide your users with advanced functionality. Expressions in this article align with the **Columns Expressions** in the **Expression Builder** of the **Advanced Editor.** The Expression Builder can be found anywhere you are able to enter a formula/expression, noted by a little "flask" symbol next to it. Clicking on the flask will bring up the Expression Builder. The Expression Builder is "context-aware," i.e. it shows you expressions that are relevant to the specific table you are editing. Also included in the builder is an "instant" expression checker, to verify that the expression is valid.

Expression c omponents

Use any of the following values as part of an expression:

Constants

- Words, Dates, Times. Highlight all values with **"quotes"**, except for numeric values, e.g. **"Value"**, **"01/01/2016", "12:00:00" for time, "012:00:00" for duration.**
- Numeric values are noted just as they are, e.g **10**.

Columns conditions

Columns Expressions refer to a column to return a value.

Columns operators

- Use **[_THIS]** to refer to the value of the current column (used in Valid_If, Show_If, and Required_If conditions). See *Column Constraints*.
- Name any column using square brackets around the exact column name: **[ColumnName]**. When combining a [ColumnName] with another value, put the expression in **(parentheses)**. May be used in any expression; however, when used in the Initial Value feature, it may only refer to a separate table.

Common and complex expressions

Common expressions

- ([_THIS]>25)

Complex expressions

- Pending

Columns expression patterns and examples

From the **Expression Builder**, follow the pattern below for a result. See examples for further clarity.

Pattern	Result	Example
[_THIS]	Text	[_THIS]
[{column_name}]	Number	[Project #]

Related articles

- *Lists expressions and Aggregates*
- *Controlling data inputs with column constraints*
- *Expressions*
- *Yes/No expressions*
- *Time expressions*

Lists expressions and Aggregates – AppSheet

Expressions may be used in various AppSheet features -- Initial Values, App Formulas, Virtual Columns and Column Constraints (Valid_If, Show_If, Required_If) -- to customize app behavior and provide your users with advanced functionality. Expressions in this article align with the **Lists Expressions** in the **Expression Builder** of the **Advanced Editor**. The Expression Builder can be found anywhere you are able to enter a formula/expression, noted by a little "flask" symbol next to it. Clicking on the flask will bring up the Expression Builder. The Expression Builder is "context-aware," i.e. it shows you expressions that are relevant to the specific table you are editing. Also included in the builder is an "instant" expression checker, to verify that the expression is valid.

Expression components

Use any of the following values as part of an expression:

Constants

- Words, Dates, Times. Highlight all values with "quotes", except for numeric values, e.g. **"Value"**, **"01/01/2016", "12:00:00" for time, "012:00:00" for duration.**
- Numeric values are noted just as they are, e.g **10**.

Column Names

- Name any column using square brackets around the exact column name: **[ColumnName]**. When combining a [ColumnName] with another value, put the expression in **(parentheses)**. May be used in any expression; however, when used in the Initial Value feature, it may only refer to a separate table.

Lists conditions

Lists Expressions utilize operators that return a list or numeric value. AppSheet conditions are not a 1:1 match with Google Sheets functions; however, in some cases the formatting similarity may help you construct your expressions. Alternatively, if the formatting is not similar, the Google Sheets function page may provide context for use of the function. If available, see the Google Sheets function link in parentheses next to applicable operators.

Lists operators

- LIST({*},{*})
- SELECT({List}),{Yes/No})
- COUNT({List}) (*COUNT*)
- SUM({List}) (*SUM*)
- MIN({List}) (*MIN*)
- MAX({List}) (*MAX*)
- AVERAGE({List}) (*AVERAGE*)
- ANY({List})

Context

In addition to basic column types, we have a meta-type List_Of_X (eg: List Of Number, List Of Enum, etc) that represents a (potentially empty) list of unique values. This becomes really important for more powerful expressions.

There are functions to construct lists and functions that use lists:

Construct a list:

- The simplest way to construct such a list is by explicitly writing it {1, 2, 3} or {[ColumnA], [ColumnB]}
- The **LIST** function is syntactically equivalent to an explicit list --- **LIST**(1,2,3) is the same as {1, 2, 3}
- A table column list âÄì the values are constructed from the unique values in a specific column of a specific table. Eg: Customers[Phone Number] returns a list of unique phone numbers in the Customers table
- **SELECT** is a more powerful way to construct a list from another table. It is a stylized SQL select-from-where query. It returns a single list of values from one column of a table. However, a filter can be applied to eliminate some of the rows.
- SELECT(Customers[Phone Number], [State] = WA) --- returns a list of phone numbers of WA customers

Use a list (however it is constructed):

The most common use case is to check if a value is in a list:

- IN([ColumnName], {1, 2, 3})
- IN([_THIS], {1, 2, 3}) --- a special case used for column constraints like Valid_If and Show_If
- For column constraints, we accept a short form of the previous expression which is just the listâÄ¦ so {1, 2, 3} is treated as IN([_THIS], {1, 2, 3})

Aggregate functions

- /**COUNT**(<list of anything>)/
- **SUM**(<list of numeric type>)
- **MIN**(<list of numeric type>)
- **MAX**(<list of numeric type>)
- **AVERAGE**(<list of numeric type>)
- **ANY**(<list of anything>)

The behavior of COUNT(), SUM(), MIN(), MAX(), and AVERAGE() are self-evident. ANY() picks a single value from a list of values. For example, if there is a table called Customers, each of whom has a Name and a Phone Number, the expression ANY(SELECT(Customers[Phone], [Name] = "John Doe")) gets the phone number of a specific customer.

In many usage scenarios, a **SELECT** expression is used in the context of a particular column (eg: in a Valid_If or a Show_If) in a particular row. In these contexts, the condition used in the **SELECT** function can utilize not just the columns of the table (Customer) but also the column values from the context in which it is used. **[_THIS]** refers to the cell/column from the context and **[_THISROW]** refers to the row from the context. See *Column Constraints*.

- A sample Valid_If condition for a column that accepts a State: COUNT(SELECT(Customers[Phone Number], [State] = [_THIS])) > 100 --- this says the State is valid only if the number of customers in that state > 100

- A sample Show_If condition for a subsequent column in a form that first asks for a State: COUNT(SELECT(Customers[Phone Number], [State] = [_THISROW].[State])) > 1000 --- ask this question only for states with a lot of customers

Common and complex expressions

Common expressions

- Customers[Phone Number]: returns a list of unique phone numbers in the Customers table
- LIST(1,2,3): returns 1,2,3
- SELECT(Customers[Phone Number], [State] = WA): returns a list of phone numbers of WA customers
- COUNT({Dogs,Cats,Birds}): returns the number 3.
- COUNT({3,4,9,15,32}): returns the number 5.
- SUM({3,4,9,15,32}): returns the number 63.
- MIN({3,4,9,15,32}): returns the number 3.
- MAX({3.1,4.2,9.3,15.4,32.5}): returns the number 32.5
- AVERAGE({1,2,3,4}): returns the number 2.5

Complex expressions

- for Valid_If: COUNT(SELECT(Customers[Phone Number], [State] = [_THIS])) > 100: the State is valid only if the number of customers in that state > 100
- for Show_If: COUNT(SELECT(Customers[Phone Number], [State] = [_THISROW].[State])) > 1000: ask this question only for states with a lot of customers

Lists expression patterns and examples

From the **Expression Builder,** follow the pattern below for a list or numeric result. See examples for further clarity.

Pattern	Result	Example
table_name[{column_name}]	List	Field Types[Address]
LIST({*},{*})	List	LIST([ChangeTimestamp],"value_1")
SELECT({List},{Yes/No})	List	SELECT({"value_1", "value_2"}, [Price] = 19.99)
COUNT({List})	Number	COUNT({"value_1", "value_2"})
SUM({List})	Number	SUM({"value_1", "value_2"})
ANY({List})	Number	ANY({"value_1", "value_2"})

Related articles

- *Other expressions*
- *Expressions*
- *Columns expressions*
- *Controlling data inputs with column constraints*
- *Column types*

Other expressions – AppSheet

Expressions may be used in various AppSheet features -- Initial Values, App Formulas, Virtual Columns and Column Constraints (Valid_If, Show_If, Required_If) -- to customize app behavior and provide your users with advanced functionality. Expressions in this article align with the **Other Expressions** in the **Expression Builder** of the **Advanced Editor**. The Expression Builder can be found anywhere you are able to enter a formula/expression, noted by a little "flask" symbol next to it. Clicking on the flask will bring up the Expression Builder. The Expression

Builder is "context-aware," i.e. it shows you expressions that are relevant to the specific table you are editing. Also included in the builder is an "instant" expression checker, to verify that the expression is valid.

Expression c omponents

Use any of the following values as part of an expression:

Constants

- Words, Dates, Times. Highlight all values with **"quotes"**, except for numeric values, e.g. **"Value"**, **"01/01/2016", "12:00:00" for time, "012:00:00" for duration.**
- Numeric values are noted just as they are, e.g **10**.

Column Names

- Name any column using square brackets around the exact column name: **[ColumnName]**. When combining a [ColumnName] with another value, put the expression in **(parentheses)**. May be used in any expression; however, when used in the Initial Value feature, it may only refer to a separate table.

Other conditions

Other expressions impact a range of scenarios and don't fit into any of the previous Yes/No, Math, Time, Columns, or Lists expression categories. AppSheet conditions are not a 1:1 match with Google Sheets functions; however, in some cases the formatting similarity may help you construct your expressions. Alternatively, if the formatting is not similar, the Google Sheets function page may provide context for use of the function . If available, see the Google Sheets function link in parentheses next to applicable operators.

Current operators

- LEN(<text-expression>) to get the length of a text value (*LEN*)
- CONCATENATE(<text-expression>, <text-expression2>, ...) to combine two or more text values (*CONCATENATE*)
- LEFT({Text},{Number}) Returns a substring from the beginning of a specified string. (*LEFT*)
- RIGHT({Text},{Number}) Returns a substring from the end of a specified string. (*RIGHT*)
- FIND({Text},{Text}) Returns the position at which a string is first found within text, case-sensitive. (*FIND*)
- IF({condition},{then-expression},{else-expression}) (*IF*) The last two values (then, else) of the expression, must be of the same type, i.e. text, number, etc.
- UNIQUEID() to create unique Text values for Keys
- HERE() for LatLong of the current user
- USEREMAIL() for the Email of the current user
- USERNAME() for the Name of the current user

Legacy operators

For backwards compatibility, we also support the function syntax below for a set of functions that have been supported from the earliest AppSheet release.

- @(_UNIQUE) to create unique Text values for Keys
- @(_HERE) for LatLong of the current user
- @(_USEREMAIL) for the Email of the current user
- @(_USERNAME) for the Name of the current user

Common and complex expressions

Common expressions

- LEN("AppSheet"): Returns 8.

- CONCATENATE([First Name]," ",[Last Name]): Returns a Full Name.
- IF([Status]="Open"."Green","Red"): Returns "Green" when Status equals Open; otherwise, returns "Red".

Example: Column called [AppName] with a value of "Sales-10305"

- LEFT[AppName, 5]: Returns "Sales".
- RIGHT[AppName, 5]: Returns "10305"
- LEFT([AppName], FIND("-",[AppName])) gives you "Sales"

Use the following expressions in the Initial Value feature of the **Advanced Editor**:

- UNIQUEID(): Use to generate a unique Text value, e.g. a unique Invoice ID.
- HERE(): Use to identify the user's current LatLong.
- USEREMAIL(): Use to populate record value based on user login
- USERNAME(): This value is not reliably populated by cloud providers, please defer to USEREMAIL(). To obtain USERNAME() in this case, utilize a reference table based on the USEREMAIL(). See *References between tables*.

Complex expressions

- LEN([_THIS])<=10: Use this expression in the Valid_If Constraint to restrict form field input to a maximum of 10 characters.
- IF([Status]="Open","Green",IF([Status]="Closed","Red",IF([Status]="Not Started","Blue","Purple"))): Returns "Green" when Status equals Open; returns "Red" when Status equals Closed; returns "Blue" when status equals Not Started; otherwise, returns "Purple".

Other expression patterns and examples

From the **Expression Builder**, follow the pattern below for a null, numeric, text, email, name, or LatLong result. See examples for further clarity.

Pattern	Result	Example
[{ref_column}].[{lookup_column}]	null	[Ref].[Ref]
LEN({*})	Number	LEN([ChangeTimestamp])
CONCATENATE({*},{*})	Text	CONCATENATE([ChangeTimestamp],"value_1")
LEFT({Text},{Number})	Text	LEFT([Text],1)
RIGHT({Text},{Number})	Text	RIGHT([Text],1)
FIND({Text},{Text})	Number	FIND([Text],"text value")
IF({Yes/No},{*},{*})	*	IF([Yes/No],"value_1","value_2")
USERNAME()	Name	USERNAME()
USEREMAIL()	Email	USEREMAIL()
UNIQUEID()	Text	UNIQUEID()
HERE()	LatLong	HERE()

Related articles

- *Lists expressions and Aggregates*
- *Expressions*
- *Time expressions*
- *Required_If*
- *Yes/No expressions*

Controlling Data Inputs (Column Constraints)

Required_If – AppSheet

Expression components

These expression components utilize the following formatting:

- Name any column using square brackets around the exact column name: [ColumnName]. When combining a [ColumnName] with another value, put the expression in (parentheses).
- Use [_THIS] as a "virtual column name." It refers to the value of the current column (used in Valid_If, Show_If, Required_If conditions).
- Use [_THISROW] as a "virtual reference column." It refers to the current row. For example, use [_THISROW].[ColumnName].
- Highlight all values with "quotes," except for numeric values: "Value".
- Numeric values are noted just as they are, e.g 10.

Required_If

A 'required' input is one that must be filled in before the record can be saved. A Required_If column constraint is used when a field is 'required' depending on the values of earlier form inputs.

A Required_If constraint is a condition expression that indicates whether a specific column is 'required' in an input form. It is usually based on the values of other columns (e.g. *[Country Of Birth] = "US"A*).

Related articles

- *Show_If*
- *Valid_If and Dependent Dropdowns*
- *Controlling data inputs with column constraints*
- *Other expressions*
- *AppSheet is NOT...*

Show_If – AppSheet

Expression components

These expression components utilize the following formatting:

- Name any column using square brackets around the exact column name: [ColumnName]. When combining a [ColumnName] with another value, put the expression in (parentheses).
- Use [_THIS] as a "virtual column name." It refers to the value of the current column (used in Valid_If, Show_If, Required_If conditions).
- Use [_THISROW] as a "virtual reference column." It refers to the current row. For example, use [_THISROW].[ColumnName].
- Highlight all values with "quotes," except for numeric values: "Value".
- Numeric values are noted just as they are, e.g 10.

Show_If

A Show_If column constraint is used when an input field should be shown or hidden depending on the values of one or more earlier field values in the form.

A Show_If constraint is a condition expression that determines whether or not an input for this column should be shown in a form. This is usually based on the values of other columns. For example, the condition expression

[UserRating] = ""5 will display this column if the value in the "UserRating" column is "5". The condition expression *[Status] = "Gree"n* will display this column if the value in the "Status" column is "Green".

The special column name "_THIS" is used to refer to the current column being constrained. For example, a Show_If condition of *ISBLANK([_THIS])* can be used to show an input field only if the column itself is blank. The column will be hidden, once a data value has been entered and saved for the column.

Show_If conditions can also be defined on Page Header columns to *conditionally show or hide entire pages*.

Related articles

- *Multi-page forms with conditional branching*
- *Valid_If*
- *Other expressions*
- *Presentation types*
- *Controlling data inputs with column constraints*

Valid_If – AppSheet

Expression components

These expression components utilize the following formatting:

- Name any column using square brackets around the exact column name: [ColumnName]. When combining a [ColumnName] with another value, put the expression in (parentheses).
- Use [_THIS] as a "virtual column name." It refers to the value of the current column (used in Valid_If, Show_If, Required_If conditions).
- Use [_THISROW] as a "virtual reference column." It refers to the current row. For example, use [_THISROW].[ColumnName].
- Highlight all values with "quotes," except for numeric values: "Value".
- Numeric values are noted just as they are, e.g 10.

Valid_If

Every input in a form is checked for validity based on its type. For example, a column of Number type will not accept "Hello" as an input. A Valid_If column constraint is used in situations where the validity of the input requires richer data-dependent logic.

A Valid_If constraint is a condition expression that determines whether or not the form input for this column is valid. For example, the *Quote Calculator sample* utilizes a Valid_If condition to ensure that the Cost Per Hour must be less than $20.

Here are examples of commonly used Valid_If constraints:

1. Does the value of the column satisfy a simple condition?
   - For example, comparing the value with a constant: *[_THIS] > 5*
   - For example, comparing the value with another column: *[_THIS] > [ColumnA]*
2. Is the value of the column in a list?
   - A list of constant values. For example: *{100, 200, 300}*
   - A list of values from a column in another table. For example: *LookupTable[ColumnC]* . This is particularly useful if the list of allowed values should itself be allowed to change while the app is being used.
   - A list of values from specific rows in another table. For example: *SELECT(LookupTable[ColumnC], [ReportDate] > Today() - 7)* specifies that the rows in LookupTable should be filtered to find those where ReportDate is within the last week, and the corresponding values in ColumnC of LookupTable become the allowed list of valid values.

Whenever a list of allowed values is provided, it is actually a syntactic shortform for an expression that uses the IN function. For example, *{100, 200, 300}* is the same as *IN([_THIS], {100, 200, 300})*. In input forms in the mobile app, columns with such Valid_If constraints are provided with dropdowns or enumeration selectors that reflect the list of choices.

Related articles

- *Valid_If and Dependent Dropdowns*
- *Controlling data inputs with column constraints*
- *Other expressions*
- *Show_If*
- *Required_If*

Valid_If and Dependent Dropdowns – AppSheet

Expression components

These expression components utilize the following formatting:

- Name any column using square brackets around the exact column name: [ColumnName]. When combining a [ColumnName] with another value, put the expression in (parentheses).
- Use [_THIS] as a "virtual column name." It refers to the value of the current column (used in Valid_If, Show_If, Required_If conditions).
- Use [_THISROW] as a "virtual reference column." It refers to the current row. For example, use [_THISROW].[ColumnName].
- Highlight all values with "quotes," except for numeric values: "Value".
- Numeric values are noted just as they are, e.g 10.

Valid_If

Every input in a form is checked for validity based on its type. For example, a column of Number type will not accept "Hello" as an input. A Valid_If column constraint is used in situations where the validity of the input requires richer data-dependent logic.

A Valid_If constraint is a condition expression that determines whether or not the form input for this column is valid. For example, the *Quote Calculator sample* utilizes a Valid_If condition to ensure that the Cost Per Hour must be less than $20.

Here are examples of commonly used Valid_If constraints:

1. Does the value of the column satisfy a simple condition?
   - For example, comparing the value with a constant: *[_THIS] > 5*
   - For example, comparing the value with another column: *[_THIS] > [ColumnA]*
2. Is the value of the column in a list?
   - A list of constant values. For example: *{100, 200, 300}*
   - A list of values from a column in another table. For example: *LookupTable[ColumnC]* . This is particularly useful if the list of allowed values should itself be allowed to change while the app is being used.
   - A list of values from specific rows in another table. For example: *SELECT(LookupTable[ColumnC], [ReportDate] > Today() - 7)* specifies that the rows in LookupTable should be filtered to find those where ReportDate is within the last week, and the corresponding values in ColumnC of LookupTable become the allowed list of valid values.

Whenever a list of allowed values is provided, it is actually a syntactic shortform for an expression that uses the IN function. For example, *{100, 200, 300}* is the same as *IN([_THIS], {100, 200, 300})*. In input forms in the mobile app,

columns with such Valid_If constraints are provided with dropdowns or enumeration selectors that reflect the list of choices.

Valid_If and Dependent Dropdowns

Dependent dropdowns are a common design pattern in apps that capture input. For example, consider an app like *this Lead Tracker sample* that asks for the 'Lead Region' (America, Asia, Europe) and then for a 'Country' within that region. This is actually requires relatively complex logic, but AppSheet tries to make it simple. Dependent dropdowns are driven by a separate lookup table.

In the sample, there is a separate 'Regions' lookup table with two columns: 'Region' and 'Country'. This acts as the lookup table for allowed combinations of regions and countries. *Here is the table data* used in the sample.

The 'Lead Region' column has a regular Valid_If constraint: *Regions[Region]*. Therefore, when a new entry is being added, the input for this column shows three choices: America, Asia, and Europe.

Likewise, the 'Country' column also specifies a similar Valid_If constraint: *Regions[Country]*. However, because it follows the 'Lead Region' column and because both specify columns from the same lookup table 'Regions', AppSheet recognizes the intent and implements a dependent dropdown.

Internally, AppSheet creates an expression to capture the allowed set of values for the 'Country' column. The expression must say (in English!):

- Look at the Regions table
- Filter the rows to make sure that the Region column of the table matches the value in the 'Lead Region' column of the row being edited in the form
- Now extract the 'Country' column from those filtered rows
- Eliminate any duplicates --- these are the allowed countries!
- Recompute this list each time the 'Lead Region' is changed

Strictly for an expression afficionado, here is the full underlying AppSheet expression: *IN([_THIS], SELECT(Regions[Country], [_THISROW].[Lead Region] = [Region]))*

While most app creators will never need to express something this complicated, you could infact provide this expression as a Valid_If constraint. It is useful to know for advanced use cases. For example, instead of using an equality condition, an app creator could use inequality or richer expressions to build very expressive dynamic dropdowns.

Related articles

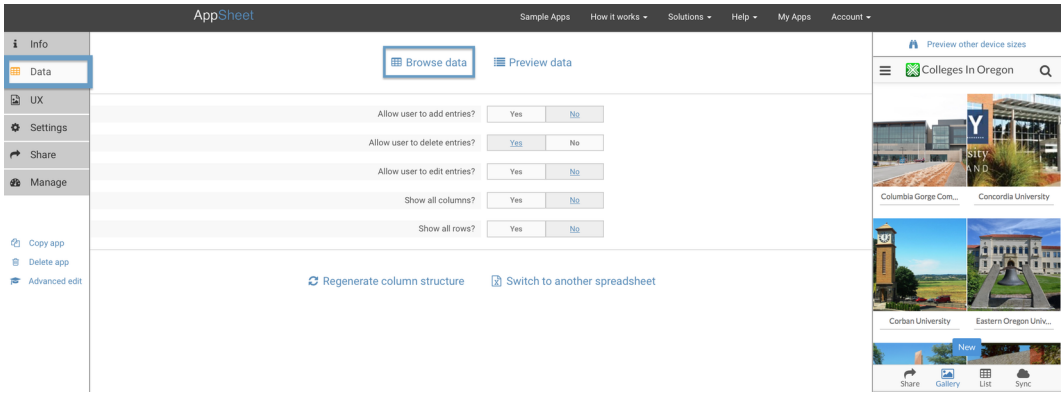
- *Lists expressions and Aggregates*
- *Controlling data inputs with column constraints*
- *Valid_If*
- *Show_If*
- *Columns expressions*

Tutorial Content

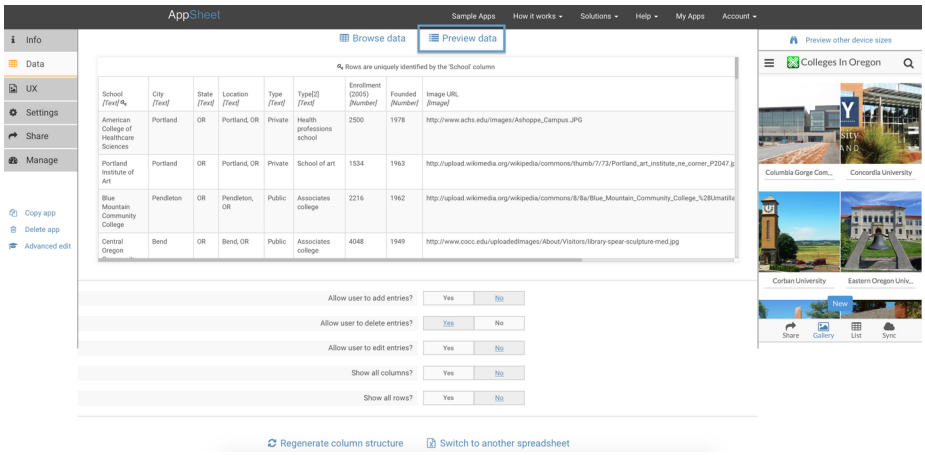
Using the app editor and a spreadsheet – AppSheet

AppSheet extracts tabular column structure from your spreadsheet, and works best when the spreadsheet has regular tabular data-- the first row has column headers and the remaining rows are data. Extraneous data, drawings, charts, etc. may cause problems. Here's how you understand and control the process.

In the **Data** section of the Basic Editor, you see a few simple controls. The **Browse Data** link opens the spreadsheet in a separate browser tab. If you are using Google Sheets, this allows you to edit the Google sheet in one tab while you see the effects on your app in the AppSheet tab.

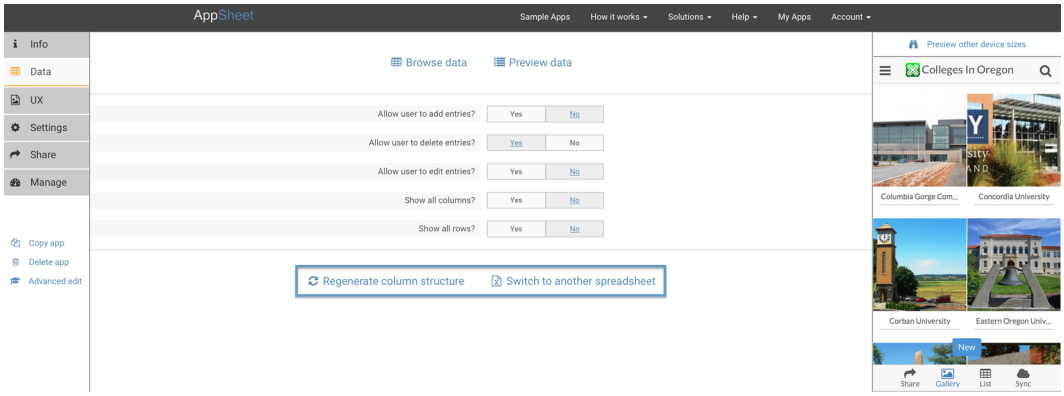


If you click on the **Preview Data** link, you see the first few rows of the spreadsheet conforming to the column structure that AppSheet has extracted. Each column has a header name and a data type. Also, one of the columns has been identified as the *unique key*. *This choice* is very important-- the unique key helps AppSheet synchronize changes made in the app back to the spreadsheet.



Every time you refresh the Basic Editor page, AppSheet checks your app and shows error, warning, and info messages at the top of the page. Wherever possible, AppSheet tries to automatically fix problems found and inform you.

When you change the structure of the spreadsheet, AppSheet notices the difference between the spreadsheet and what it expects. So it gives you a warning: 'More columns than expected' or 'Less columns than expected'. If you are editing the app on our web page, it will try to automatically pick up the new structure. If the extracted column structure is not what you expect, or there are errors/warnings you want to act on, you would typically make changes to your spreadsheet (as described below) and then click **Regenerate Column Structure**.



You can also choose to **switch to another spreadsheet**. There are two reasons to do this-- perhaps you started with an example to play with and now you want to switch to a 'real' spreadsheet with a very different structure, or perhaps you want to switch to a different spreadsheet that has the identical structure (eg: switching from test data to production data). In both cases, AppSheet detects the new column structure and reacts appropriately to it.

Related articles

- *Create an app*
- *How should I choose a key?*
- *Column types*
- *Modifying column structure in the editor*
- *Controlling data updates*

How should I choose a key? – AppSheet

In this article, we'll explain:

- what a key is
- why it matters
- how keys are chosen and how you can influence the choice
- how keys are displayed in your app
- why RowNumber is bad for a key
- how to generate unique key values

The example of an auto repair shop will be used to expand on these topics.

- **What is a key and why does it matter?**

A key **uniquely** identifies each individual row in the table. For example, for a product catalog, the Product ID or Product Name are good keys since each of those items will likely never repeat twice. Likewise, a First Name column wouldn't be a good key since first names are likely to repeat more than once.

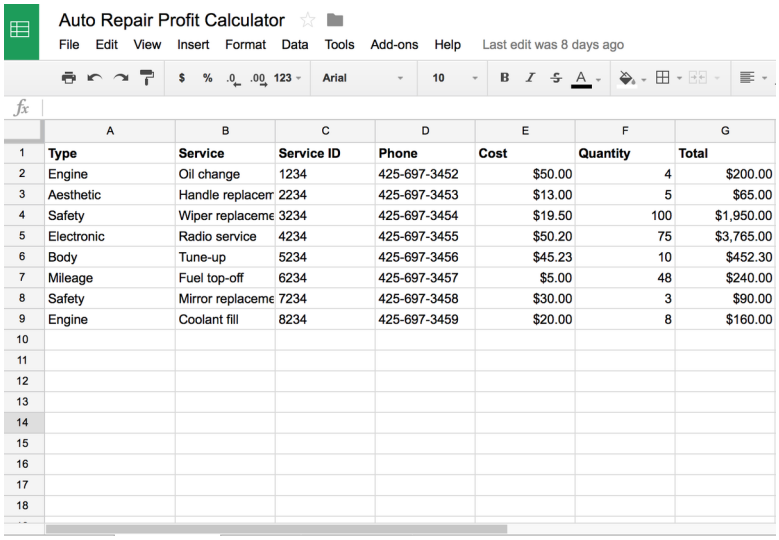
The key may be a single column (such as Product ID) or a composite column (such as FirstName and LastName). Many data sets naturally have a key.

- **How keys are chosen**

When you create an app from your spreadsheet, AppSheet intuitively assigns a key by searching your data for columns without duplicate values. If AppSheet can't find a suitable key, it will combine two columns to create a Computed key. If it *still* can't find a good key by combining two columns, row number will become the key; but row number as a key is generally not a good idea-- we'll talk more about that later in this section.

It's a good practice to make your intended key the leftmost column in your spreadsheet, if possible, since AppSheet reads the data from left to right. That being said, AppSheet will still scan the data and assign a key it thinks is best.

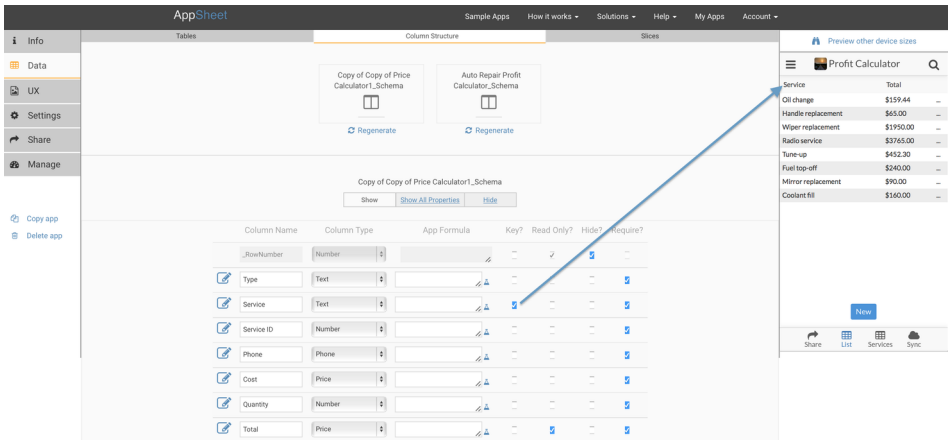
Let's say I work for an auto repair shop and I'm keeping track of our weekly services rendered and the total resulting profit. In my spreadsheet the *type* of service is the leftmost column.



	A	B	C	D	E	F	G
1	**Type**	**Service**	**Service ID**	**Phone**	**Cost**	**Quantity**	**Total**
2	Engine	Oil change	1234	425-697-3452	$50.00	4	$200.00
3	Aesthetic	Handle replacem	2234	425-697-3453	$13.00	5	$65.00
4	Safety	Wiper replaceme	3234	425-697-3454	$19.50	100	$1,950.00
5	Electronic	Radio service	4234	425-697-3455	$50.20	75	$3,765.00
6	Body	Tune-up	5234	425-697-3456	$45.23	10	$452.30
7	Mileage	Fuel top-off	6234	425-697-3457	$5.00	48	$240.00
8	Safety	Mirror replaceme	7234	425-697-3458	$30.00	3	$90.00
9	Engine	Coolant fill	8234	425-697-3459	$20.00	8	$160.00
10							
11							
12							
13							
14							
15							
16							
17							
18							

- **How keys are displayed**

Although Type is my leftmost column, once I generated a mobile app from the data (using the Google Sheets add-on) AppSheet chose Service as the key, because there is repeating data in the Type column and unique data in the Service column. AppSheet assumes the column with the key is the most important and therefore displays it first, *according to Rule 1 of how columns are displayed*.

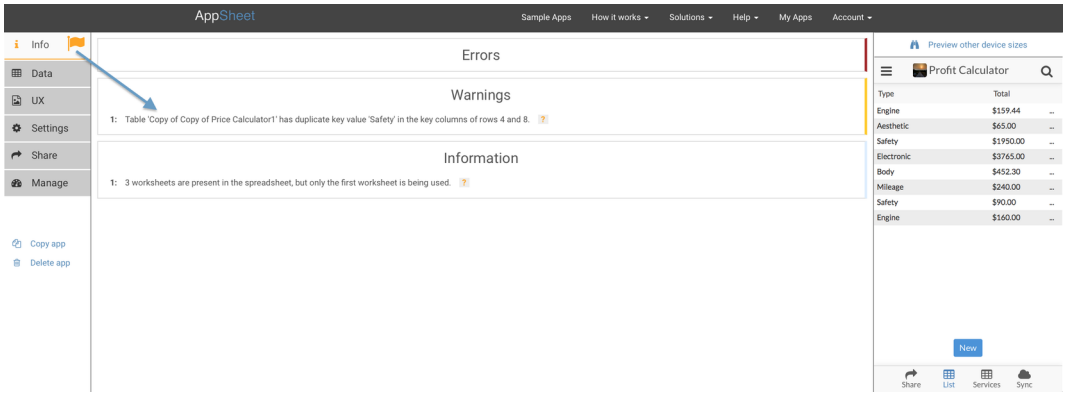


You can always choose a different key than what AppSheet has initially designated. Do this in the Advanced Editor>Data>Column Structure tab.

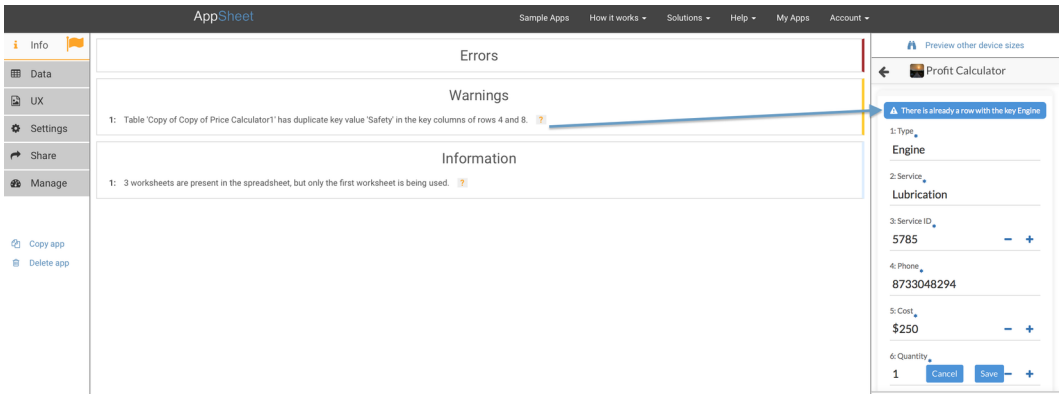
In the case of the auto repair shop, it can be assumed that eventually the Service column will contain duplicates and therefore might not be a good key. To work around this, I assigned a unique ID number to each service I offer and created a corresponding column in the spreadsheet. (Since I added another column to my spreadsheet, I will need to regenerate the column structure in the Editor to see my changes show up). Now I can use the Editor to switch my key to the Service ID column.

Service ID is now the first column in my app-- and it makes a great key because each value in that column will never repeat, nor will it change over time.

If you choose as the key a column that already has duplicate values, AppSheet will give you a warning. For example, I've switched the key to Type-- since it has repeating values, I get the warning (seen on the **Info** tab):

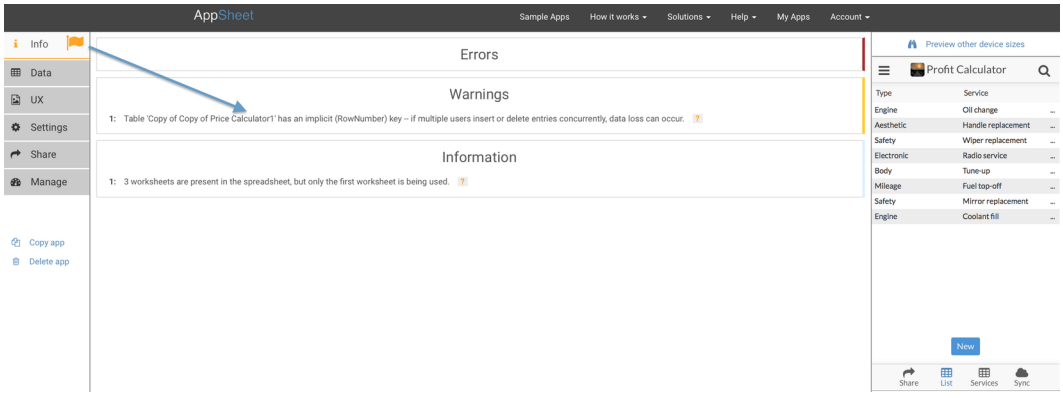


And if you try to submit an entry with data the key column already contains, AppSheet will give you a warning in the app and will not allow you to save the submission.



- **Why is RowNumber bad for a key?**

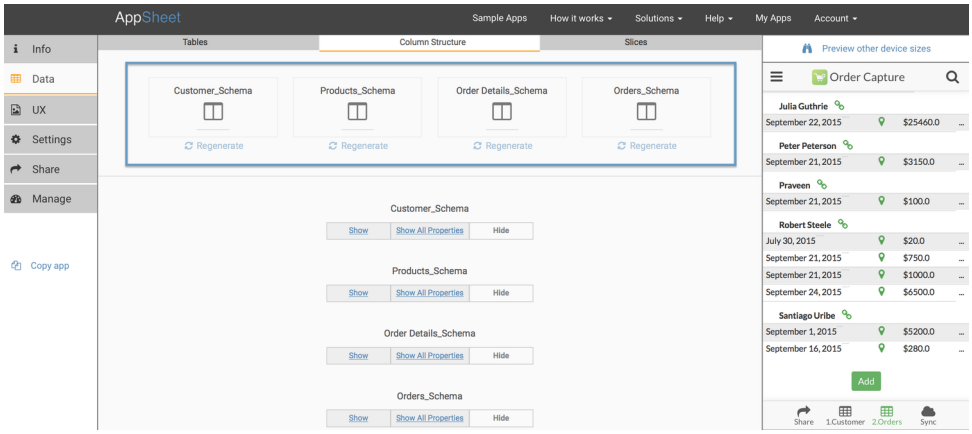
As stated before, if AppSheet cannot find an appropriate key within your spreadsheet, it will default the key to row number. This isn't a good key however, because if entries are moved or deleted, or if users add or delete entries simultaneously, the row number for each row then changes and there is no way for AppSheet to uniquely identify the row. AppSheet will also give a warning if you choose row number as the key:



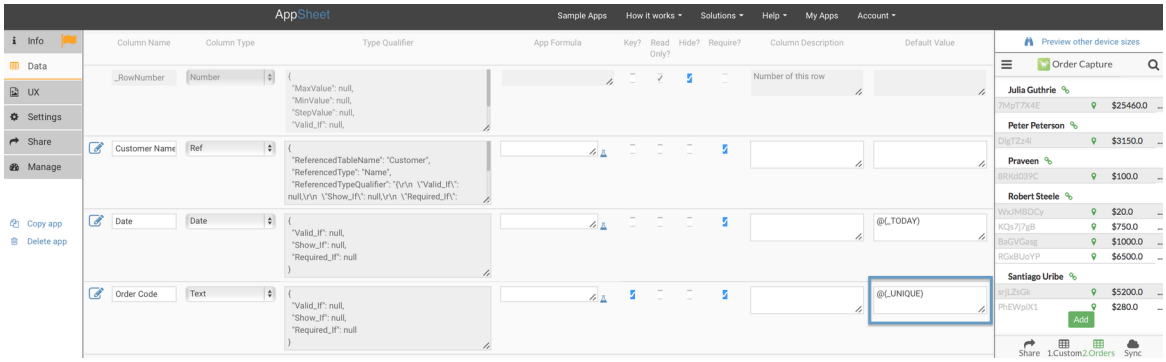
- **How do I automatically assign unique key values to my data?**

In some cases, you may need entries to automatically generate their own unique keys-- for example, in an app designed to capture product orders, each order must contain a unique order ID that cannot and should not be manually assigned. Our *Order Capture sample* shows you how to do this.

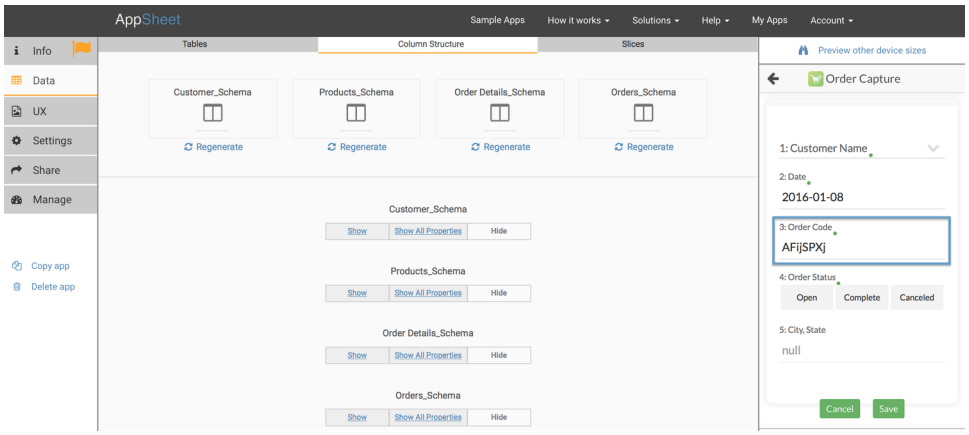
You'll see the app employs four different tables: Customers, Products, Order Details, and Orders.



If you scroll through the table structures you'll see that each table has its own key. In the Orders table, Order Code has been made the key, but there's just one more step to ensure each Order ID will be unique. You'll need to scroll to the right and insert UNIQUEID() in the Default Value field.



Now, when I go back to the Orders view and add a new order, a unique order code has been automatically generated by AppSheet.



Related articles

- *Specifying a key*
- *Expressions*
- *How do I control the order of columns displayed in the app?*
- *How can I receive (or send) an email alert when data changes?*
- *Change alerts and workflows*

Is it possible to create an incrementing number for each record? – AppSheet

Lots of customer ask for the ability to create monotonically increasing record number keys that accurately reflects the order in which the records were created. This is a natural desire, but it is technically impossible in a system that combines multiple distributed clients, who are allowed to concurrently add records, and who can add records while offline.

It is possible, but can be expensive, if you are willing to relax one or more of these requirements. 1. Monotonically increasing key, such as 1, 2, 3 ... or 1000, 1001, 1002, ... 2. Keys accurately reflect the order the records were first created by the client. 3. Two or more clients can add records concurrently. 4. Clients can add records while offline. For example, you can achieve a monotonically increasing key through a central locking scheme if you are willing to give up doing adds while offline. If you are willing to give up having the record number increase strictly monotonically and to reflect the exact order the records were created, you can give each client a block of numbers that only that client uses for its newly added record key values. The block of numbers reserved for each client must be large enough so that no client runs out of preassigned numbers while it is working offline. Obviously, the record number keys are no longer strictly monotonically increasing and they don't strictly reflect the order of creation, but they roughly reflect the order the records were added. You can assign monotonically increasing number keys to newly added records, by having the server assign the record number keys after the client synchronizes. The resulting record number keys reflect the order in which the newly added record first reached the server, rather than the absolute time they were created by each client. This can lead to some complications because the client only knows the record's "real key" after it synchronizes with the server. If the client adds a set of records that are related by key value, the client must assign the actual key values to each of the newly created records, and ensure they the references between these records reflect the actual key values. The client can only do this after the actual key values are returned by the server. There are other design alternatives, but all of them involve tradeoffs regarding which requirements you relax and how much cost you are willing to bear to achieve nearly monotonically increasing record number keys.

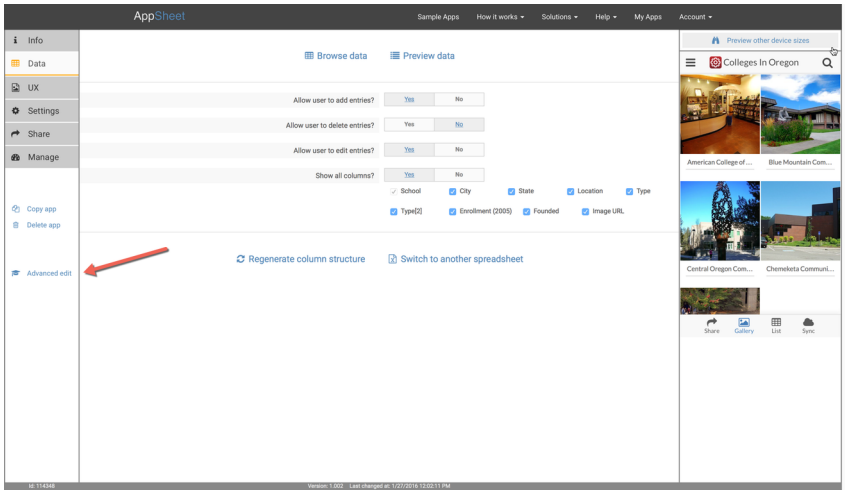
Related articles

- *Team collaboration, shared authoring of apps*
- *Required_If*
- *Other expressions*
- *How do I convert from US Dollars to my local Currency?*
- *Including PDF's in your application*

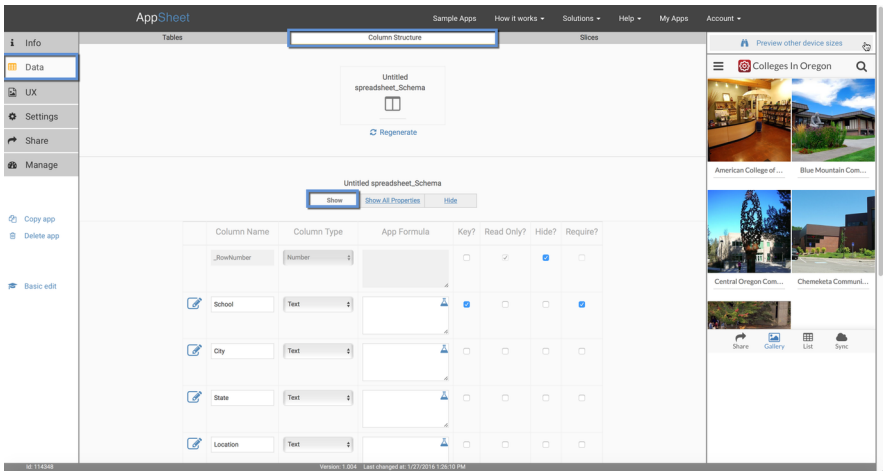
How do I change a Column Type? – AppSheet

Since ***Column Type*** impacts how AppSheet handles your data, selecting the appropriate column type is important. You designed your spreadsheet to be ***"AppSheet-friendl"y*** , but you would prefer to change one of your column types, perhaps from Text to Name.

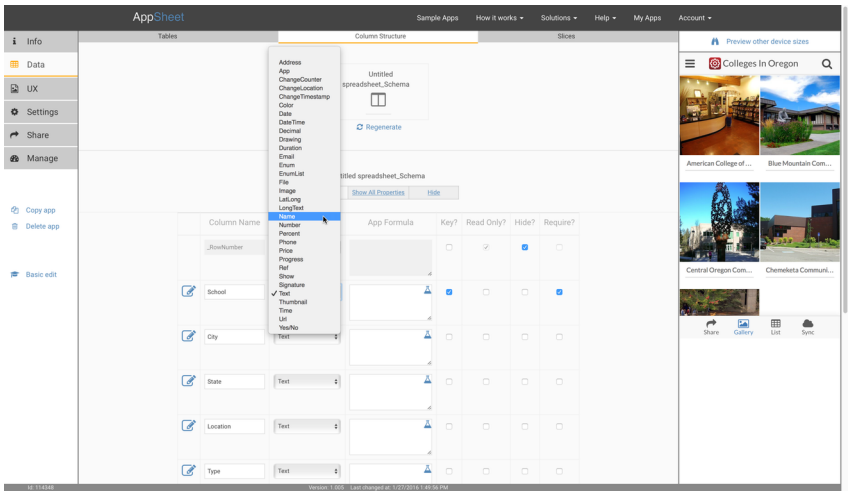
1. To change your Column Type, select the **Advanced Editor** from the left sidebar.



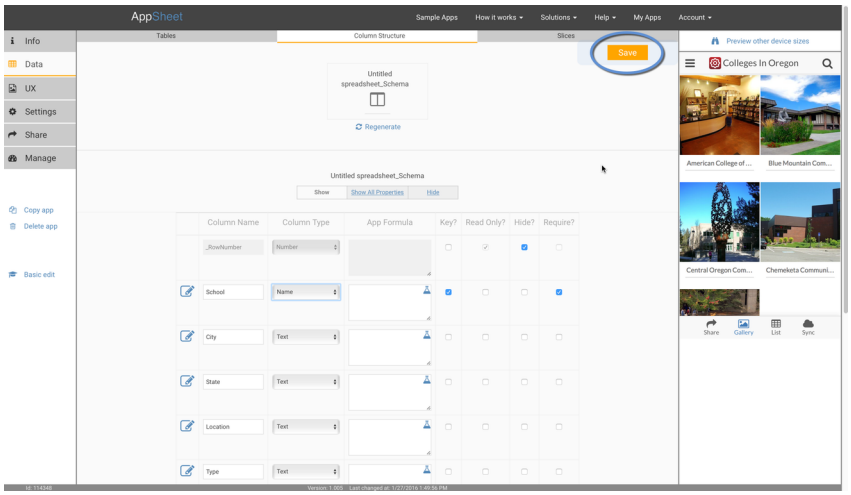
2. Next, choose **Data** > **Column Structure** and select **Show** for the table you would like to change.



3. Locate your Column Name and the corresponding **Column Type** that you would like to change. In this example, for the Column Name "School", Column Type was changed from "Text" to "Name."



4. Select **Save** and your Column Type change is complete.



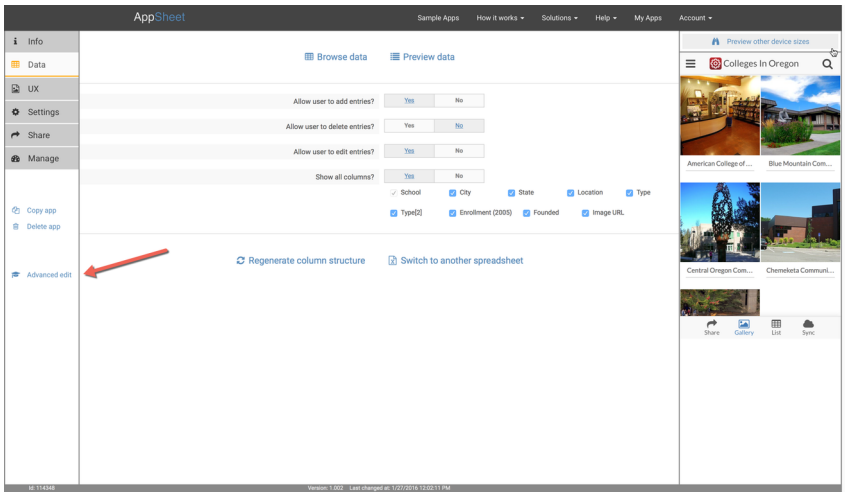
Related articles

- *How do I convert from US Dollars to my local Currency?*
- *Column types*
- *Team collaboration, shared authoring of apps*
- *Advanced app customizations*
- *Yes/No expressions*

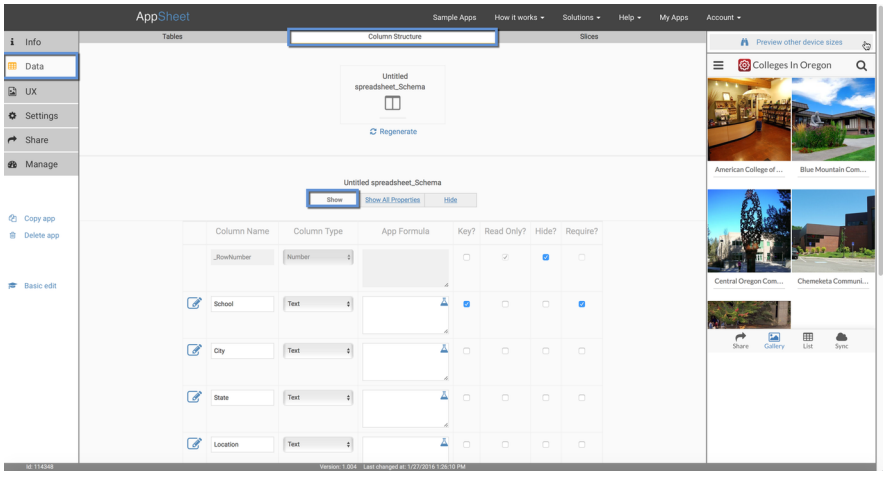
How do I convert from US Dollars to my local Currency? – AppSheet

Since *Column Type* impacts how AppSheet handles your data, selecting the appropriate column type is important. AppSheet may not have recognized your data as currency either because it doesn't yet recognize "Price" in your language or because you used a spreadsheet column header such as "Tuition." You would like to change your Column Type to "Price" and since AppSheet displays currency in US Dollars, regardless of your selected Data Locale, you would like to select the correct form of currency as well.

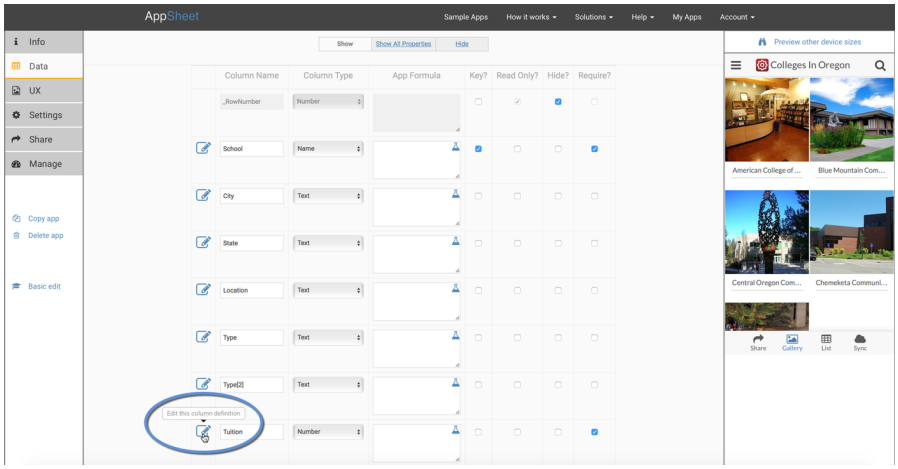
1. To change your Column Type, select the Advanced Editor from the left sidebar.



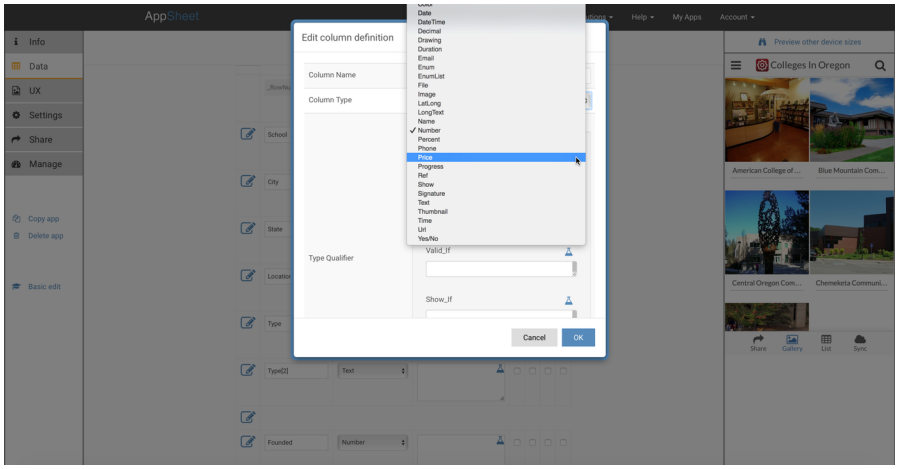
2. Next, choose Data > Column Structure and select Show for the table you would like to change.



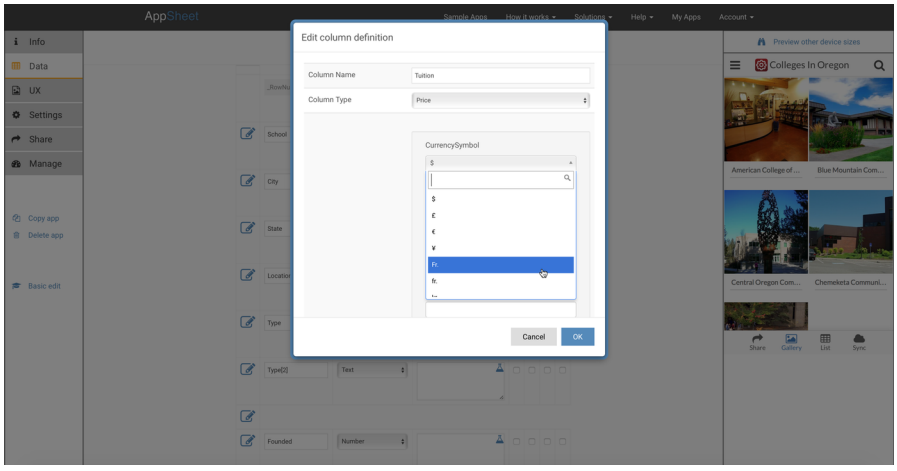
3. Locate the Column that you would like to change and select "Edit this column definition."



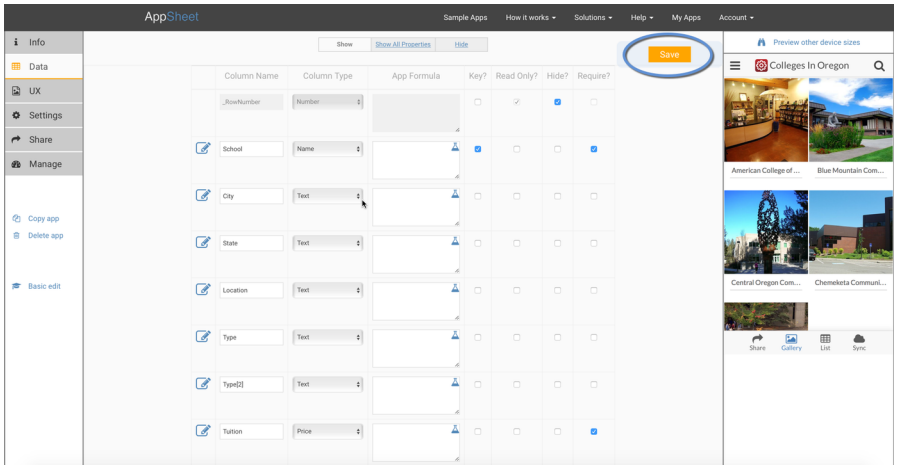
4. Select the Column Type dropdown and change it from "Number" to "Price."



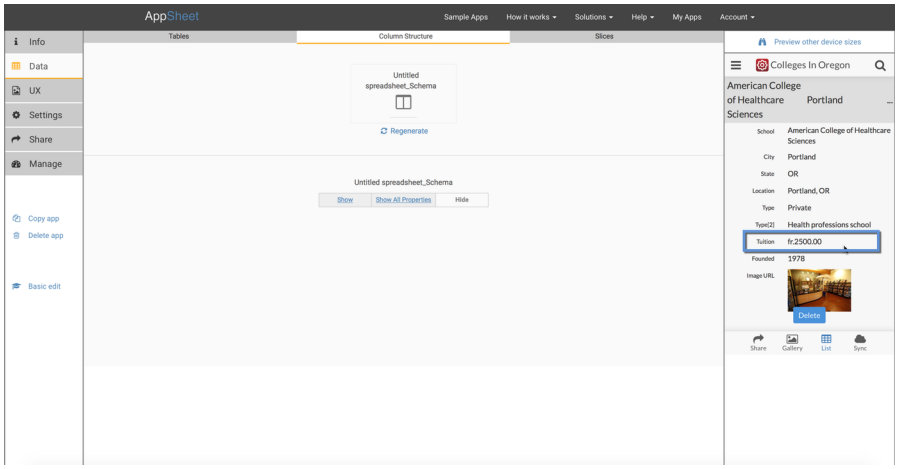
5. Select the CurrencySymbol dropdown and choose your local Currency Symbol. Click Ok.



6. Select Save and your Column Type "Price" and CurrencySymbol selection will be saved.



7. In this example, see the "Tuition" currency is now in Swiss Francs (fr.)



Related articles

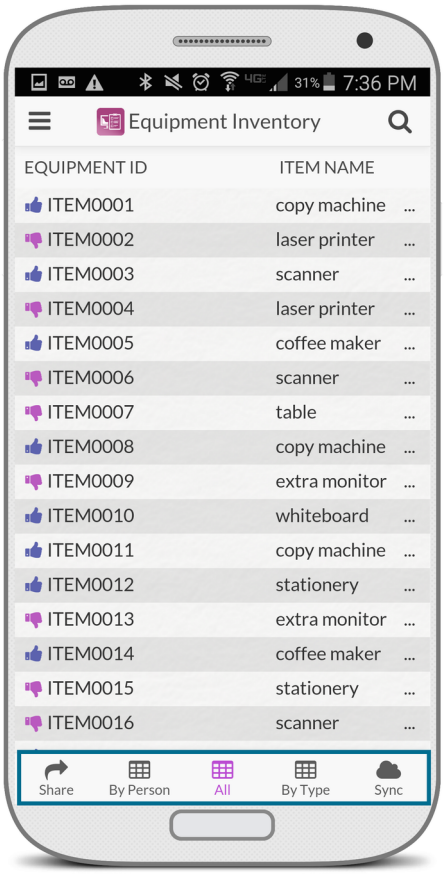
- *Locale support in AppSheet*
- *Column types*
- *Launching AppSheet apps from other AppSheet apps*
- *Lists expressions and Aggregates*
- *Required_If*

App Definition: User Experience

Presenting information

Choosing and adding data views – AppSheet

All AppSheet apps have a launch bar at the bottom. Users use the control buttons in the launch bar to switch between different views of the data.



Four buttons are present in all apps-- the Share button, the Sync button, the About button in the upper left "hamburger" menu, and the Log out button also found in the upper left "hamburger" menu.

- The **Share** button lets a user send another person an install link for the app.
- The **Sync** button lets a user synchronize data in the app with the data in the back-end spreadsheet.
- The **About** button is where you can provide information about your app.
- The **Log out** button allows you to log out of your app.

Then there are buttons present in some apps, depending on the update modes you have chosen for your app-- **New, Edit, Delete, Save,** and **Cancel**.

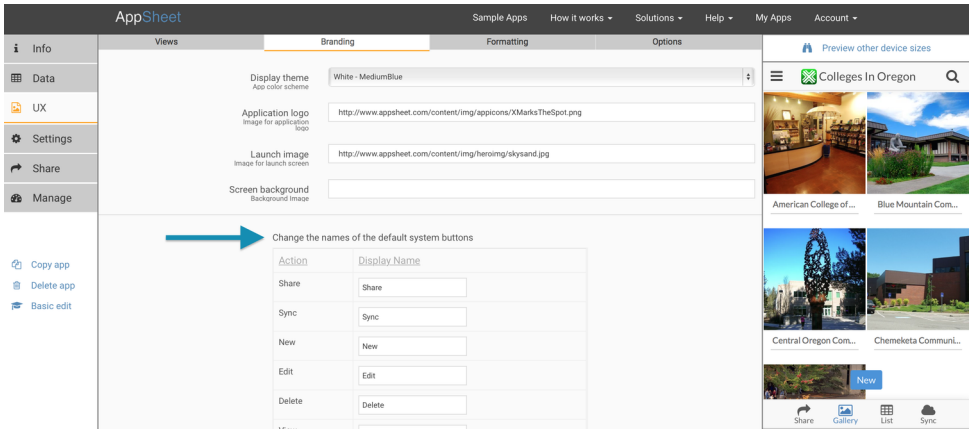
Language-specific UX

You can choose to change the names of all of the above buttons to accommodate your language or terminology preferences.

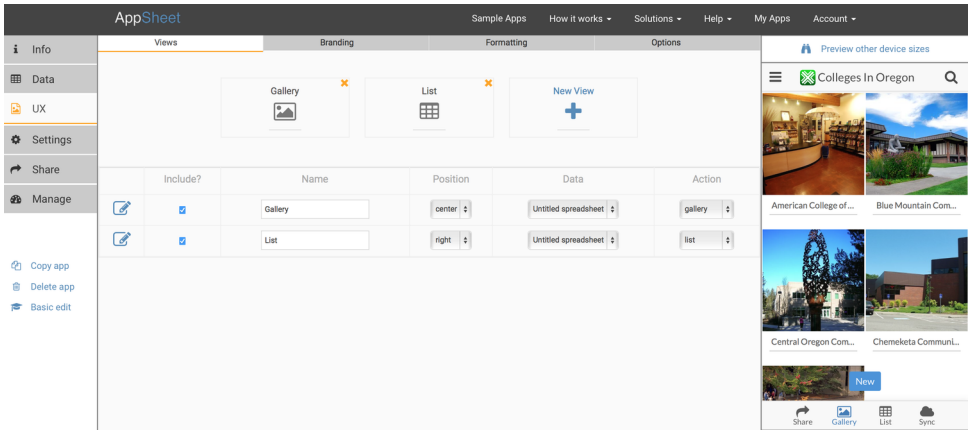
To do this, use the Advanced Editor>UX>Branding tab. You can change the button names at the bottom of this page.

We generally suggest using short names, so that they fit comfortably on the screen.

While this is a very simple solution, it is usually sufficient to address a large class of "localization" requirements for apps. Most apps target users with a specific language in mind. If an app needs to work with different languages simultaneously, this solution will not be adequate.

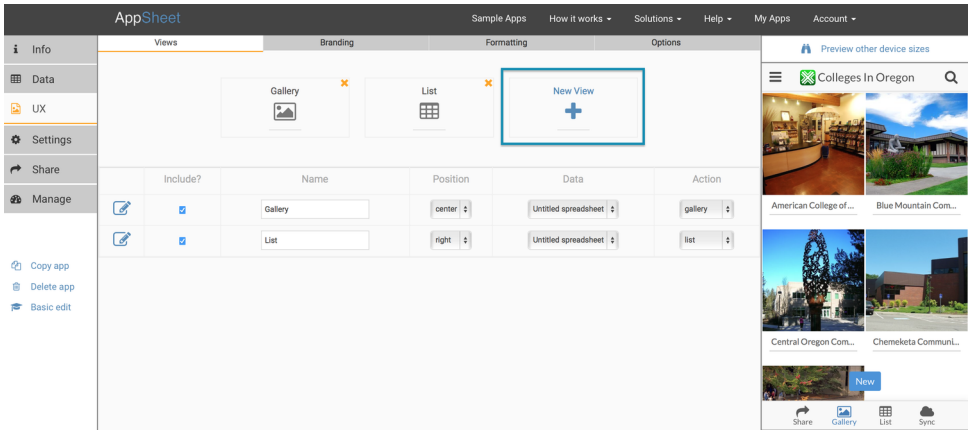


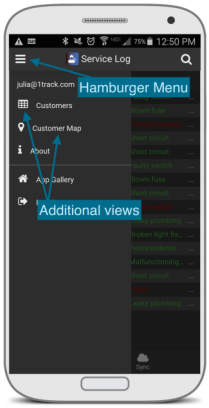
Defining your view controls As the creator of the app, you get to define the other view control buttons. You do this in the UX tab of the editor. The buttons are identified by their position-- Left, Center, Right, or Menu (in the hamburger menu). For each of the positions, you choose a control name, a presentation type, and a couple of presentation options as described below. If you leave the name empty, no view control button is shown in that position.



Adding more than three data views

Your app can specify more than three data views from the Advanced Editor>UX>Views tab. To add a new view, click the 'New View' button. Each view has a position, which can be chosen under the 'Action' dropdown menu-- Left, Center, or Right. For more than three control views, choose the 'Menu' option-- those will appear in the app when you expand the dropdown ('hamburger') menu at the top-left corner.



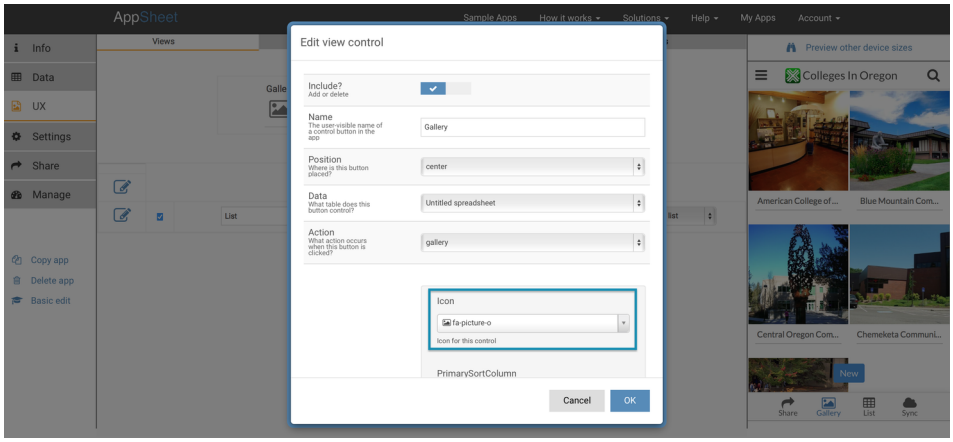


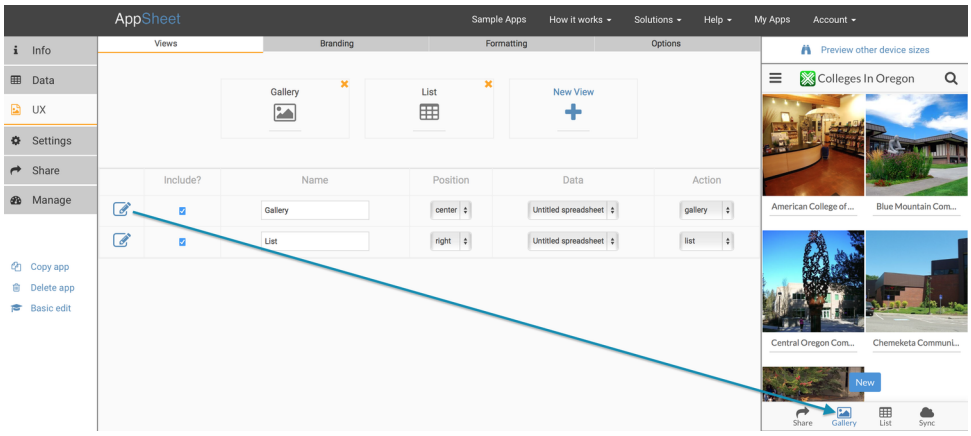
See an example of extra view controls in the *Contact Directory sample.*

Finally, to define "reference views" as described in this *article*, you can choose 'ref' as the position of a view.

Specifying an icon for your data views

You can choose a custom icon for each of your data views. To do this, use the Advanced Editor>UX>Views tab. When you edit the view control, you can choose an icon for that view from the icons dropdown menu.





Related articles

- *Advanced app customizations*
- *Working with charts*
- *Multi-page forms with conditional branching*
- *Reflecting your brand*

- *Working with maps*

Presentation types – AppSheet

AppSheet supports five presentation views-- **list, image gallery, map, chart,** and **form**. All of the views show a 'summary' view of the data with the ability to drill into a 'detailed' view of one specific entry. As with other aspects of AppSheet, the platform automatically generates most of the UX and the app creator guides the outcome primarily be changing the structure of the data.

- **List**-- this is the default view of tabular data. There are three display styles to choose from:
- Tabular: each row is showed in a compact representation. Tapping or clicking on a row expands it to show all the details. An extra option specifies the column to sort the data on. You can choose to sort the data in ascending or descending order.
- Grouped: rows are grouped together based on the value in the grouping column specified. Within each group, the data is shown using the tabular view and sorted based on the order of data in the underlying spreadsheet.
- Deck: this is a special presentation for data that has an image column. The data is displayed as a stack of 'cards', and clicking on any of them expands the card and its image. The deck view allows for round and square image presentations.

See how lists work in our Recruiting sample app.

2. **Gallery**-- this is the default view of data that has an image or thumbnail image column. The content is shown in a 'summary' view as an image gallery where each image is annotated with the row key. Clicking on any image opens up a full screen 'details' view of the image along with the other row columns. Once in the full screen details view, the user can swipe left or right to move to adjacent rows. This presentation has three sub-options that control the size of the images shown in the summary view:

- Small: appropriate for thumbnail images.
- Medium: fits two images side by side on a phone screen.
- Large: shows large images full-width with a vertical scroll.

As with the List view, you can choose a column to sort the order of presentation of the data. *See how the gallery works in our Product Catalog sample app.*

3. **Map**-- the map view only makes sense if you have an Address column. When you choose the Map view, AppSheet automatically picks up the data from your Address column, geocodes those addresses, and shows them on a map. *See how maps work in our Contact Directory sample app.*

Read a detailed blog post about the map feature.

4. **Chart**-- we support a number of different chart presentations. In general, charting is a complex topic. As the designer of a chart, you typically have to make three choices:

- How many individual series 'lines' do you draw?
- What values are on the X and Y axes?
- What chart display type do you use-- line, bar chart, etc.

We have tried to keep it simple by limiting the possible answers for each of these choices:

- Row Series chart-- this is appropriate for spreadsheet data where most of the data is numeric on a uniform scale (eg: monthly sales data). Every row forms a series, each of which is identified by the row key. You choose a subset of the column names that will be charted on the X axis. The Y axis is based on the numeric values in the row cells. You can choose a line chart, a bar chart, or a stacked bar chart as the display type. A good example of a Row Series chart is the 'By Month' view in the *Sales Report sample.*
- Column Series chart-- this is appropriate for spreadsheet data with just a few rows corresponding to items to compare (eg: monthly sales data by category). Every column forms a series, each of which is identified by the column name. Each row (using its row key column) forms one entry on the X axis. The Y axis is based on the numeric values in the row cells. You can choose a line chart, a bar chart or a stacked bar chart as the display type. A good example of a Column Series chart is the 'By Type' view in the *Sales Report sample.*
- Histogram-- this is a special kind of bar chart to show aggregate distributions. For example, if you want to see the number of customers who purchased each automobile model, you specify the column whose values will be

aggregated. The X axis gets one entry for each unique value in this column and the Y axis shows the count of the number of rows that have that value in that column.

5: **Form**-- Your apps give you the ability to capture signatures, photos, and location services.You can use AppSheet for Google Forms to both distribute forms through a mobile app as well as use a mobile app to view and interact with form responses. But if you're only using your app to distribute forms, you probably don't want your app showing each user's response, and instead have each new user simply be able to fill out a new form with the app.

To do this, click the UX tab in either editor. In the Basic Editor, choose 'form' from the dropdown list of presentation view types. In the Advanced Editor, click the UX tab, then 'Controls'. From there choose 'form' from the dropdown list of presentation view types under the Action field.

New guests to the form will only be able to fill out a new form and will not see entries made by previous users.

AppSheet also helps you unlock mobile-specific features for your Forms that the web cannot capture. Your apps give you the ability to capture signatures, photos, and location services.

6. **Deck**-- A view that displays a large image on the left, in your choice of either a round, square, or full presentation, with the most important information on the right and any action buttons that correspond to your field types.

7. **Table**-- This presentation is best with handling large data sets with many rows.

Over time, we will add more chart types, while trying to keep the concepts simple.

Read a detailed blog post about presentation types.

Related articles

- *How do I control the order of rows displayed in my app?*
- *Expressions*
- *Conditional formatting*
- *How do I control the order of columns displayed in the app?*
- *Multi-page forms with conditional branching*

Working with maps – AppSheet

Whenever you have address or lat/long geocoded data, AppSheet provides the ability to see it on a map. Columns of the Address or LatLong data types can be shown on maps. We utilize the Bing maps platform to handle addresses and mapping.

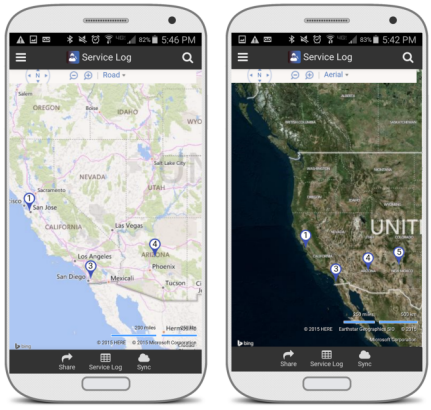
We rely on **Google Maps** to provide driving directions. If you are using an iPhone or iPad device, you must install Google maps on that device to obtain driving directions.

See how the map function works in our Service Log sample app.

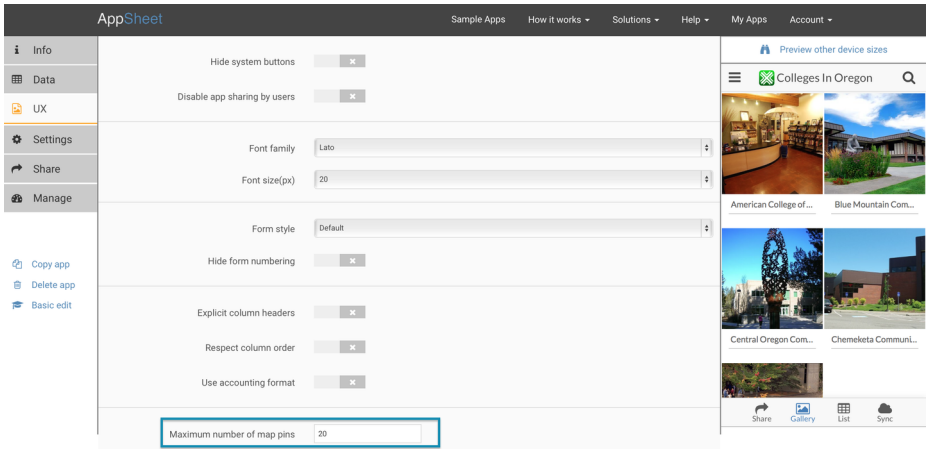
Read a detailed blog post about using the map feature.

Here are some other tips to make sure maps work well for you:

- **If you use the LatLong data type**
  - the data values should be of the form "44.2456, -122.3348", i.e. comma-separated latitude and longitude values.
  - LatLong data can also be captured in a form field when editing entries or adding new entries. This uses high-accuracy GPS location capture on your device, which is potentially a time-consuming and battery-intensive operation. This function should therefore be used only in appropriate situations.
- **When using addresses**
  - try to provide complete addresses (including city and country). This may seem redundant. For example, a local business in one city may record addresses without a City, State, Country, and ZipCode. While this makes sense to the customer, Bing maps does not have the context to know to which city '100 Main Street' refers. Always try to provide as much information as possible.
  - if you find that your addresses are not being mapped correctly, try to look up the address at maps.bing.com to see if there is a problem with the way the address is written.



Each of the entries in the table or slice being mapped is shown as a pin on the map. If you have many entries, this could lead to a messy map overrun with pins. Instead, AppSheet specifies a maximum number of pins to show, and chooses the closest pins around the current location. You can modify this maximum number for you app via the Advanced Editor>UX>Options tab.



You can interact with maps in the expected manner. Maps can be shown as road view or aerial view. The map view can be zoomed and panned as normal. By default, it starts zoomed out to a scale that can show all the pins. So, if you do have addresses to map in Australia as well as Canada, you will start with a map that is zoomed out to show half the world!

When you choose a specific item (or click/tap on a map pin), the map zooms in to the vicinity of that location. Some information about the highlighted item is shown at the top left of the view. This information typically includes a name (the contents of a column of type Name, or a key, or a Text column), an image if the data entry has an Image column, and a link to driving directions from the current location. This information can itself be clicked/tapped to see the full details of the entry.

Related articles

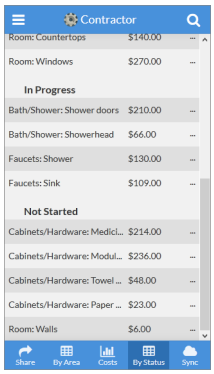
- *Capturing GPS location*
- *Conditional formatting*
- *Working with charts*
- *Capturing images and documents*
- *Expressions*

Grouped views and data filtering – AppSheet

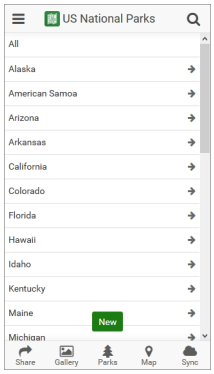
AppSheet offers a number of ways to view your data. *References* are great when you have data across tables. Spreadsheets with hundreds of rows present a different problem. Large data-sets need to be sorted and grouped to be useful.

In addition to adding group headers, AppSheet can also show a "drilldown" view where data can be filtered to a single column value.

Grouping headers

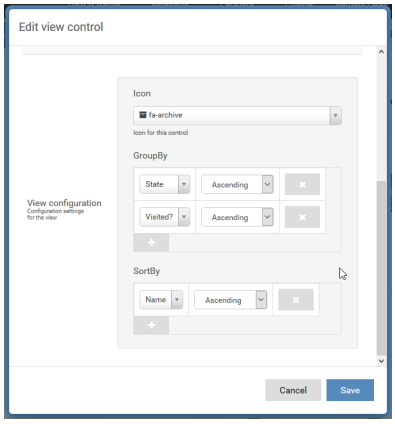


Drilldown view



These views can be created in the Advanced Editor>UX>Views by editing a view control and setting the GroupBy. You can group by as many columns as you would like by pressing the + button. To remove grouping columns you can press the X button.

If you just have one grouping column the app will show that view with group headers like the Contractor image above. Adding more columns will add drilldown views.



Note: If you don't want to have grouping headers on the final level of grouping you can add a GroupBy entry that groups on the _RowNumber column.

Related articles

- *References between tables*
- *Controlling data inputs with column constraints*
- *Expressions*
- *Conditional formatting*
- *App formulas*

Working with charts – AppSheet

We support a number of different chart presentations. In general, charting is a complex topic. As the designer of a chart, you typically have to make three choices:

- How many individual series 'lines' do you draw?
- What values are on the X and Y axes?
- What chart display type do you use-- line, bar chart, etc.

We have tried to keep it simple by limiting the possible answers for each of these choices:

- **Histogram** -- this is the ideal chart type to segment and drilldown a dataset. A histogram is a special kind of bar chart to show aggregate distributions. For example, if you want to see the number of customers who purchased each automobile model, you specify the column whose values will be aggregated. The X axis gets one entry for each unique value in this column and the Y axis shows the count of the number of rows that have that value in that column. You can specify multiple levels of drilldown by specifying multiple columns.
- **Row Series chart**-- this is appropriate for spreadsheet data where most of the data is numeric on a uniform scale (eg: monthly sales data). Every row forms a series, each of which is identified by the row key. You choose a subset of the column names that will be charted on the X axis. The Y axis is based on the numeric values in the row cells. You can choose a line chart, a bar chart, or a stacked bar chart as the display type. A good example of a Row Series chart is the 'By Month' view in the *Sales Report sample.*
- **Column Series chart**-- this is appropriate for spreadsheet data with just a few rows corresponding to items to compare (eg: monthly sales data by category). Every column forms a series, each of which is identified by the column name. Each row (using its row key column) forms one entry on the X axis. The Y axis is based on the numeric values in the row cells. You can choose a line chart, a bar chart or a stacked bar chart as the display type. A good example of a Column Series chart is the 'By Type' view in the *Sales Report sample.*

Over time, we will add more chart types, while trying to keep the concepts simple.

Related articles

- *Presentation types*
- *Customizing input forms*
- *Conditional formatting*
- *Expressions*
- *Viewing external content*

Capturing information

Customizing input forms – AppSheet

You can control the presentation of data input forms.

If you choose to allow users to add or edit data from within the Data pane, your app will give users the ability to create or modify a data row using a form. The form is a sequence of labels (column names) and input fields (specific to the column data type). You can customize the form in the following ways:

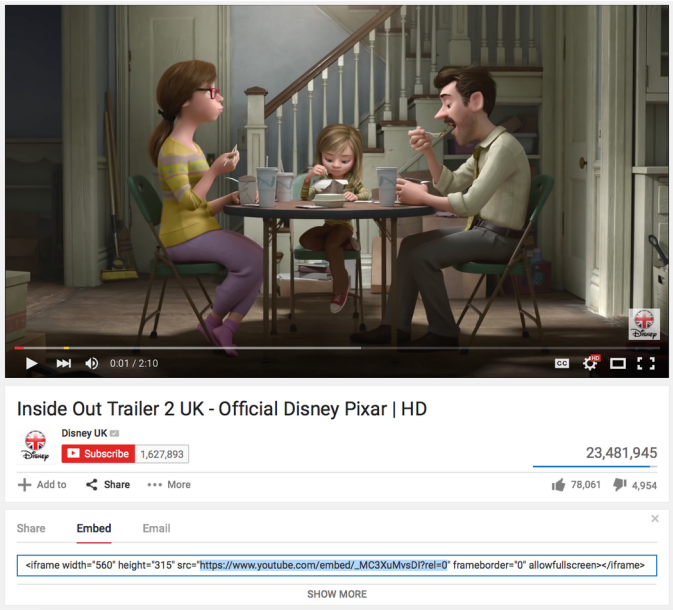
- Column descriptions-- you might want to provide a verbose description of each column for the user. If a comment is found, it is used in the label instead of the column name.
- Form style-- from the Advanced Editor>UX.Options tab, you can configure the layout of the form and whether the individual fields are numbered.

- Using "Show"-type columns.

Show-type columns are empty columns in your spreadsheet that serve the sole purpose of improving the presentation of data capture forms. There are six Categories of Show types. Use the Type Qualifier to tell the Editor what to show.

Categories:

1. Page_Header: used to create a new page within the form
2. Section_Header: used to create a new section within the same form page
3. Text: used to show some descriptive text
4. Url: used to show a clickable url **Here's an example of the Url Type Qualifier format: {"Category":"Url", "Content":"http://www.your-url.com"}**
5. Image: used to show a static image **Here's an example of the Image Type Qualifier format: {"Category":"Image","Content":"http://www.your-url.com/your-image.png"}**
6. Video: used to show an MP4 video or Youtube video (For MP4 videos, use the hosted link. For Youtube videos, use the embed link.)



Here's an example of the Video Type Qualifier format: {"Category":"Video","Content":"https://www.youtube.com/watch?v=IdwbSNJwLAk"}

Check out our *Movie Survey sample* to see a video embedded into a form using the 'Show' type.

Read a detailed blog post about customizing input forms.

Input form as the only view in the app

You can choose to make your input form the only view in your app, so new users are not able to see past submitted data.

To do this, simply remove any data view from the UX tab of either Editor (Advanced Editor>UX>Views, only keeping the form view.

Related articles

- *Multi-page forms with conditional branching*
- *How should I use multiple sheets in my app?*
- *Sync-- between the app and the backend*
- *Change alerts and workflows*
- *Who can discover and install your app?*

Sync-- between the app and the backend – AppSheet

Apps built with AppSheet are designed to handle intermittent connectivity loss, or full offline operation. This is achieved by maintaining a copy of the relevant data locally on the mobile device (securely, of course!). As a side-effect of this design, we have to consider when and how the local copy of the data stays in sync with the backend spreadsheet.

There are three cases to consider:

1. Changes are made in the app and need to be propagated to the backend.
2. Changes are made to the backend data (either directly, or by other users of the AppSheet app) and these need to be propagated to the app.
3. Changes are made to the app definition itself and need to be propagated to the app.

All of these occur as part of "Sync". There are four ways in which Sync may be invoked:

1. If this is the first time the app is being invoked on a specific device, Sync is automatically invoked when the app is started. This fetches the latest app definition along with the data needed to run the app.
2. For apps that work primarily online and want close to synchronous behavior, the app creator should disable the "Delayed Sync" setting. Every time the app user saves any edit, add, or delete action, this not only makes the change locally on the device but also immediately invokes Sync.
3. If the app has local changes that have not been propagated to the backend, the Sync button (at the bottom right) is highlighted. Clicking on Sync will manually invoke the Sync function. For apps that work primarily offline, the app creator should enable the "*Delayed Sync*" setting, and the app users will explicitly invoke Sync when they have network connectivity.
4. The app creator can also enable the "Sync on Start" option. When this is enabled, the app syncs every time it is started by the user.

Each Sync action itself has three steps that occur sequentially:

1. Any local changes are sent to the backend and applied to the backend data in the order they were originally executed.
2. The latest app definition is fetched down to the mobile device. If no change has occurred since the last Sync, this step is optimized.
3. The latest backend data is fetched down to the mobile device. If no change has occurred since the last Sync, this step is optimized.

Related articles

- *Multi-page forms with conditional branching*
- *Customizing input forms*
- *Plan upgrade required*
- *Mobile-specific features in my app*
- *Who should use AppSheet and what kind of apps can it create?*

Multi-page forms with conditional branching – AppSheet

You might want to break up a long input form into several pages for convenience. In some apps, it makes sense to conditionally show or hide a specific page based on choices made earlier. For example, in a form for a vehicle inspection report, you may have an initial question asking if there is damage. If the answer is Yes, you may want to show a Form page that has specific questions about the damage. If the answer is No, you would want to skip that page.

Conditional branching in Google Forms

When you are building your app from a Google Form, use **Add Item -> Page break** to split the form into multiple pages (Note: You can optionally provide a page header or description. This is helpful to remember where you're directing respondents and helping respondents understand your form's structure). You can then add **Multiple choice**

or **Choose from a list** items with the option "Go to page based on answer" and choose different navigation paths for each option. *Often you will also want these "conditional branchin"g questions be be marked as required.*

At each page break, you can also choose where the form should go next. Keep in mind that the page break navigation will only take effect under certain circumstances:

1. When the page has no conditional branching questions
2. When conditional branching questions are not required and no answer is given
3. When "Continue to next page" is chosen for specific answers to branching questions

After setting up your form, start the *AppSheet add-on* and click "Prepare". The AppSheet add-on creates "Page Header" columns in your response spreadsheet to represent each page break and automatically converts the form's "go-to-page" navigation into AppSheet's "show if" expression model. The Page Header columns appear in AppSheet as "*Show*" type columns, meaning they affect the visual presentation of forms. The Type Qualifier for these columns specifies the Category as 'Page_Header'. You can examine the column structure in the *Advanced Editor*>Data>Column Structure.

Most forms can now be correctly converted to AppSheet apps automatically, but there are two main limitations:

1. Reverse navigation is not supported by AppSheet. Try to arrange your form such that all navigation proceeds to higher page numbers.
2. In Google Forms you can include a special "Other" option for Multiple choice questions and assign it specific navigation behavior. However, the navigation associated with this choice is not made available to the AppSheet add-on. We recommend avoiding use of "Other" on questions where "Go to page based on answer" is enabled.

For more information about the conversion from Google Forms to AppSheet, see *this article*.

Conditional branching in your spreadsheet and the Advanced Editor

When you are building your form app from a spreadsheet, insert an empty column at each point where you'd like a page break in the form.

While conditional branching in Google Forms is based on a "go-to-page" navigation model, AppSheet uses a "show if" model to describe whether a given page should be shown or hidden. By default, pages are always shown. To make a page appear conditionally,

1. Find the corresponding Page Header at Advanced Editor>Data>Column Structure (the empty page-break columns in the spreadsheet are treated within AppSheet as "*Show*"-type fields with Category "Page_Header").
2. Click the "edit" icon on the left side and scroll down to the Type Qualifier settings.
3. Create an *expression* in the "Show_If" field that defines when the page should be shown.

Example

Suppose you have an Enum (dropdown) on the first page called "Cat or dog person?" with choices "Cat" and "Dog", and you have page breaks for an "About Cats" page and an "About Dogs" page.

The "About Cats" page can be shown only when "Cat" is chosen by setting the Show_If field of the Page Header to the expression **[Cat or dog person?]="Cat"**.

Likewise the "About Dogs" page can be shown only when "Dog" is chosen by using the Show_If expression **[Cat or dog person?]="Dog"**.

Which should I use?

Auto-generating your app from a Google Form with the *AppSheet add-on* is often easier and more convenient, since the add-on takes care of building the Show_If expressions for you. However, if your form requires complex navigation conditions that depend on multiple fields, you may not be able to represent it with Google's "go-to-page" model.

Creating your own Show_If expressions in the Advanced Editor gives you more freedom to define complex composite or comparison conditions that can't be represented in Google Forms using functions like AND, OR, NOT, etc.

Related articles

- *Customizing input forms*
- *Controlling data inputs with column constraints*
- *How should I use multiple sheets in my app?*
- *Conditional formatting*
- *Capturing GPS location*

Capturing GPS location – AppSheet

Use a column of type LatLong to capture GPS locations.

There are three ways to capture the GPS location in a form:

1. If you give the column a default value of '@(_HERE)' (using the advanced editor) then every time a new entry is added, the value of this column is auto-populated with the current location.
2. The input field for a LatLong column has a clickable icon that lets you capture the current GPS location of the device. This is useful particularly when you want to update the value of a field to a new current location.
3. You can explicitly type in a LatLong value-- eg: '46.34,-32.34'

As a special case, you can also use a column of type ChangeLocation to capture GPS locations. Such a column captures the location when some other value in the record changes. A separate article discusses Change types.

Whether on a mobile device or in a browser, the system will probably ask you to give AppSheet permission to access your current location. If you are running your app within a browser, this permission request may not be very prominent (some browsers bring up a subtle request near the address bar and it can be easy to overlook).

Related articles

- *Working with maps*
- *Capturing images and documents*
- *Tracking changes using 'Change' column types*
- *Column types*
- *Displaying images and documents*

Tracking changes using 'Change' column types – AppSheet

In some apps, it is important to track when and where some data was changed. For example, in a field service app, it may be important to record when a certain field was changed.

AppSheet has a simple mechanism to track changes-- via 'Change' column types. There are three "Change" column types-- **ChangeTimestamp, ChangeCounter,** and **ChangeLocation**. These column types all have a set of shared characteristics that are specified via the column Type Qualifier:

1. What triggers a change: any change to any other column, any change to some specific fields or a single field, or changes to a specific field that result in one of a known set of values
2. What should happen when such a change is detected: increment a counter (ChangeCounter), set a timestamp to the current time (ChangeTimestamp), or record the current location (ChangeLocation)

As a simple example, in the *Interview Feedback sample*, the Feedback table has a Changes column of type ChangeColumn. The type qualifier doesn't "qualify" the type in any way, so this column will track changes to any other field in the row. If any other row changes, the value of the Changes column is incremented.

A more typical example would want to track changes to specific columns, perhaps even more specifically changes of those columns to specific values. In the *Store Inventory sample*, the Inventory List table has a Date Checked column. It listens for changes to a specific other column, Quantity In Stock. And in fact, it looks to see if that value changes to 0. If it does, then the column records the current timestamp.

Related articles

- *Using a scanner*

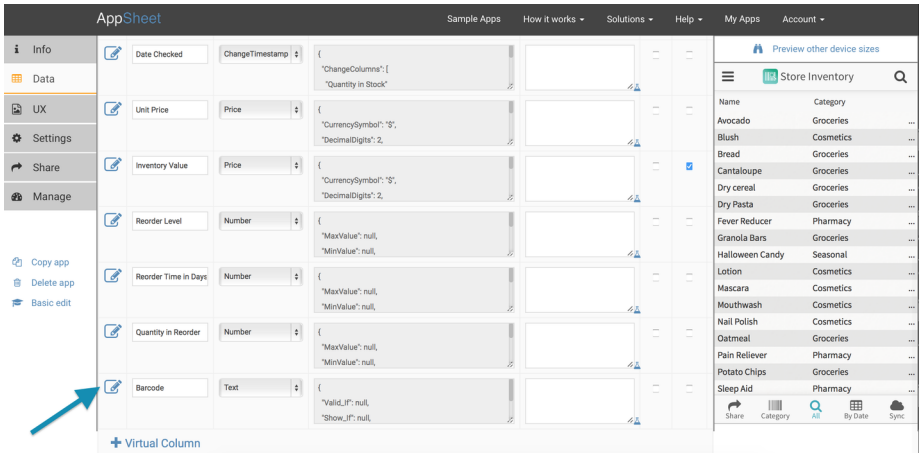
- *Grouped views and data filtering*
- *Expressions*
- *Viewing external content*
- *Slices*

Using a scanner – AppSheet

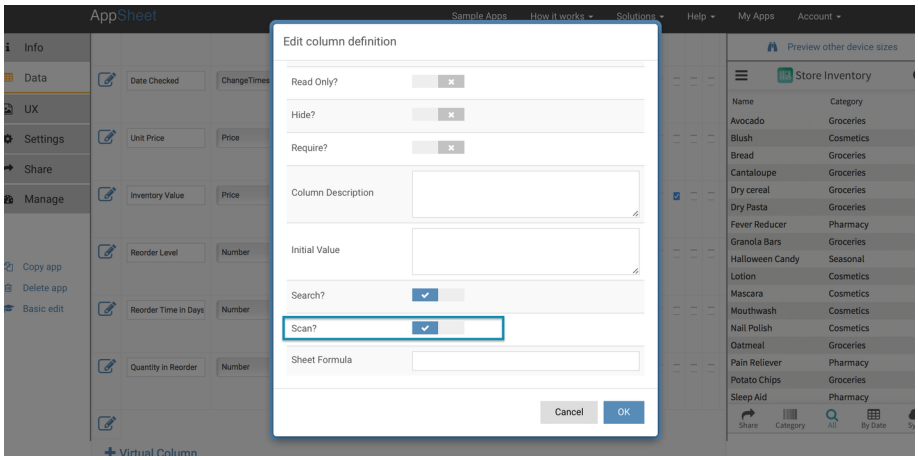
AppSheet can use the camera on your phone or tablet to capture barcoded or qr-coded data.

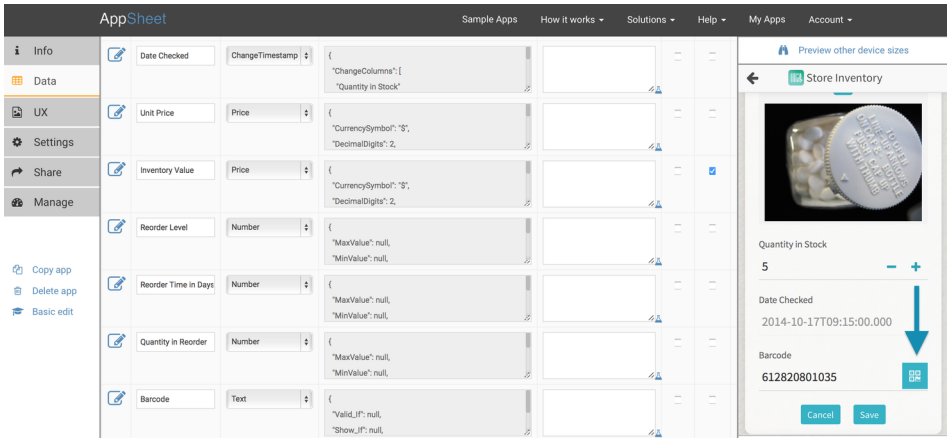
When you have marked an input column as scannable, your AppSheet application displays a scan icon next to that input field. When you click this icon, AppSheet activates the camera on your device and automatically reads the barcode or qr-code. AppSheet then saves this scanned value in the input field. AppSheet will recognize the column type you've chosen for that field (i.e. URL, number, decimal, and so on) and populate the data in the correct format.When you save your changes, the scanned value is saved along with any other values you may have entered.

Use the Advanced Editor>Data>Column Structure to choose the column or columns that you'd like to be scannable.



For each scannable column, simply check the 'Scan?' checkbox.





The Search box is also scanner-enabled when a ScanText column is searchable.

Note: you do not need to download a separate scanner app to use this option. Most smartphones come automatically equipped with scanning functionality.

Related articles

- *Displaying images and documents*
- *Column types*
- *Capturing signatures and drawings*
- *App formulas*
- *How do I design a secure app?*

Working with Media Types

Displaying images and documents – AppSheet

Here are some guidelines on using images and documents in your app.

Many applications (e.g. a product catalog or a photo diary) use images. Use the Image column type. Likewise, some apps use PDF documents. Use the File column type.

There are two ways to represent an image or document in your spreadsheet-- as a URL or as a filename.

If you are providing a URL, please be careful to ensure that the URL is publicly accessible! Many users who choose this option provide image URLs that appear accessible to them, but are not really publicly accessible. Here are some examples of **bad** image URLs:

- An image URL from your Facebook album-- you may be able to access the image because you are logged into Facebook, but when some other user tries to see the image, they will get an error. Whenever you are using an image from some kind of online storage provider, make sure that public access permissions are provided.
- An image URL from your Dropbox or Google Drive-- most cloud file systems show you an image on a webpage, but the webpage itself is not actually an image. It is a page that _hosts_ an image. For example, : *https://drive.google.com/file/d/{fileId}/view* is an image hosting page on Google Drive, but the actual image URL is https://drive.google.com/uc?export=view&id={fileId}
- An image URL from your local computer-- of the form file://MyImages/MyImage.jpg or C:/MyImages/MyImage.jpg. AppSheet has no access to your local computer, nor do you have the option to 'upload' an image to AppSheet. Instead, place your images in your cloud storage provider and use one of the mechanisms described above.

If you want to use a filename, it must be stored in your cloud file system, not on your desktop!

- In the app editors, we provide a file selector widget that lets you browse your cloud file system to find an image.

- If you are specifying an image or document file name in your spreadsheet, the file should be in the same folder location as your spreadsheet. For example, if you use Google Drive and your spreadsheet is in the /appsheet/data/MyApp folder, then if you have the image 'MyImage.jpg' in the same folder, you can just use the value 'MyImage.jpg' in the appropriate cell.
- It is sometimes easier to organize images in their own folder. For that reason, we allow image file names to be specified relative to the location of the spreadsheet. For example, if your images are in a subfolder called 'Images', you can use the cell value 'Images/MyImage.jpg' or './Images/MyImage.jpg'.

For apps that want to exclusively work with small thumbnail images, we also provide a Thumbnail column type. However, AppSheet does not currently automatically detect the difference between a regular image and a thumbnail. To explicitly choose the Thumbnail type, please use our Advanced Editor, as described in this document. To see an example of the use of thumbnails, look at the *Marketing Docs sample*.

See this *Google Sheets article* if you need assistance with image sizing.

Related articles

- *Capturing images and documents*
- *Reflecting your brand*
- *Column types*
- *Capturing GPS location*
- *Specifying a key*

Viewing external content – AppSheet

Your app can also show URLs and documents.

We support two data types for external content-- **URL** and **File**. If you have URLs in a column, the app allows users to click on the URL and see the content in a hosted web browser window. Additionally, the users have the option of opening the content in the native web browser on the device (eg: Mobile Safari on iOS).

If you have PDFs or other documents, the app similarly opens the content in a hosted web browser. Depending on the device and operating system, there may be support for viewing of a variety of file types-- for example, most browsers enable PDF viewing.

To include files or PDFs in your app, you will need to add either the file name or the URL where the document is hosted. For example, in the *Marketing Docs sample*, you'll see that the PDFs are coming from Google Drive URLs. In order for these to be viewable by third parties, we made sure they were publicly accessible by placing them in a public folder in Google Drive.

Related articles

- *Capturing images and documents*
- *Customizing input forms*
- *Capturing signatures and drawings*
- *Working with charts*
- *Column types*

Capturing images and documents – AppSheet

Documents (PDF files) can be captured by an app only when it is run in a desktop browser. Documents require a column of type File that is editable. Mobile devices do not have a standard file system, which is why file upload is not supported on mobile devices at the moment.

Many applications (e.g. a vehicle inspection app) need to capture an image in a column of a data row. If your app has a column of type Image or Thumbnail, and you have enabled data Edit or Add, then the app can capture images when editing or adding a data row.

On Android devices, by default, the user is prompted with the choice to take a new photo or use the camera roll. On iOS devices, by default, the camera is launched. If you want to optionally browse the photos in your camera roll, you should enable a setting for the AppSheet app on the device.

Here's how AppSheet processes the photos taken with the camera:

- Any photo taken on a device with your app is saved in full resolution in the camera roll on the device. *See how the camera function works in our Site Scout sample app.* Note: if you use the app on the desktop, the image capture function only allows you to open an image file. Using the image function on a mobile device will allow you to either upload an image from your device's gallery or to manually take a photo.
- When the user presses the Sync button, the photo is downloaded at a reduced resolution (to conserve network bandwidth) and is saved in the same folder location as the spreadsheet (all images are saved into a subfolder). The image resolution used is controllable via an app setting in the Advanced Editor>Settings>Content tab.
- The row in the spreadsheet includes the name of the image file just created, so that all subsequent accesses work smoothly. The image file name includes the row key as well as the column name so that it can easily be correlated to the corresponding row.

If you would like to have URLs to view these images, you should add an additional URL column and use the following spreadsheet formula (on Google Sheets) to construct image urls from the image file names:

Format:

=SUBSTITUTE(CONCATENATE("https://www.appsheet.com/template/gettablefileurl?appName=","**[AppName-Account#]**","&tableName=","**[TableName]**","&fileName=",+**[ImageColumnCell]**) , " " , "%20")

Sample:

=SUBSTITUTE(CONCATENATE("https://www.appsheet.com/template/gettablefileurl?appName=","**Inventory-114348**","&tableName=","**Orders**","&fileName=",+**B2**) , " " , "%20")

Note: Your image folder must be publicly accessible.

Now that you have the correct URL format, you may also embed the image inline within your spreadsheet (on Google Sheets). To do so, wrap the URL formula in an IMAGE() expression.

Format:

=IMAGE(SUBSTITUTE(CONCATENATE("https://www.appsheet.com/template/gettablefileurl?appName=","**[AppName-Account#]**","&tableName=","**[TableName]**","&fileName=",+**[ImageColumnCell]**) , " " , "%20"))

Sample:

=IMAGE(SUBSTITUTE(CONCATENATE("https://www.appsheet.com/template/gettablefileurl?appName=","**Inventory-114348**","&tableName=","**Orders**","&fileName=",+**B2**) , " " , "%20"))

If you have issues with device crashes or hangs while taking photos please see *this article*.

Related articles

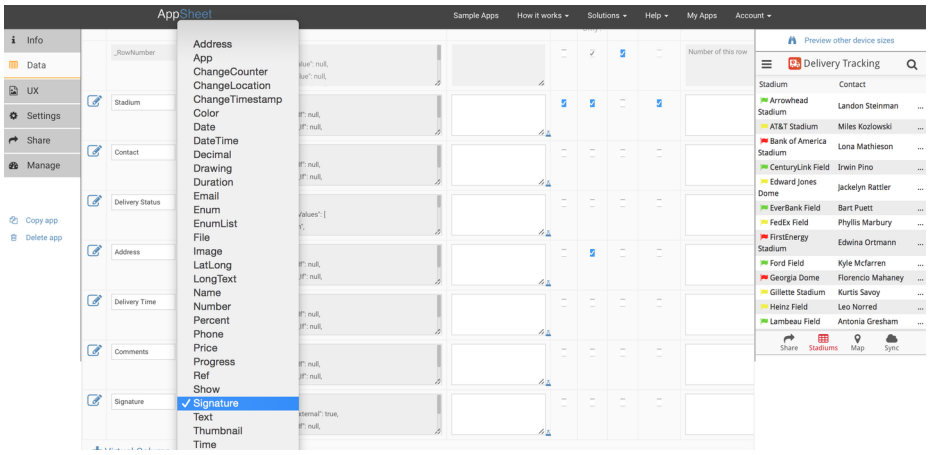
- *Displaying images and documents*
- *Capturing signatures and drawings*
- *Capturing GPS location*
- *Reflecting your brand*
- *Errors and warnings in the editor*

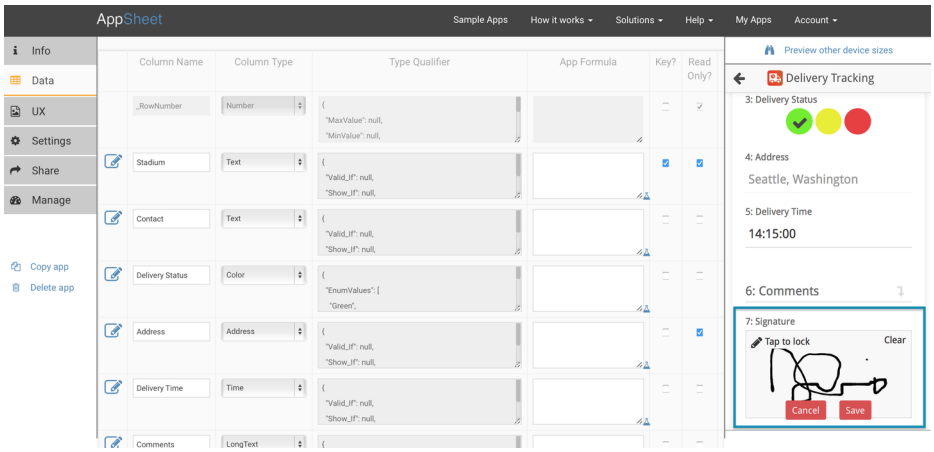
Capturing signatures and drawings – AppSheet

Capturing signatures is easy with AppSheet. Simply insert a column in your spreadsheet with the word "Signature" as the *column header* where you would like the signature pad to appear in the form. AppSheet will automatically detect this new column as a Signature Type after you *regenerate column structure*.

Alternatively, you can name the column anything you like and change the Column Type to Signature manually in the Advanced Editor>Data>Column Structure tab.

The Drawing type is similar to Signature, but provides a larger canvas and colored pens in order to capture free-form sketches within the app.





Related articles

- *Capturing images and documents*
- *Displaying images and documents*
- *Column types*
- *Capturing GPS location*
- *Expressions*

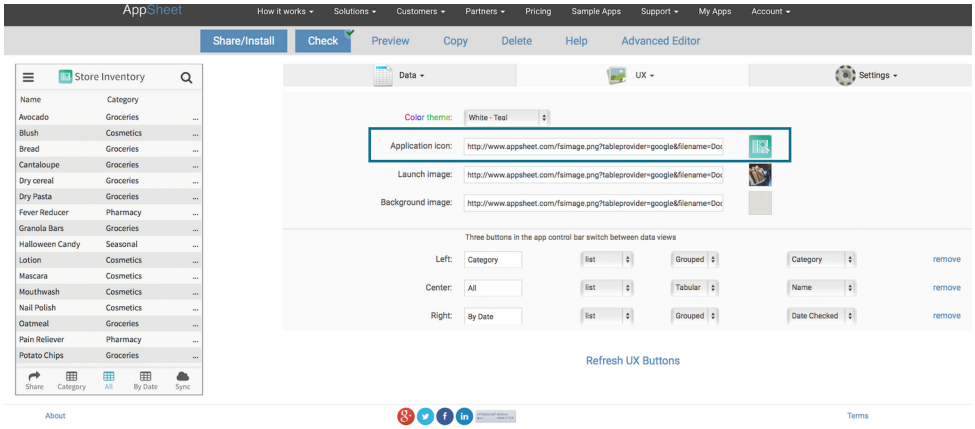
Branding and styling the app

Reflecting your brand – AppSheet

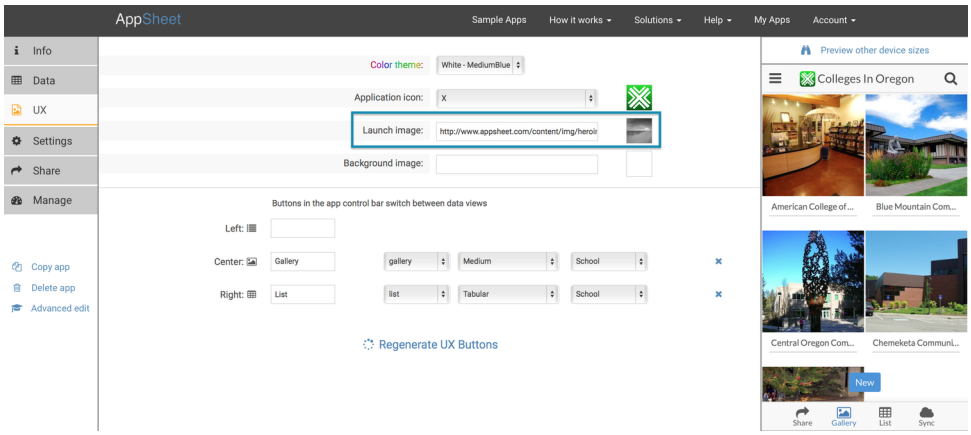
There are different options to reflect the brand of your business in your app. These options are found on the UX tab of both the Basic and Advanced Editors. To add branding images, click into the corresponding image fields in either Editor. AppSheet will produce a popup that will allow you to pick images from your cloud provider. It's a good idea to keep your images in the same folder your app's spreadsheet is housed.

1. **Icon**-- this is the icon that your users will click to launch your app from the device's home screen. We encourage you to provide an icon that reflects your business or brand. This icon should

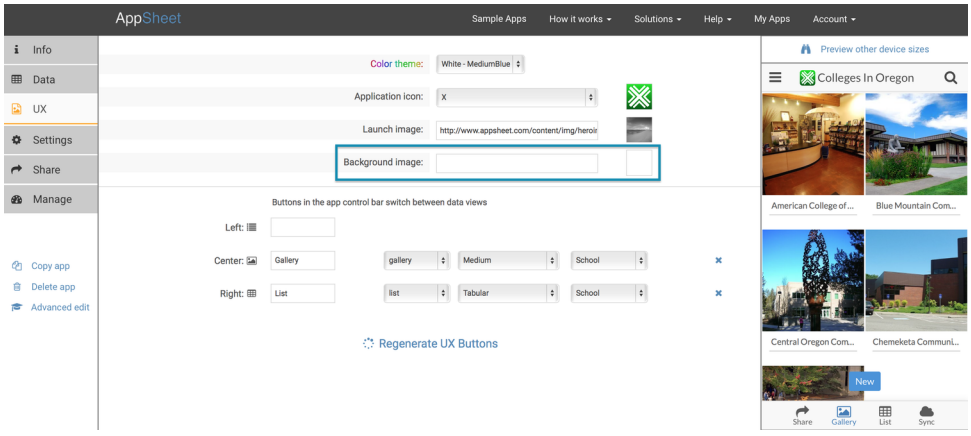
be square and up to 196X196 in size (we automatically resize it for different environments).



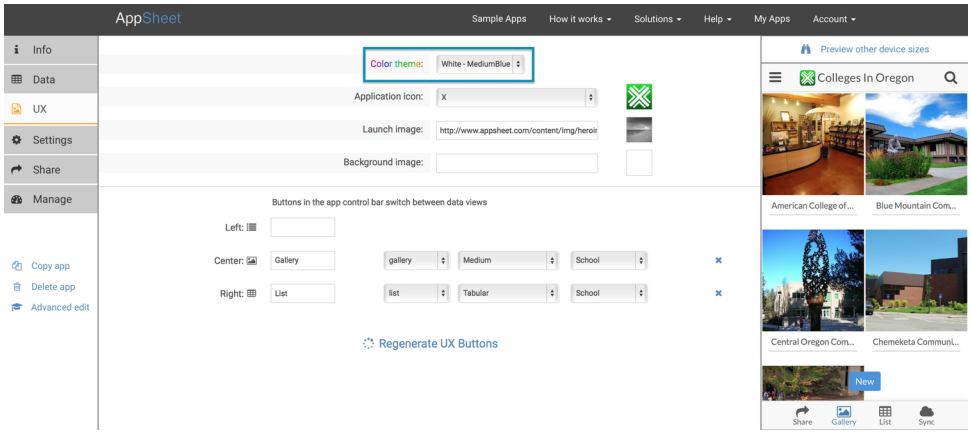
2. **Launch image**-- this is image that shows while the application is initially launched, and when Sync is in progress. It should be at least as wide as 600 pixels.



3. **Background image**-- this image forms a semi-opaque backdrop for all pages in the app.It should be at least as wide as 600 pixels. Try different background images to find one that is aesthetically pleasing given the actual data in your app. We have found that if your data already contains images, a background image can be distracting.



4. **Color theme**-- the color theme represents the color palette used in the app. We support both light and dark color themes, with the option to add more colors to the header, footer, text, and icons in your app.



Related articles

- *Displaying images and documents*
- *Offline behavior*
- *Check your app for deployment*
- *Advanced app customizations*
- *Conditional formatting*

Changing UX styles – AppSheet

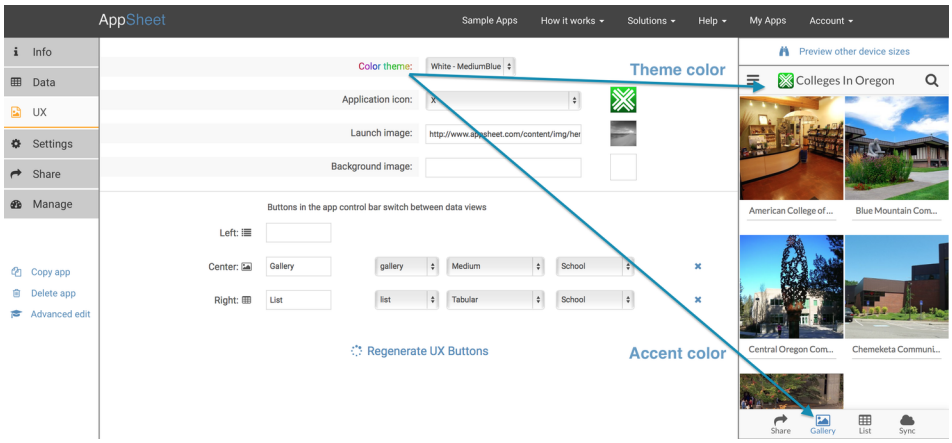
There are different options to change the color theme and text fonts of the app.

Changing theme colors

You can customize the theme and accent colors of your app in the UX tab of both editors.

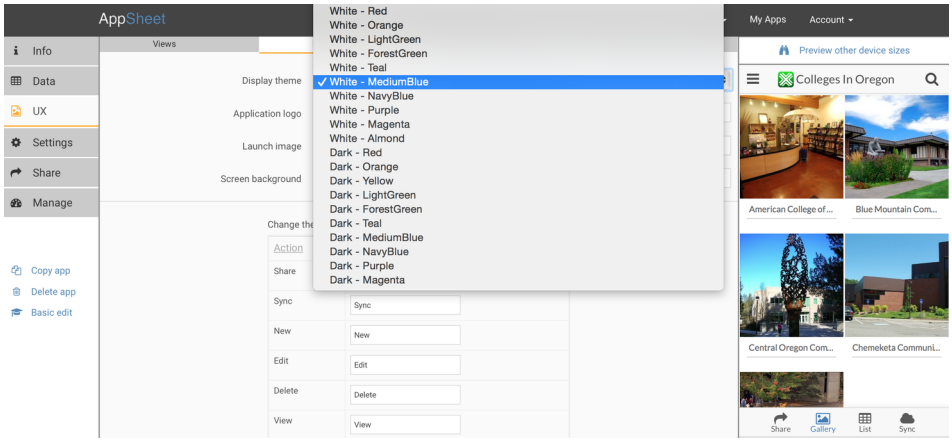
Next to 'Color theme' you'll see a dropdown list of colors. Single colors set the theme of the app to that color only, and double colors change both the theme color and the accent colors.

Basic Editor



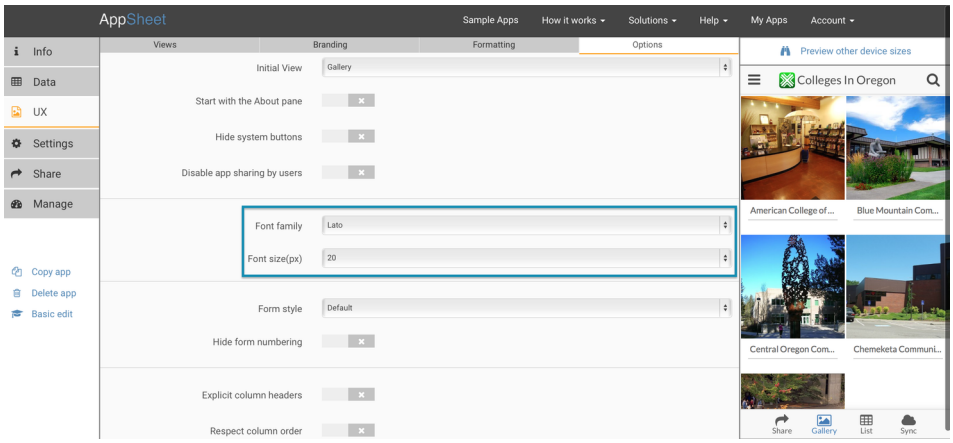
Advanced Editor

Go to the Advanced Editor>UX>Branding tab.



Changing fonts and font sizes

Using the Advanced Editor>UX>Options tab, you can choose from a set of available fonts and also modify the font size.



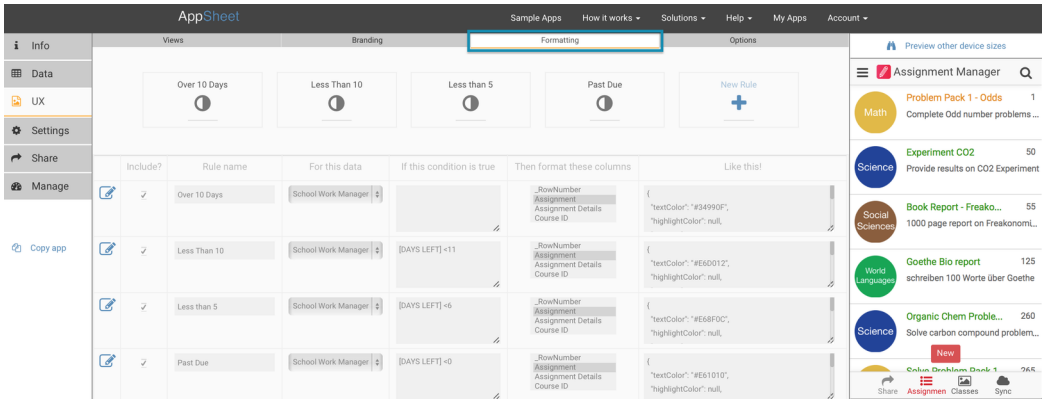
Related articles

- *Conditional formatting*
- *Setting the launch page of your app*
- *Advanced app customizations*
- *Reflecting your brand*
- *Column types*

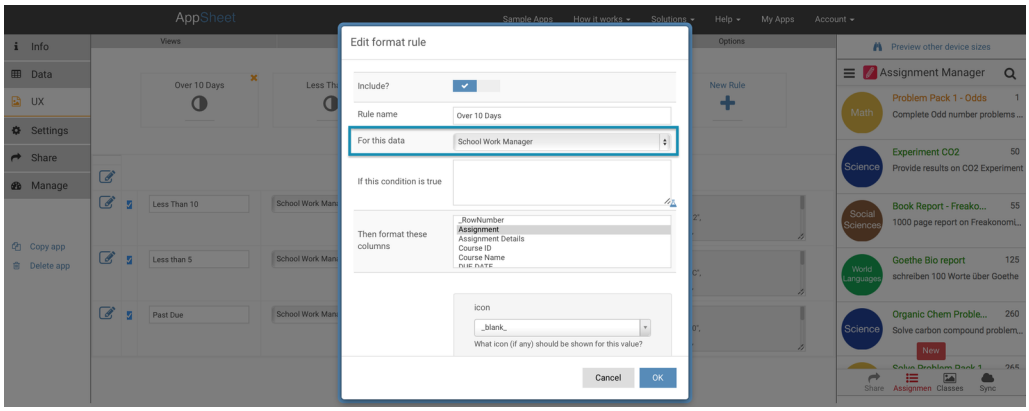
Conditional formatting – AppSheet

You can change the way information is displayed throughout your apps.

Go to the Advanced Editor>UX>Formatting:

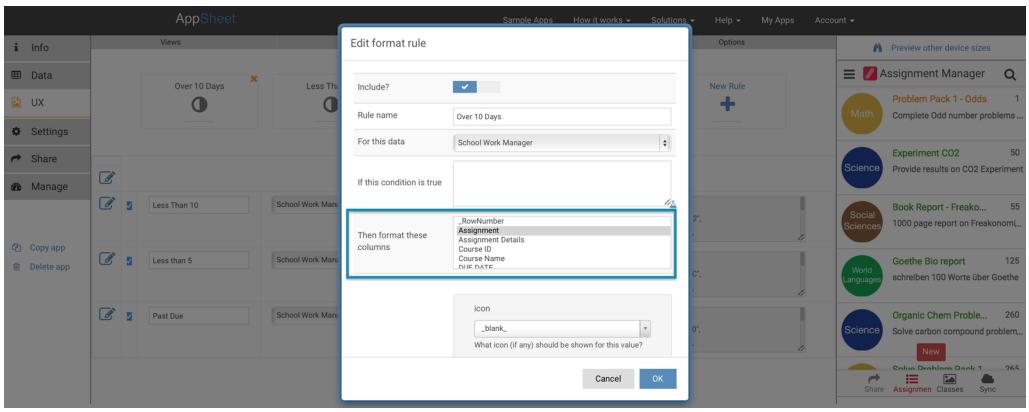


When you create a new rule, the first thing you need to choose is which table the rule applies to in the "For this data" dropdown.



Then specify a condition that must be satisfied for the formatting rule to take effect. If you leave this blank, the rule applies to all rows in the table. However, you could specify a simple formula like [Country] = "USA" or a more complex formula for richer behaviors.

Next, choose the columns that you want formatted. You can hold the 'Ctrl' key to select multiple columns.



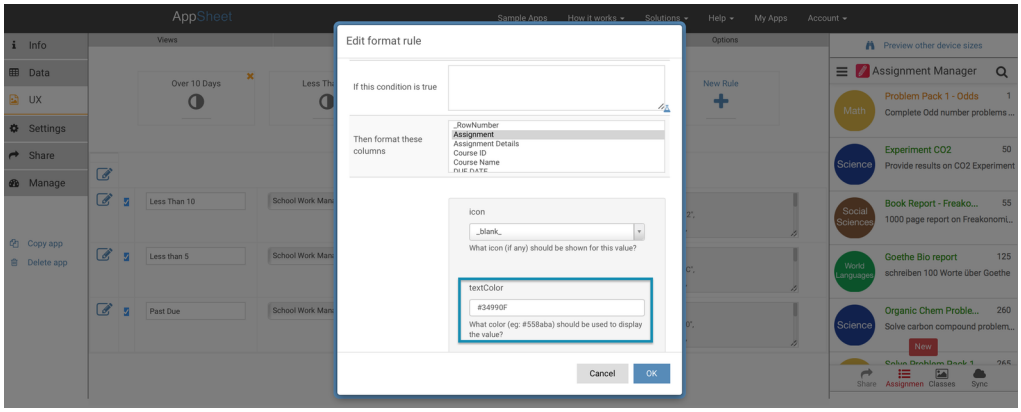
Finally, specify some formatting settings. You will see a list of things you can customize.

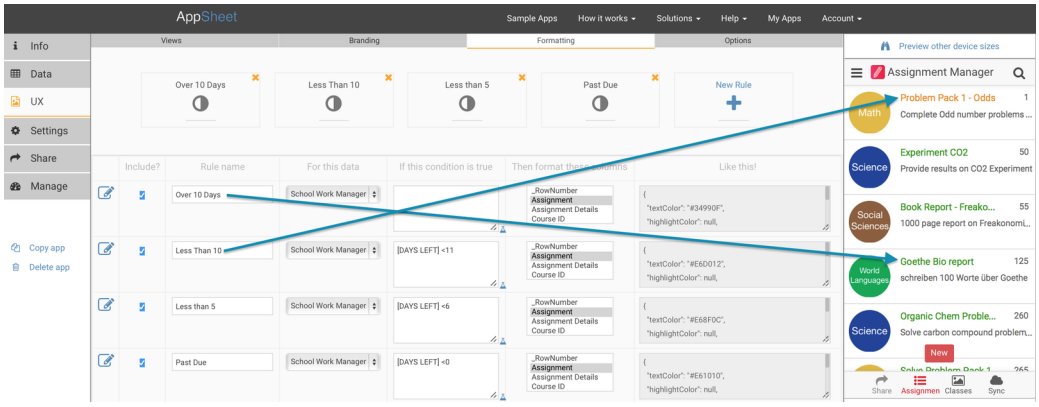
Text Color

Text color and highlight color can be any 'hex' color code, for example this is the color blue: #0F77FF

You can use this tool to find color codes: *http://www.colorpicker.com*

In the *Assignment Manager sample app*, we created four rules-- each one colors the assignment name a different color based on the time left until it's due. You'll see we've specified the color choices in the textColor field.



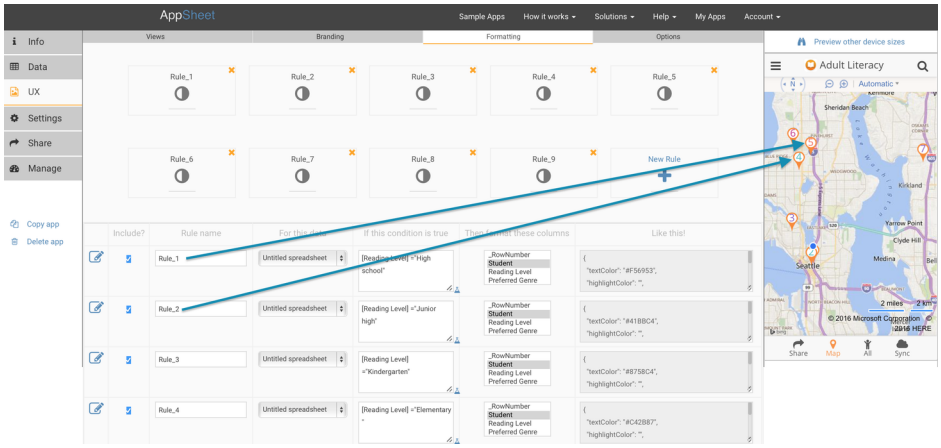


Text Size

Text size can be a number or decimal between 0 and 3 where 3 means the text is 3 times larger than normal and 0.5 means text is half size.

Map Pin Colors

Additionally, adding a highlight color to a geographic type will change the map pin color.

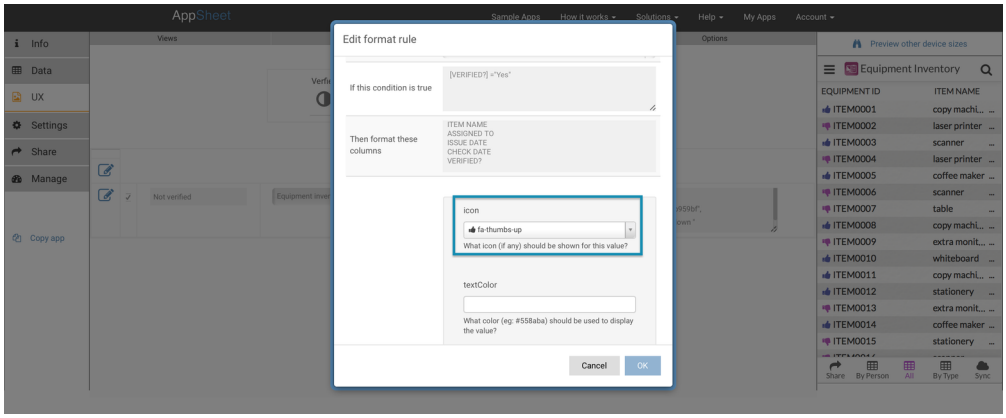


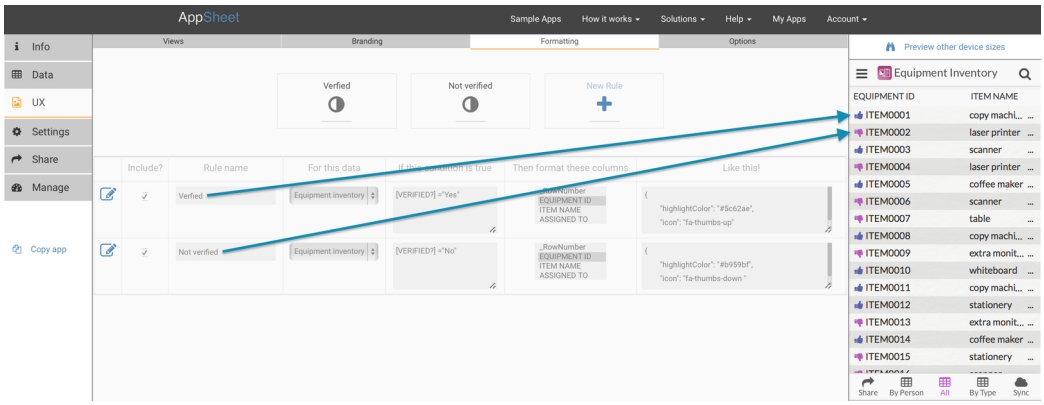
Icons

You can choose to show an icon next to the row, inside the map pin, or on any UX view.

Use the Advanced Editor>UX>Formatting tab to set icons based on rules for the data in your app. Make sure to select any column where you would like the icon featured (choose your location column to feature icons inside map pins instead of numbers). You can use the icon dropdown menu to choose your icons. In our

Equipment Inventory sample, you'll see we've chosen to mark items with a "thumbs-up" or "thumbs-down":





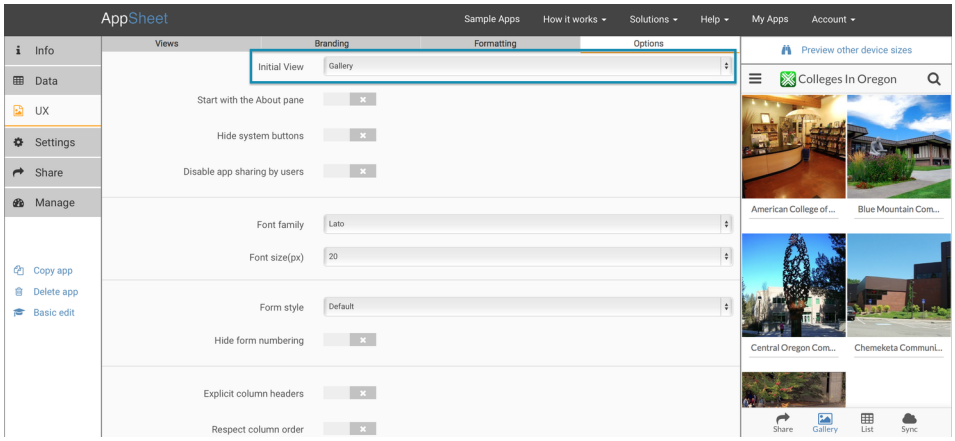
Related articles

- *Multi-page forms with conditional branching*
- *Customizing input forms*
- *Reflecting your brand*
- *How do I control the order of rows displayed in my app?*
- *Working with maps*

Setting the launch page of your app – AppSheet

You can decide which page your app opens up to.

Using the Advanced Editor>UX>Options tab, choose your preferred launch page from the "Initial View" dropdown menu.



Related articles

- *Help content for the app user*
- *Reflecting your brand*
- *Choosing and adding data views*
- *Conditional formatting*
- *Changing UX styles*

Help content for the app user – AppSheet

Users of your app may need information about the app. Every AppSheet app has an 'About' page that is accessed from the hamburger menu at the top left. In some cases it may be helpful to set the About page as the app's initial launch page.

The view has the following information that you can configure via the Basic Editor:

- The name of the app
- The logo
- The description

The view also shows more information that you can configure via the Advanced Editor>Settings>Description tab:

- The full description
- User guidance
- App version
- About Url (the image gets linked to the url you provide)

For example, here's the About view for one of our sample apps:

Related articles

- *Conditional formatting*
- *Choosing and adding data views*
- *Displaying images and documents*
- *Data access for different classes of users*
- *Expressions*

Tutorial Content

How do I control the order of columns displayed in the app? – AppSheet

If you're using the Tabular or Grouped List views, or the Table view, you might want to control the order in which your data columns appear. AppSheet does its best guessing which are most important to show, but you may need to take actions if you'd like to see columns in a different order in your app. We'll describe how to manually control column order later on in this article.

Note: the size of your device's screen determines how many columns you see represented in a given view.

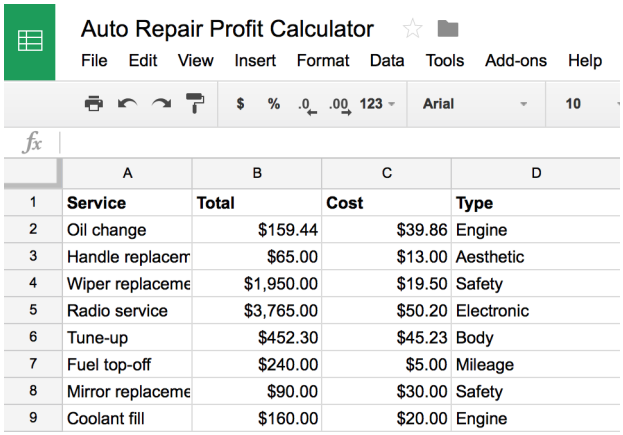
How AppSheet automatically orders columns

Here are the general rules for how AppSheet handles column ordering:

- Rule 1. We always show the key because the key, by definition, uniquely identifies each row. Without the key, it is unclear which data entry is being shown. If the key is hidden (by checking the Hidden checkbox in the Advanced Editor), we don't show the key and instead show the data left to right. *You can read more about how to pick an appropriate key here.*
- Rule 2. Next we add any fields that call for 'actionable' icons (eg: a phone number, email address, etc). AppSheet assumes these items are things you'd want a 'quick click' action on. If Explicit column headers are enabled (in the UX>Style tab of the Advanced Editor), we don't show actionable icons.
- Rule 3. Then we proceed left to right in spreadsheet column order-- if some columns are more important to you, please move them to the left in the spreadsheet.
- Rule 4. Many spreadsheets have a rightmost column that is computed by a formula-- eg: cost of items. If we recognize this pattern, we make sure to include this column.

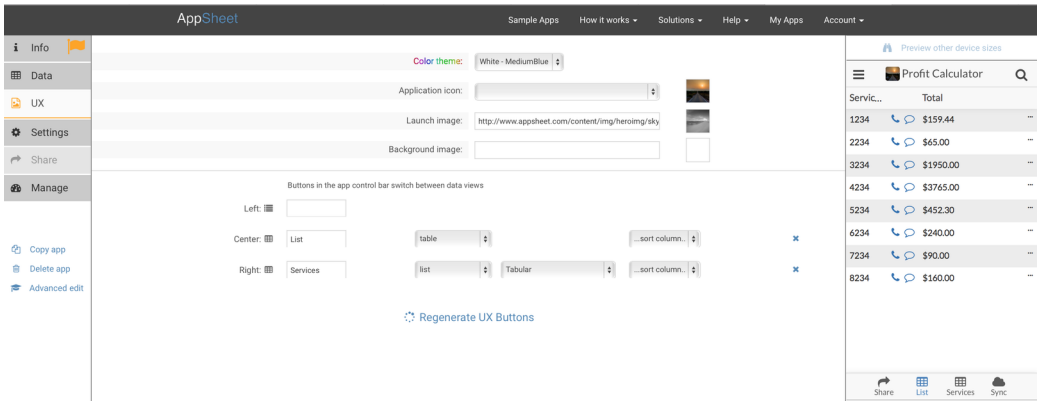
Let's see how this works in a real app built with AppSheet: a cost estimator put together by a fictional auto repair shop to calculate the resulting total profit from jobs done in one week.

First, let's take a look at the original spreadsheet-- there are four columns: Service, Total, Cost, and Type.



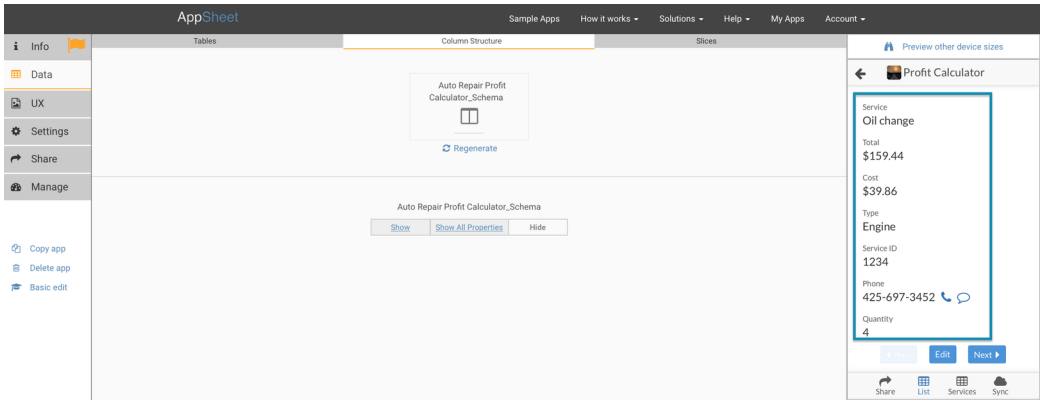
	A	B	C	D
1	**Service**	**Total**	**Cost**	**Type**
2	Oil change	$159.44	$39.86	Engine
3	Handle replacem	$65.00	$13.00	Aesthetic
4	Wiper replaceme	$1,950.00	$19.50	Safety
5	Radio service	$3,765.00	$50.20	Electronic
6	Tune-up	$452.30	$45.23	Body
7	Fuel top-off	$240.00	$5.00	Mileage
8	Mirror replaceme	$90.00	$30.00	Safety
9	Coolant fill	$160.00	$20.00	Engine

Once the app is created from the spreadsheet, either through AppSheet's Google Sheets add-on or straight from AppSheet.com, you can now view it live in the app emulator.



You'll note that my leftmost column, Service, is showing first in the app. Since the Service column is the first column in my spreadsheet *and* the items within the column don't repeat twice, AppSheet chose Service as the *key* for my app--so that's what shows first.

If you click into any of the entries in the app, you'll be able to see the rest of the columns and data.



Manually ordering columns

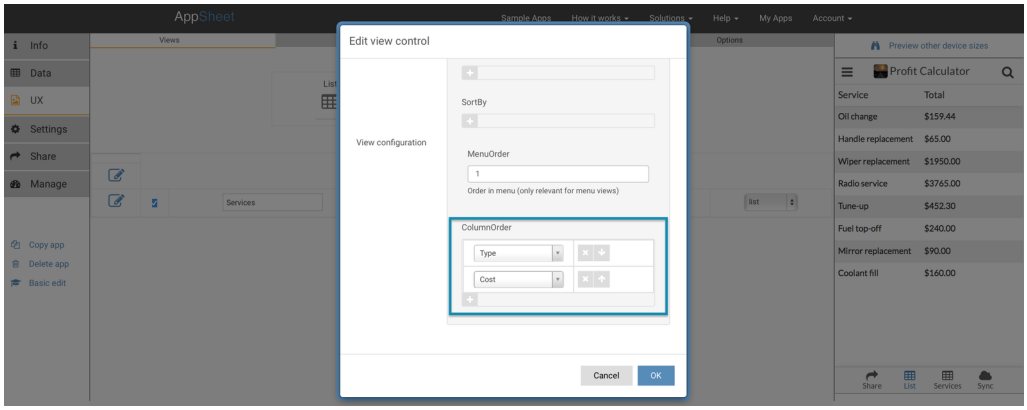
Our goal is to do our best at guessing your preferred column order based on how your spreadsheet is constructed. However, there may be some cases where you wish to manually decide which columns are displayed in the main view. You can manually reorder columns with the **Table UX view type**.

To do this, go to the Advanced Editor>UX>Views tab. Click the edit icon next to the view whose columns you want to reorder.

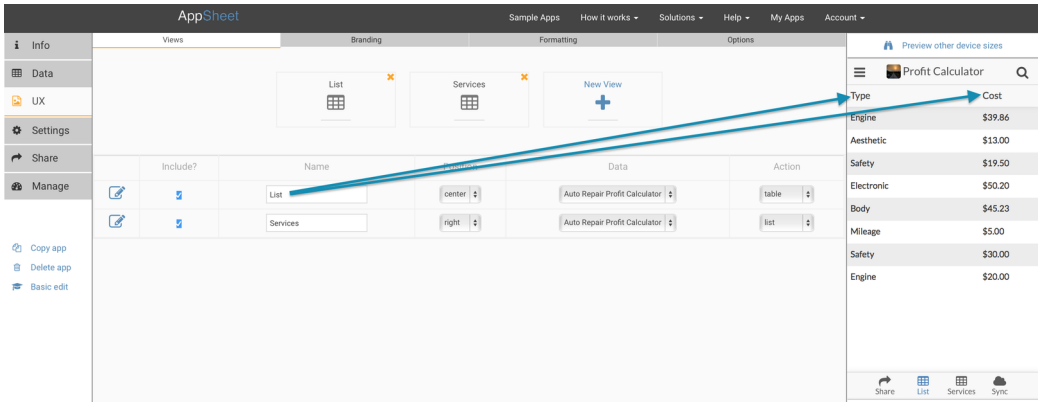
Let's see how this works in the Profit Calculator app.

Originally, my columns were ordered according to my spreadsheet, with "Service" first and then "Total" next. But let's say that I would like to instead see "Type" and "Cost" first.

To do this, in the Advanced Editor>UX>Views tab, I'll first click the edit icon next to the view I want to edit. Then, in the ColumnOrder field, I'll click the "+" button for each column that I want to see in the main view.



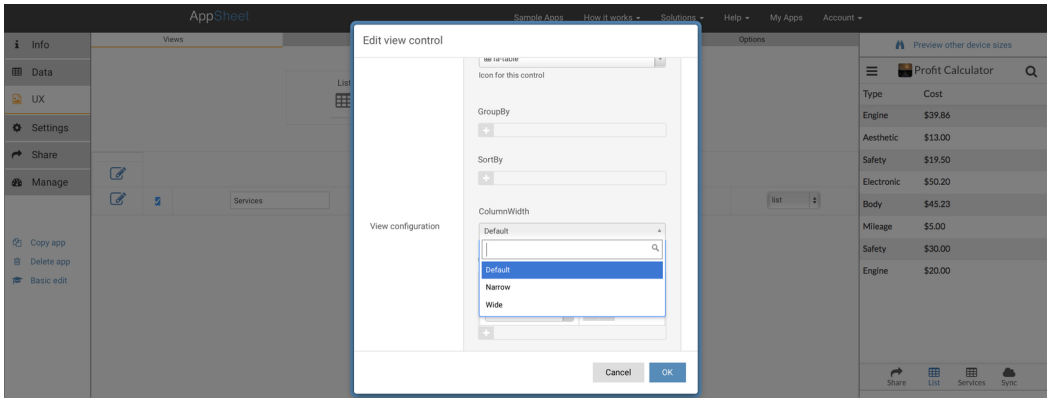
Now, the main Table view of my app is ordered the way I wish:



Controlling column width

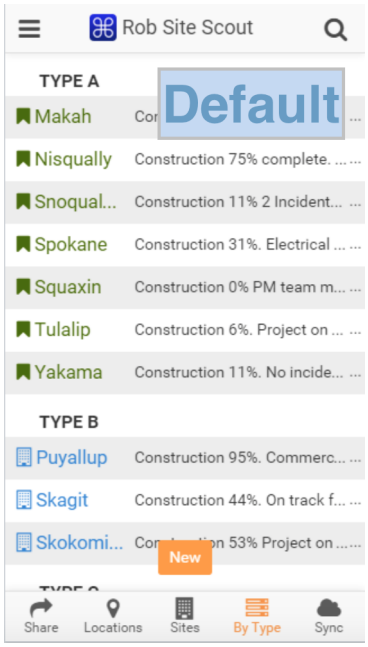
AppSheet doesn't let app creators manually set column sizes due to the wide variety of phone and tablet screen sizes. Instead we provide an option to control how wide the columns are.

While editing a view in the Advanced Editor>UX>Views tab. There is an option named ColumnWidth.:

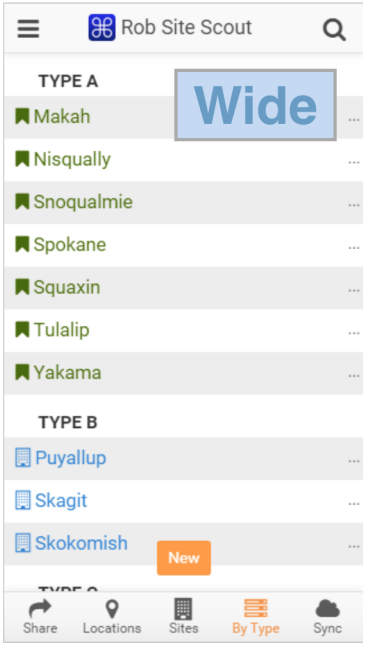


ColumnWidth has three settings:

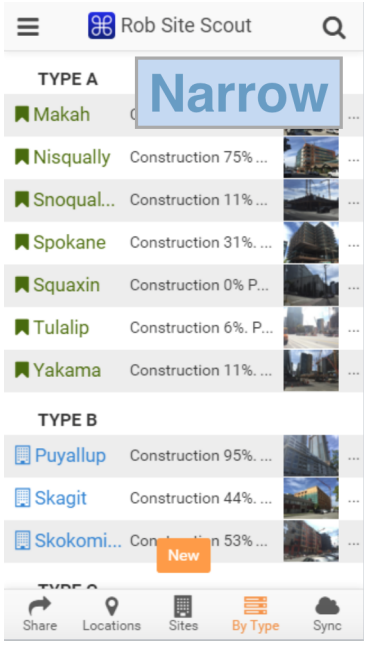
- **Default**-- Columns will be automatically sized based on the size of the row content and column type.



- **Wide**-- Columns will be automatically sized but will be given some extra space. Use this option if you're having trouble with some columns being truncated.



- **Narrow**-- Columns will be automatically sized but will be smaller than they otherwise would be. Use this option to fit more columns onto small phone screens by truncating more text than normal.



Related articles

- *Specifying a key*
- *How should I use multiple sheets in my app?*
- *Choosing and adding data views*
- *Reflecting your brand*
- *Change alerts and workflows*

How do I control the order of rows displayed in my app? – AppSheet

For most of the presentation view types (except for Map and Chart), it is possible to specify a sort order-- i.e. a column which tells AppSheet how to order the data.

In the Basic Editor, the ordering column is picked via a dropdown associated with each view. By default, the order is assumed to be Ascending (i.e. "AAA" is shown before "ZZZ") but you could choose to sort Descending instead (i.e. "ZZZ" is shown before "AAA"). To do this, simply scroll the dropdown to choose the column name with a down arrow in front of it.

In the Advanced Editor, you achieve the same outcome by editing the view properties.

The Table presentation view additionally provides richer options to *group the data* as well by one or more columns.

Related articles

- *How do I control the order of columns displayed in the app?*
- *Presentation types*
- *Conditional formatting*
- *Working with maps*
- *Multi-page forms with conditional branching*

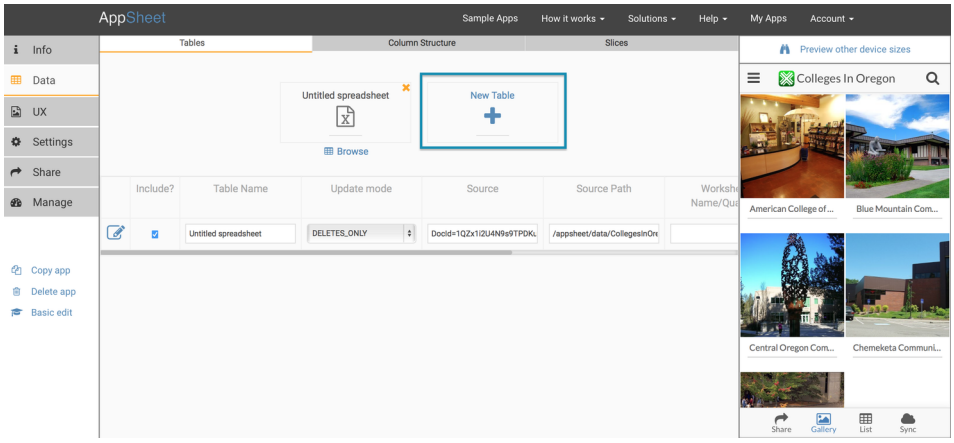
App Definition: Advanced Features

Rich data definition

Using multiple spreadsheets – AppSheet

Richer apps can use more than one table or spreadsheet data set.

In the Advanced Editor>Data>Tables tab, click on '+ Table' to choose another spreadsheet. If you need to use a worksheet from the same workbook you are already using in your app, simply define the sheet name (case included) in the Worksheet Name/Qualifier field. The sheet automatically gets added to your app when you save your changes. You can now define control views over this data as well.



See a good example of multiple tables and cross-table references in the *Order Capture sample.*

Related articles

- *References between tables*
- *How should I use multiple sheets in my app?*
- *Advanced app customizations*
- *Multi-page forms with conditional branching*
- *Using a specific worksheet*

References between tables – AppSheet

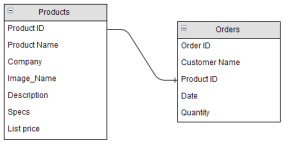
Once you have an app with *multiple spreadsheets* it's useful to create connections or "References" between the sheets and slices.

Columns with a Ref column type always point to the ***Key*** field in the table they are referencing. If you need other columns besides the reference key field, consider using a De-reference (See *Expressions*).

A reference example, the *Order Capture sample app* has a spreadsheet of products in inventory and a separate spreadsheet of orders. The products spreadsheet is kept up to date with all the products in the warehouse and when someone buys a product a new row is added to the orders spreadsheet.

The problem is when a new order is created, the product information from the products spreadsheet has to be manually copied into the orders spreadsheet.

This is a great place to use References. Since each row in the orders spreadsheet has one product, we can add a column with the product id to the orders spreadsheet. This connection between the spreadsheets will be used a in few different areas of AppSheet to make your app much more convenient and powerful.



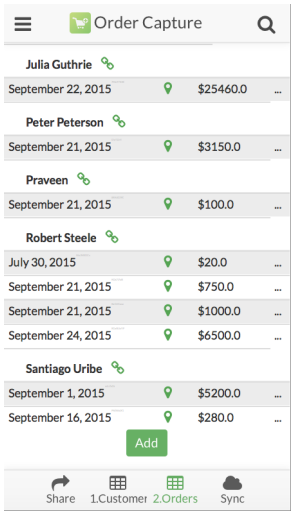
Creating references

AppSheet will automatically create references between tables if there is a column name that contains the name of another table in your app, followed by the name of a column in that table. For example if you have a Customers table that has a Name column, another sheet that has a column named Customer Name will automatically be a Reference when the row is added.

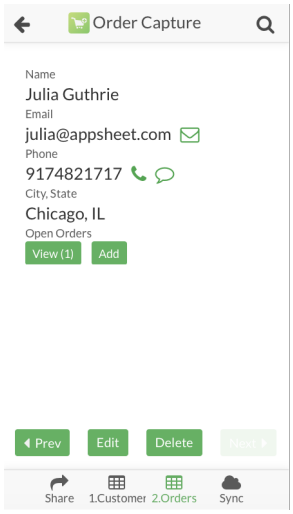
Otherwise, you can create and modify references manually using the Advanced Editor by setting the Field Type of a column to Ref and putting the name of the spreadsheet to reference in the ReferencedType field.

Using references

Appsheet will automatically turn References into links that you can select to navigate to the referenced record.



Additionally any record that is linked to will have a View and Add button.



Selecting the View button will show a list of items with the matching Reference. In this case it will show the product in this order.

Selecting the Add button will open a form to create a new order with this product.

Refs as views in the app

If you'd like to see a graph, map, or customize the view that you are taken to when selecting View, you can create a new View in the Advanced Editor>UX.Views tab and set the Position to ref.

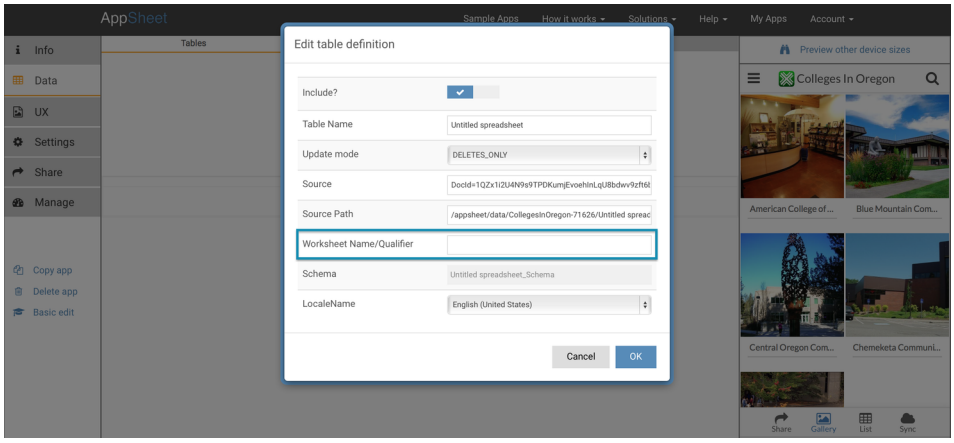
Related articles

- *Expressions*
- *App formulas*
- *Using a specific worksheet*
- *Presentation types*
- *Conditional formatting*

Using a specific worksheet – AppSheet

Instead of using the first worksheet in a spreadsheet file, you can specify a specific worksheet by name.

In the Advanced Editor>Data>Tables pane, when you look at the details for the Tables in the app, you can provide a specific worksheet name in the 'Worksheet Name/Qualifier' field.



If a worksheet with this name is not found in the referenced spreadsheet, AppSheet will default to the first worksheet.

Related articles

- *How should I use multiple sheets in my app?*
- *Using multiple slices*
- *References between tables*
- *Multiple data sources*
- *Column types*

Using multiple slices – AppSheet

Richer apps can use more than one table slice.

In the Advanced Editor>Data>Slices tab, click on 'New Slice' to add another data slice. It automatically gets added to your app when you save your changes. You can now define control views over this slice as well.

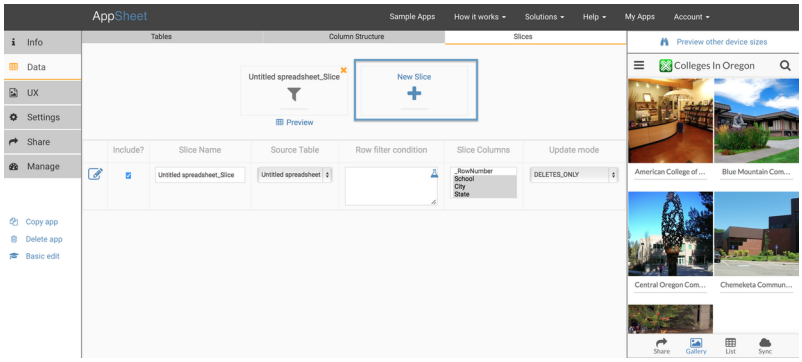


Table slices have a source table, an optional row filter condition, and optionally specify a subset of columns.

Please be careful editing table slice definitions in the Advanced Editor. Slices without a source table or with an invalid source table name will lead to errors in your app.

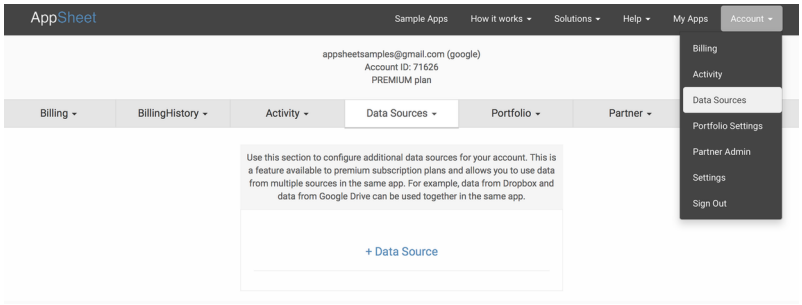
Related articles

- *Defining and using slices*
- *Multiple data sources*
- *Slices*
- *App formulas*
- *Choosing and adding data views*

Multiple data sources – AppSheet

Your AppSheet apps can access data from multiple data sources. The AppSheet account always has a primary data source used for account authentication (for many of our customers, this is Google Drive) and the spreadsheets from that account can be used in your apps. However, you can add other data sources (eg: Office 365, Smartsheet, Dropbox, Box, and more).

To add a new data source, go to your Account tab where you can examine your current data sources and add or remove sources. Adding a source is as simple as providing authentication.



All the data sources in your account are available when you add tables to your AppSheet app. In fact, the same app can combine tables from multiple data sources. This becomes a really powerful mechanism to "mashup" information sitting in different cloud-based data sources.

Related articles

- *Using multiple slices*
- *How should I use multiple sheets in my app?*
- *Offline behavior*
- *Locale support in AppSheet*
- *Viewing external content*

Controlling system behaviors

Locale support in AppSheet – AppSheet

Locale vs Device Settings

Data Locale

See the **Advanced Editor > Data > Tables > Data Locale** setting. This setting is designed to align with the settings in your spreadsheet, so that AppSheet reads the information in your spreadsheet correctly and so that AppSheet returns data to your spreadsheet correctly. The focus is 1) parsing numbers correctly with commas and decimals for Field Types of **Decimal, Price,** and **Percent** and 2) reporting dates and times correctly for Field Types of **Date, DateTime,** and **Time.** You may have one locale setting per spreadsheet and one Data Local setting per corresponding AppSheet table.

Note: **Data Locale** does not impact **Price Currency** settings and AppSheet does not provide currency conversions. Data should be entered in your spreadsheet/app according to the currency you would like displayed. See *How do I convert from US Dollars to my local Currency*.

Locale for Google Sheets

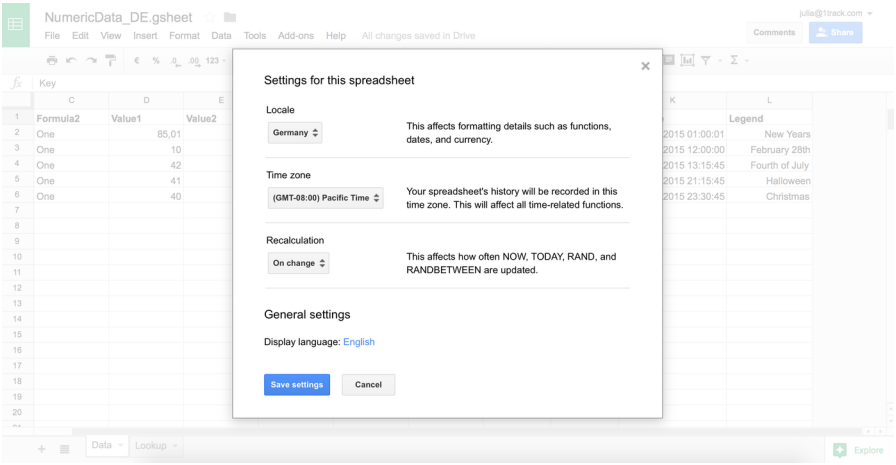
The Google Sheets Locale setting controls how Date, DateTime, Decimal, Price, Percent, and Time values are entered into Google Sheets. For example, when the Locale is United States, United Kingdom or Japan, you must enter a period as the decimal separator between the whole and fractional parts of Decimal, Price, and Percent values. Conversely, when the Locale is Germany, France, or Brazil, you must enter a comma as the decimal separator.

If your AppSheet application uses Google Sheets or Google Forms and specifies a Locale other than United States, you must specify the corresponding Locale in AppSheet. This is necessary because when AppSheet adds or updates a Date, DateTime, Decimal, Price, Percent, or Time value in a Google Sheet, it must use the appropriate data format based upon the Google Sheets Locale setting. Unfortunately, Google Sheets does not provide a way for AppSheet to retrieve the Google Sheets Locale setting automatically. Instead, we ask you to specify the Locale setting through the Advanced Editor.

You can check the Locale assigned to your Google Sheet by clicking the Google Sheet File menu, selecting "Spreadsheet settings", and viewing the Locale setting. If the Locale is anything other than United States, you must specify a Locale in AppSheet.

Configuring the Locale

1. Make certain that your Google Sheet specifies the appropriate locale. Do this by opening the Google Sheet. From the "File" menu select the "Spreadsheet settings". On the "Settings" dialog set "Locale" to your locale. For example, in Thailand select "Thailand".

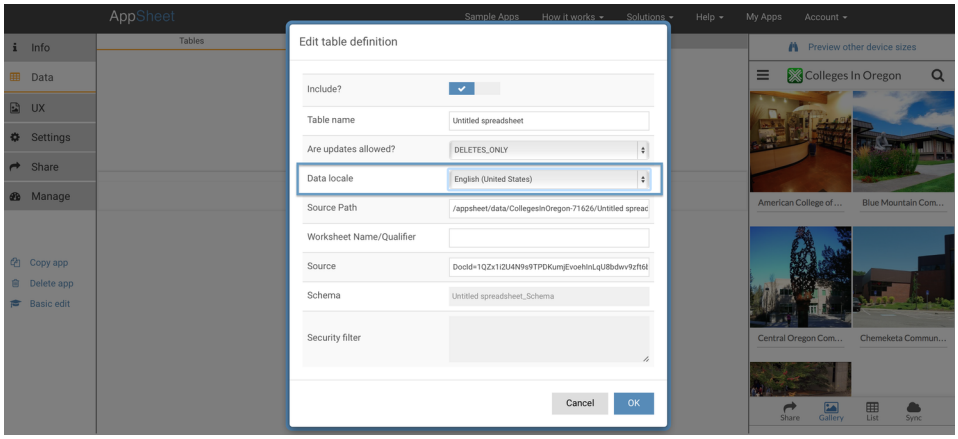


2. Make certain that each Date, Time, DateTime, Number, Currency, and Percent value in the Google Sheet is formatted appropriately. Do this by selecting all of the cells in the column containing the data values. From the

"Format" menu select "Number" and then the appropriate formatting style. For example, for date values select "Date", for currency values select "Currency", for time values select "Time", and so forth. Do the same for each Date, Time, DateTime, Number, Currency, and Percent column.

3. Open your AppSheet app in the Advanced Editor and specify the locale for each table.

- Open your app.
- Go to the Advanced Editor>Data>Tables tab.
- Click the 'Edit this table definition' button to the left of each table name.
- In the Edit table definition dialog window, pick the appropriate Locale setting from the Locale dropdown. You should pick the Locale that most closely matches your Google Sheet Locale setting. For example, in Thailand select "Thai (Thailand)".
- Click the Save button at the bottom of the Edit table definition dialog window.
- Click the Save button at the top right of the Advanced Editor window.
- If your application includes multiple tables, repeat this for each of your tables.



4. Make certain that your browser or device is set to use your locale. For example, in Thailand select the Thai locale. All data is sent between the AppSheet client and the AppSheet server in a common universal format. The browser or device setting completely determines how data values are displayed on your browser or device.

5. If you use a calendar other than the Gregorian calendar, make certain that your browser or device is set to use that calendar. For example, in Thailand select the "Buddist" calender. Many countries use the Gregorian calendar, so in many cases you can skip this step.

6. Click the "Sync" button in the application to read the latest values from the Google Sheet. See if the Date, DateTime, Decimal, Price, Percent, and Time values are displayed correctly in the AppSheet application. If not, verify your browser or device locale settings.

7. Try updating a Date, DateTime, Decimal, Price, Percent, or Time value and saving the changes to the server. See if the correct values appear in the Google Sheet and the AppSheet application.

Compatibility Locale

One of the Locale values appearing in the Locale dropdown menu is the Compatibility Locale. This value is present for backward compatibility only. It preserves AppSheet's existing but limited Locale behavior. If you choose Compatibility Locale, we use a period as the decimal separator between the whole and fractional parts of Decimal, Price, and Percent values. We use simple rules for saving Date, DateTime, and Time values. We save formulas in the United States Locale format, which works for many Locales that use a period for the decimal separator. However, we recommend that you pick the specific AppSheet Locale that matches your Google Sheet Locale. If you Google Sheet does not specify a Locale, we recommend you specify AppSheet Locale United States.

We currently support approximately 70 Locale values. We selected these locales based upon the Locales that Google Sheets currently supports. Please let us know if a Locale you need is missing from AppSheet.

Locale for Excel

It is not necessary to set the AppSheet Locale for AppSheet applications that store data in Excel files on Box, Dropbox, Google Drive, Office365, or OneDrive. This is a consequence of the way we add and update data and formula values in Excel files. The AppSheet table Locale should be set to either United States or Compatibility.

Locale for Smartsheet

It is not necessary to set the AppSheet Locale for AppSheet applications that store data in Smartsheet. This is a consequence of the way we add and update data and formula values in Smartsheet. The AppSheet table Locale should be set to either United States or Compatibility.

Note: Changing the Locale in AppSheet doesn't retroactively change the format of previous entries, nor does it necessarily reflect accordingly in the app emulator. What's displayed in the app emulator is based on the locale settings of your mobile phone or web browser. The Locale setting exists purely to affect the data that's pushed back to the spreadsheet.

Related articles

- *Column types*
- *App formulas*
- *Change alerts and workflows*
- *Virtual columns*
- *How do I change a Column Type?*

Offline behavior – AppSheet

AppSheet apps are designed for offline function. There are four elements to offline behavior as described below, and they are controlled by the app creator. AppSheet apps can work offline because the information needed to run the app (the app definition, the data, and optionally images) is maintained locally on the mobile device.

One important pre-condition for all offline access is that the app must initially be launched on the device when it is online. The process of launching the app on the device causes the appropriate information to be downloaded and cached locally.

Offline behavior is controlled via the Advanced Editor>Settings>Content tab. The use of these features does require the Premium subscription plan.

Delayed sync: 'Sync' is the step that sends data updates from the device to the backend and brings back the latest app definition and data from the backend. If you choose Delayed Sync for your app, it indicates that the app will not sync data immediately when a new edit/delete/add occurs. Rather, it will just be queued up until you explicitly choose to sync. You should select this option for apps that are expected to work in offline environments or where you want to control the use of data bandwidth on the device.

Intermittent connectivity: Apps that don't enable Delayed Sync will synchronize changes every time that data changes on the device. However, there may be connectivity issues that prevent this from happening. Even in a mostly connected environment, there can be intermittent network outages. If such failures occur, AppSheet automatically queues up the changes that fail to send-- in other words, the app will default to a delayed sync behavior if the attempt to sync fails. This is the default behavior and prevents loss of data. Furthermore, repeated retries will not cause any duplication or corruption of data (except for the special case where you use RowNumber as your key-- please read our warning article about that).

Viewing content offline: All spreadsheet/table data content is always copied locally on the device so that it is available offline by default. However, the images and documents associated with the app and data are not (these are typically much larger and so we need do not maintain a local copy). You can explicitly enable a setting that instructs AppSheet to make an offline copy of all image and document content. As with all other offline data caching, this copy occurs when the app is initially run on the mobile device (online) and is subsequently checked and refreshed as needed when Sync occurs.

Launching offline: If you have used the lightweight deployment mechanism, you are familiar with launching apps from their homescreen icons. By default, a mobile device needs to be online in order to launch an app from its

homescreen icon. Once launched, the app can then function offline or with intermittent connectivity. However, if you need the app to launch when the device is offline, this is an option you must explicitly set in the app definition.

To do this, go to the Settings tab in either of the editors and choose the 'launch offline' option. If you have already installed your app to your home screen, you will then need to uninstall and reinstall the app in order to launch it from the home screen icon. Otherwise you'll need to launch it from the AppSheet app itself.

Related articles

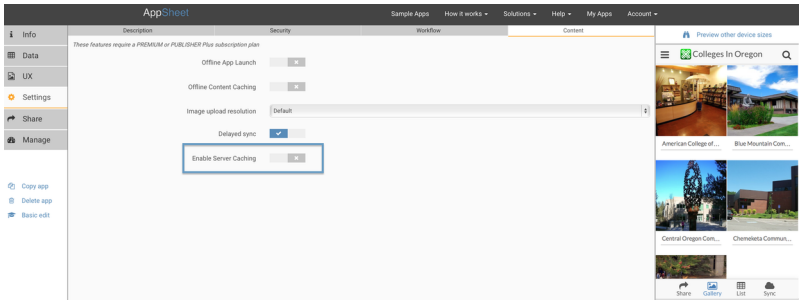
- *Plan upgrade required*
- *Reflecting your brand*
- *Displaying images and documents*
- *Advanced app customizations*
- *Check your app for deployment*

Content caching on the AppSheet server – AppSheet

You can modify the way AppSheet caches content on both server and mobile device.

By default, AppSheet does not cache spreadsheet data on the server. However, server-side data caching can significantly improve the speed perceived by app users during a Sync. Explained in simple terms, the mobile app can get the data directly from AppSheet's servers without having to wait for the data to be fetched from the backend cloud storage platform (Google Drive, Dropbox, etc).

You can explicitly enable server-side data caching via an optional setting in the Advanced Editor>Settings>Content:



This will apply only to read-only data. The data is cached for a variable timeframe at the discretion of the server (but less than an hour). Please be aware that if the sheet has formulas, they will only be calculated when the sheet is actually fetched. As a result, you should not use server-side caching if your data has formulas that have more current content (eg: a formula that accesses an external site or utilizes the current time to compute something).

Related articles

- *Offline behavior*
- *Plan upgrade required*
- *Choosing and adding data views*
- *Slices*
- *App formulas*

Working with your data

Change alerts and workflows - AppSheet

You might want to take actions when data updates occur. Change workflows are the mechanism to do this. Change workflow rules (emails) fire at the AppSheet backend when data changes are made and synced through the AppSheet app running on the mobile device or the desktop browser.

Every change workflow rule is structured as follows:

1. When does the rule fire?

- On specific actions: you can choose Adds, Deletes, Updates or any combination of them.
- On specific tables or slices: if you have an app with multiple tables or slices, you can choose a subset of these to trigger the workflow rule.
- Only *after* the modification has been saved to the spreadsheet, so any formulas in the spreadsheet will be computed on the row before the update email is sent.

2. What action should be performed?

    - At the moment, the only action supported is to send an email. You can customize many aspects of that email as we'll describe below.

This feature set will grow further in functionality over time. The rules can be authored in the Advanced Editor>Settings>Content. Please note that while the app is in test mode (i.e. it has not passed a *Deployment Check*), any messages sent from workflow actions will go only to the app creator.

Let's look closer at each of the components of a workflow rule.

Controlling when a workflow rule fires A workflow rule can be triggered when:

- A new row is added to a table.
- A row is updated in a table.
- A row is deleted from a table.

You may wish to further restrict when a workflow rule is triggered. For example, maybe the workflow rule should only be triggered when a row is changed and the row satisfies a specific condition. For example:

- When a new row is added to the Inspections table, and the value in the InspectionPassed field is false.
- When a row is updated in the Orders slice, and the value in the OrderAmount field is over $500.

We enable this via slices. Typically, you define slices in order to show filtered views of the data. This is a very different use of slices. You can choose to trigger a rule only if the changed row is included in a slice. So in the first example above, you would define a slice that includes only rows where the InspectionPassed field is false. You would then configure your workflow rule to refer to this slice and only take effect when a row is added.

Note that since you can specify multiple change actions and multiple tables or slices in each rule, it is a very powerful mechanism to specify the rule firing condition.

When are workflow rules invoked? The AppSheet workflow rules are evaluated on the AppSheet server when a user does an add, update, or delete through the AppSheet client and then syncs these changes back to the server. When the client syncs the changes to the server, the server first makes the appropriate changes to the spreadsheet and then checks whether any workflow rules should be invoked.

The AppSheet server invokes the workflow rule action if all of the following conditions are satisfied.

1. It first checks whether any workflow rules have been specified for the updated table or slice.
2. If workflow rules are specified for the table or slice, the server checks whether the workflow rule's "Update event" type matches the action the user performed, that is Add, Update, or Delete.
3. Finally, if the workflow rule is specified for a Slice, the server checks whether the slice filter condition is true. The server skips this step, if the workflow rule is associated with a table rather than a slice.

Changes you make directly to the spreadsheet do not go through the AppSheet server, so they do not trigger the AppSheet workflow rules. For example, if you update a Google Sheet directly, Google Sheets does not notify the AppSheet server about the change, so the AppSheet server is completely unaware of the change. When your AppSheet client performs a sync, the AppSheet server will read the Google Sheet and return the updated data values to the client, but AppSheet server is unaware that you updated the Google Sheet directly.

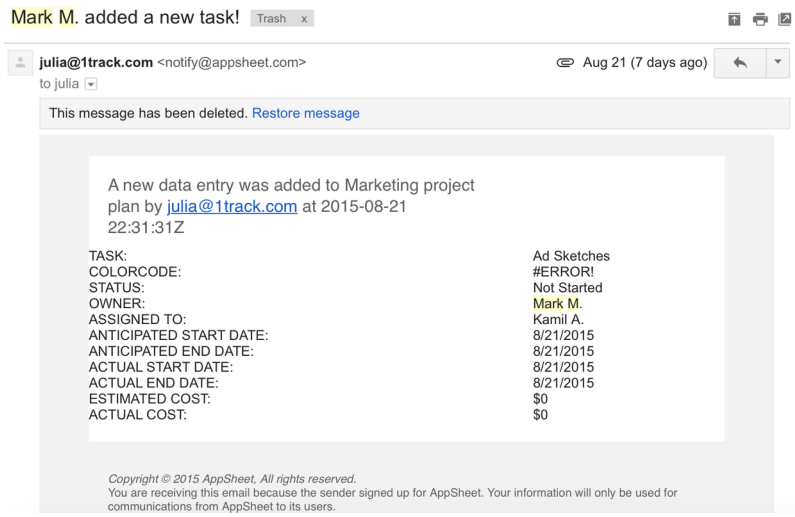
What action is performed? At the moment, the only action supported is sending email. You can configure the following aspects of the email:

- **Send to:** This is a comma-separated list of email addresses or expressions that yield email addresses.

1. You can enter specific email addresses, such as **JohnHughes@gmail.com**. If you enter an email address that contains special characters such as hyphen or plus, you must enclose the email address in quotes, such as "**John-L-Hughes@gmail.com".**

2. You can specify that the email address be taken from a field in the record that is being updated. For example, when a new order is captured, you may wish to send email to the customer who created the order. You can do this by entering an expression specifying the field name containing the customer's email address. For example, if there is a field called "CustomerEmail" in the updated record, you can specify **[CustomerEmail]** in the email address list.
3. You can specify that the email address be taken from a field in a record that is referenced by the record being updated. For example, each of your Order records might contain a reference to a Customer record. Each Customer record might contain the customer's email address. When a new order is captured, you can send email to the customer who created the order. You can do this by entering the name of the Order record field that references the customer record, followed by the name of the Customer record field containing the customer's email address. For example, assume the name of the Order record field that references the customer record is **CustRef**, Assume the name of the Customer record field containing the customer's email address is **CustEmail**. You can specify the customer's email address by entering the expression **[CustRef].[CustEmail]** in the email address list.
4. You can specify that the email addresses be taken from an entire column in another table. For example, you could create a table called **PeopleToInform** having two columns, **Name** and **EmailAddress**. Each time a new order is captured, you can send email to all of the people in the **PeopleToInform** table. Do this by entering the expression **PeopleToInform [EmailAddress]** in the email address list.
5. When you specify multiple email addresses, you can include any combination of specific email addresses and expressions in the email address list. For example, you could specify the following email address list. **JohnHughes@gmail.com,"**John-L-Hughes@gmail.com",[CustomerEmail],[CustRef].[CustEmail],PeopleToInform**[EmailAddress]**

- **Email Subject:** AppSheet uses a default subject, but you can override that with your own subject text. Here, you can define placeholders that are filled in by the values in the updated entry. Here is an example of a subject line: 'Thanks <<CustomerName>> for your order!'
- **Email body:** You have three options when it comes to the body of the email. You can customize the email body field or add a body template (as described below), or simply do nothing and receive a default email that contains **all** of the values for the updated row. Here's what it looks like if you do not choose to customize the email body or provide a body template:



- **Email body:** May contain template variables. Use this field when you have a simple email body.
- **Body template:** For more complex workflow emails, you can specify a Google doc that may contain text, images, and placeholders (see next section about placeholders). After the placeholders are replaced the resulting document becomes the email body. Use this field when you have a more complex email body. When a Body template is present, it is used in lieu of the €œEmail body€.
- **Attachment template:** You can optionally specify a template file to use as an email attachment. The file itself is chosen from your cloud file system (at the moment, we only support Google Drive for this feature and the template file is a Google Doc). In your document, or in the subject line, for example, you may want to include placeholders that correspond to your column headers, and that will be replaced when actual data updates are

made i.e. "<<CustomerName>> has been added to your app". When an email is actually sent, it will carry a PDF attachment that is a copy of this file with the placeholders replaced by actual values from the updated entry.

- **Attachment Name:** The name given to the email attachment. The default AttachmentName is €œChangeReport€. You may specify a different name and may include placeholders in the AttachmentName.
- **CC:** Email "CC value. May contain one or more email addresses. Multiple addresses must be separated by commas. You can enter both specific email addresses and expressions as described with "Send to".
- **BCC:** Email "BCC" value. May contain one or more email addresses. Multiple addresses must be separated by commas. You can enter both specific email addresses and expressions as described with "Send to".
- **Reply to:** Email "reply to" value. May contain one email address. You can enter either a specific email address or an expression as described with "Send to".
- **PreHeader:** The default PreHeader value is "'<<_UPDATEMODE>> to application '<<_APPNAME>>' table '<<_TABLENAME>>' by '<<_USERNAME>>' at '<<_NOW>>'". This would yield a PreHeader such as "Update to application 'Workflow' by 'Julie Morgan' at 8/26/2015 6:12:28 PM". You may customize the PreHeader and may include placeholders.

Over time, we intend to provide a richer set of actions including the ability to post to external systems, to utilize other communication channels, and to make other data updates. In particular, this could trigger a chain of workflow rules leading to rich, multi-step behaviors. **Placeholders** You can specify a variety of placeholders in your email subject, body, PreHeader, attachment name, and templates that will populate with the correct corresponding data when the email is fired. Here's a list of AppSheet's built-in placeholders-- note that each one carries a preceding mandatory underscore.

- **<<_APPID>>:** Application GUID (Globally Unique Identifier) that identifies the AppTemplate e.g. €œ8c26466f-1db0-4032-9c0f-40c2a588cf50€.
- **<<_APPNAME>>:** The name of the AppSheet application e.g. €œWorkflow-10301€.
- **<<_APPOWNER>>:** The Owner Id of the AppSheet application e.g. €œ10301€.
- **<<_ATTACHMENTNAME>>:** The name given to the attachment.
- **<<_NOW>>:** The current date and time e.g. "6/15/2009 1:45:30 PM".
- **<<_ROWKEY>>:** The key value of the added, deleted, or updated record.
- **<<_ROW_WEB_LINK>>:** URL that opens the added or updated record in AppSheet. The record key is displayed as the link name.
- **<<_ROW_WEB_URL>>:** URL that opens the added or updated record in AppSheet. The full URL is displayed. Customers will probably use <<_ROW_WEB_LINK>> more commonly.
- **<<_RULENAME>>:** Name of the workflow rule e.g. €œMy Update Rule€.
- **<<_TABLENAME>>:** Name of the table e.g. €œOrders€.
- **<<_TIMENOW>>:** The current time e.g. "1:45:30 PM"
- **<<_TODAY>>:** The current date e.g. "6/15/2009".
- **<<_UPDATEMODE>>:** The name of the operation that triggerred the workflow rule. Namely, €œAdd€, €œDelete€ or €œUpdate€.
- **<<_USEREMAIL>>:** The current user€™s email address e.g. €œjmorgan@google.com€.
- **<<_USERNAME>>:** The current user€™s name e.g. €œJulie Morgan€.

We also allow you to use placeholders that correspond to your column headers.The placeholders must exactly correspond to the column names in your spreadsheet, case included, with the following format: <<ColumnName>>. Do not include underscores as with AppSheet's built-in placeholders.

You can also use any AppSheet expression in a placeholder. This allows you to customize your templates with powerful data-driven logic.

Click here to see an example of how workflow emails work.

- *Advanced app customizations*
- *Printing your data*
- *Slices*
- *How should I use multiple sheets in my app?*
- *How do I control the order of columns displayed in the app?*

Printing your data – AppSheet

Many mobile apps replace paper-based processes, but you may still want to print out the data collected by your app. You can do this in a number of different ways depending on your specific needs:

1. Using the spreadsheet
   - Most spreadsheets have a variety of options to format and print data
   - If you are using Google Sheets, there are add-ons that allow you to format and print your data
2. Using AppSheet
   - You can use the change workflow rules to send emails when changes occur, along with printable attachments
   - If you run AppSheet in fullscreen mode in a browser, you can use the browser Print (Ctrl-P) functionality to print the contents of the current view

Related articles

- *Change alerts and workflows*
- *Integrating with Google Apps (Sheets and Forms)*
- *How can I receive (or send) an email alert when data changes?*
- *Expressions*
- *How do I control the order of columns displayed in the app?*

Integrating with other services

Integrating with Google Apps (Sheets and Forms) – AppSheet

The easiest way to build AppSheet apps from Google Sheets and Forms is to use the AppSheet add-ons found in the Google add-on gallery.

Google Sheets

The *Google Sheets AppSheet add-on* automatically converts a Google Sheet to an AppSheet app. It does this in one step:

1. **Go**: when you click "Go" from the AppSheet add-on pane in your Google Sheet, you are taken to www.appsheet.com to a working mobile app you can customize.

Be aware

Google Sheets allow you to create onEdit triggers-- these are code functions that will run whenever a Google Sheet is edited via Google's web interface. However, these onEdit triggers do not fire when data is edited and synced to the spreadsheet via AppSheet. If you have important functionality that needs to run on updated data, we suggest moving it to a timed trigger (a different kind of trigger that Google supports). You can learn more about these different trigger types via the Google Docs page.

Also, do not use filters on your Google Sheets --- it can make the filtered rows invisible to updates. Instead, use filter views (this is a Google Sheets feature that lets each user create and use their own filters without affecting other users of the sheet).

Google Forms

The *Google Forms AppSheet add-on* automatically converts a Google Form to an AppSheet app. It does this in two steps:

1. **Prepare**: this analyzes the form, takes information from the form questions and adds it to the response spreadsheet in the form of notes on the column headers
2. **Launch**: this creates an AppSheet app from the response spreadsheet.

It is important to understand that the AppSheet service cannot access the form itself directly-- it is only the add-on (a web component running in your browser) that can access the form. This is why the Prepare step is important.

This process has a few important caveats and limitations that you should be aware of:

Adding or moving questions

If you add new questions or move questions around, the order of columns in the response spreadsheet may no longer correspond to the latest form content. You will need to create a new response spreadsheet (via the Responses menu), Prepare, and then Launch again.

Choosing AppSheet types

You may want to use *AppSheet input types* like Photos, Signatures, etc. Those cannot be specified in a Google Form. So what can you do? The simple approach is to specify a question as being of Text type and use a question title that is suggestive of the type (eg" Customer Photo") or ("Manager Signature"), etc. AppSheet tries to automatically guess the type based on these titles. You can also change the type yourself in the Data / Column Structure tab of the AppSheet *Advanced Editor*.

Images and Videos

Google Forms can contain embedded images and videos. Although AppSheet forms can as well, the Forms add-on is unable to extract embedded image and video data during the automatic app creation process. These fields will be given placeholder URLs in the generated app that you should update in the AppSheet *Advanced Editor* with correct URLs pointing to your image or video content.

Form navigation

Google Forms has a powerful mechanism to chain sections together using a "go-to-next" navigation model tied to answers of specific questions. At the end of every section, you can also specify to jump to another section or submit the form.

AppSheet uses a different mechanism. In AppSheet, you can specify a conditional expression that controls whether or not a form page should be shown. The Forms add-on attempts to automatically generate the appropriate expressions based on the structure of your form.

Most forms can now be correctly converted to AppSheet apps automatically, but there are three main limitations:

1. Reverse navigation is not supported by AppSheet. Try to arrange your form such that all navigation proceeds to higher section numbers.
2. In Google Forms you can include a special "Other" option for Multiple choice questions and assign it specific navigation behavior. However, the navigation associated with this choice is not made available to the AppSheet add-on. We recommend avoiding use of "Other" on questions where "Go to section based on answer" is enabled.
3. There is a practical limit to how much branching navigation can occur in a single form. Part of the process of converting to AppSheet's "Show If" navigation model requires building the set of all distinct paths through the form. The total number of paths through a form can quickly become unmanageable when there are many sections with redundant navigation options, such as multiple questions in the same section that can send the user to the same place. For this reason **we strongly recommend using at most one navigating question per section, and marking these required whenever possible**. Note that when more than one navigating question occurs in one section of a Google Form, only the last answered question will affect navigation anyway, and any others will be ignored.

For more information about building conditional branching forms in Google Forms and in AppSheet, see *this article*.

Advanced options

Questions in Google Forms have Advanced options that let you constrain the type of the response. This information is unfortunately not made available to the add-on. However, you can explicitly set up equivalent behaviors in the AppSheet Advanced Editor.

Response summary

Google Forms has a convenient feature that shows a statistical distribution of responses. Unfortunately, this does not capture responses made via the AppSheet app-- although the underlying response spreadsheet has all the responses, the statistics are only measured on changes made through the Google Form itself. However, Google Sheets now has a *great new feature* that serves a similar purpose --- it is called "Explore" and is found via a plus button at the bottom right of the Google Sheet. Click on the Explore button in your response sheet to automatically get insights into your form responses.

Username

In a Google Form, you have an option at the top to collect the username (actually the user email) of the respondent. This causes an extra column 'Username' to be added to the response spreadsheet. Unfortunately, this column may be placed at any position within the spreadsheet and messes up the AppSheet service. We recommend therefore that if you want this capability, enable this option at the very end after completing the authoring of the rest of the form. This adds the Username column at the end and therefore does not cause any issues.

Triggers

If you have formSubmit triggers on the spreadsheet, they do not fire when updates are made via AppSheet. As with onEdit triggers, we recommend moving this logic to a timed trigger.

Related articles

- *Column types*
- *Multi-page forms with conditional branching*
- *Who should use AppSheet and what kind of apps can it create?*
- *How should I use multiple sheets in my app?*
- *Change alerts and workflows*

Integrating with Excel and Office 365 – AppSheet

An easy way to build AppSheet apps from Excel is to use the *AppSheet add-on* found in the Office add-in store.

While this process is mostly seamless, there are some specific behaviors to be aware of. With Office365, if one user has the spreadsheet open for editing, other users and third party apps like AppSheet cannot make updates to the spreadsheet. Consequently, if you intend to build an app that updates the data (as most apps do!), please make sure to close the spreadsheet first. Otherwise, you will see conflict ("409") error messages and the app will not able to save changes to the spreadsheet.

Related articles

- *Integrating with Smartsheet*
- *OneDrive*
- *Integrating with Google Apps (Sheets and Forms)*
- *Make a spreadsheet*
- *References between tables*

Integrating with Smartsheet – AppSheet

Related articles

- *Smartsheet*
- *Create an app*
- *Integrating with Excel and Office 365*
- *How can I receive (or send) an email alert when data changes?*
- *Integrating with Google Apps (Sheets and Forms)*

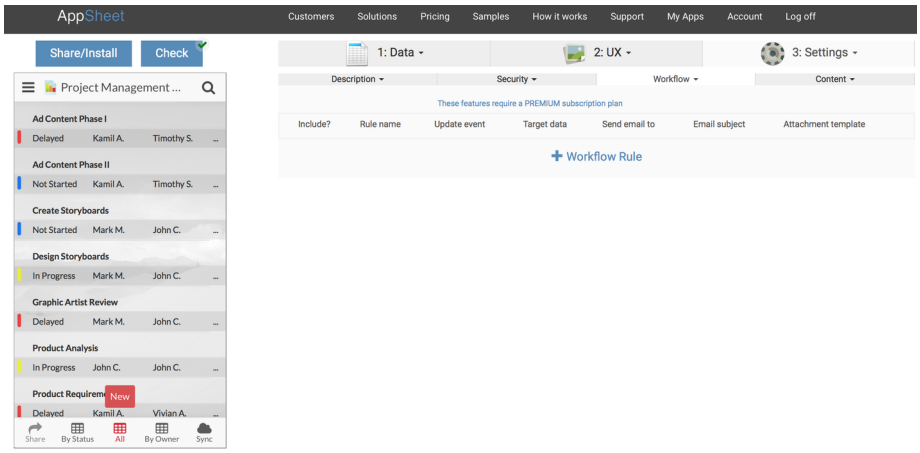
Tutorial Content

How can I receive (or send) an email alert when data changes? – AppSheet

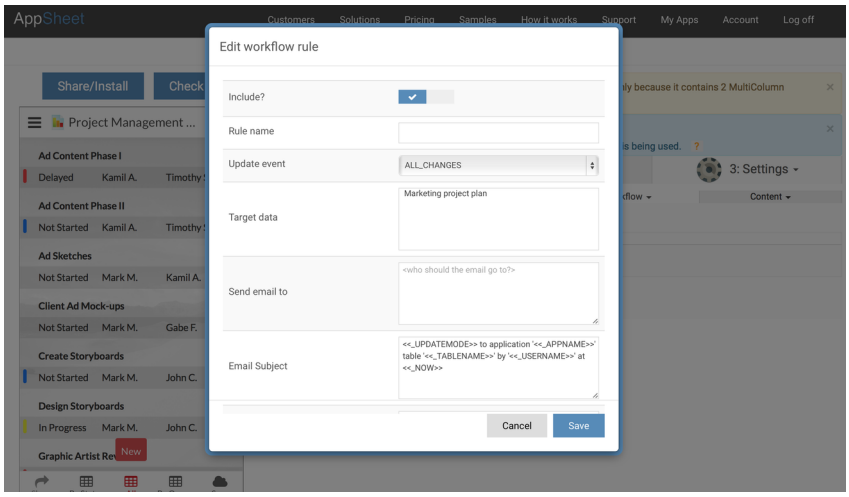
You can opt to receive an email each time data is added, updated, or deleted in the app. The emails fire when the updates are synced from the mobile device.

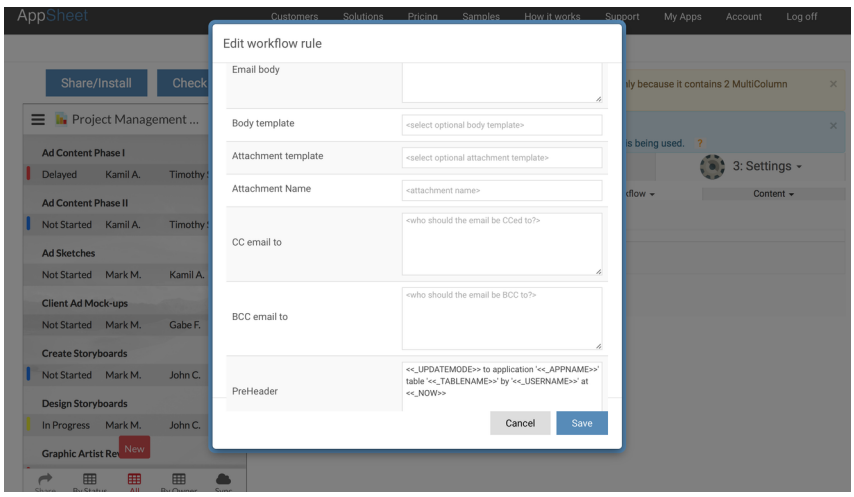
For example, let's say I've created a Project Management app I'm sharing with the rest of my team to track our projects, assigned tasks, and progress to goal. I'd like to receive an email each time someone adds a new task to the app so I am always up to date with new assignments.

I'll need to go to the Advanced Editor>Settings>Workflow.

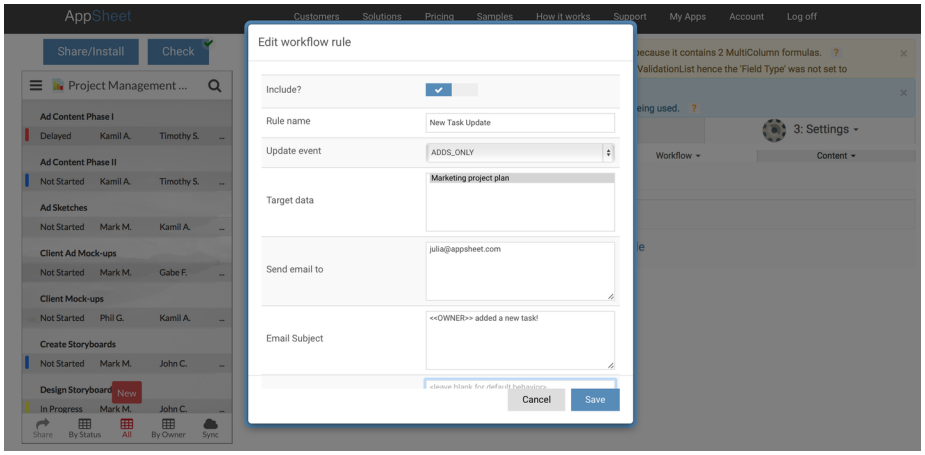


When I click +Workflow Rule, a dialog box appears called Edit workflow rule. There are *several fields* to fill out:

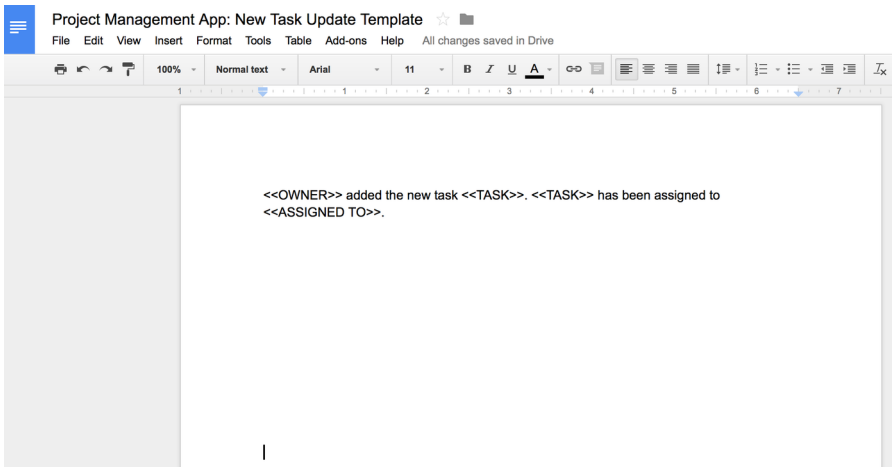




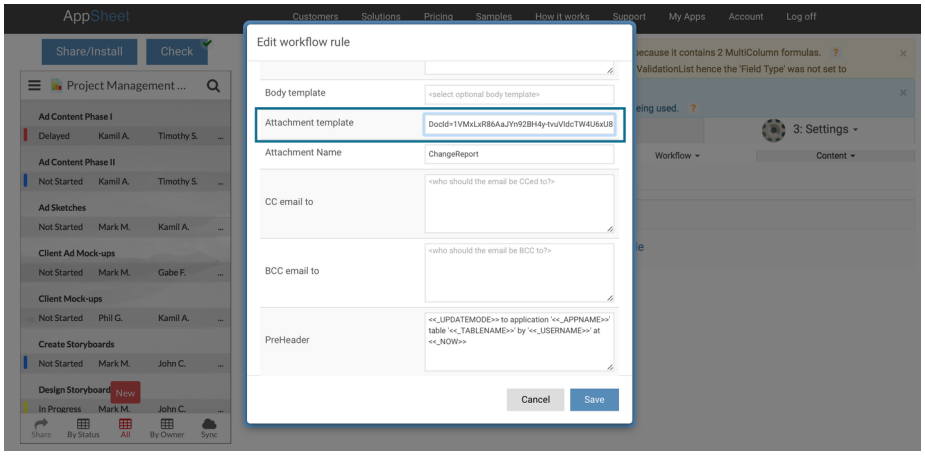
For the email subject and the attachment template, I can use placeholders that will populate with the correct information in the email. The placeholders must directly correspond to the column names in my spreadsheet or to AppSheet's built-in placeholders, case included, with the following format: <<ColumnName>>. (To see the complete list of allowed placeholders, *click here*). You'll see I've done this in the Email subject field:



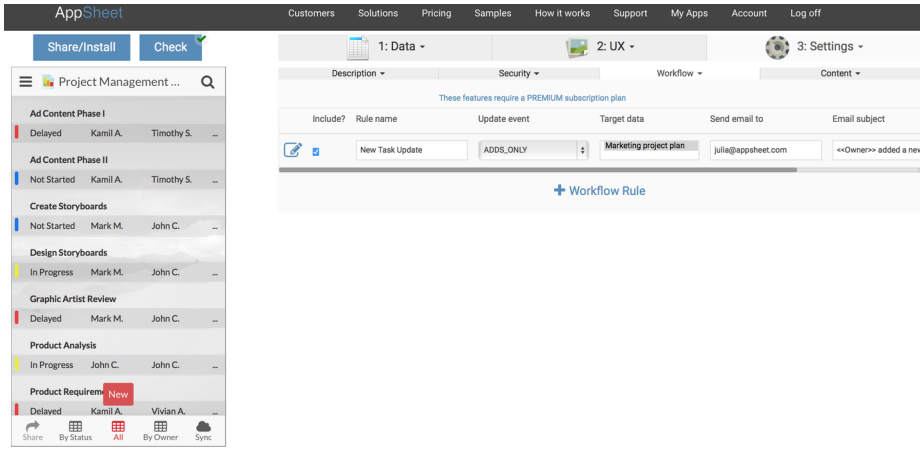
of the email, I instead want a PDF attached to the email that contains the updates. I will need to specify a Google Doc template with the corresponding placeholders in the document. Here's what my Google Doc template looks like:



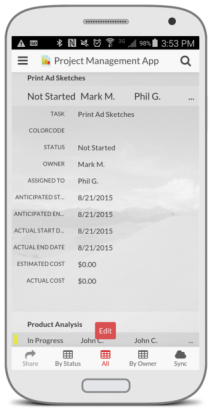
When I click into the Attachment template field, I will get the option to choose my corresponding document from my Google Drive.



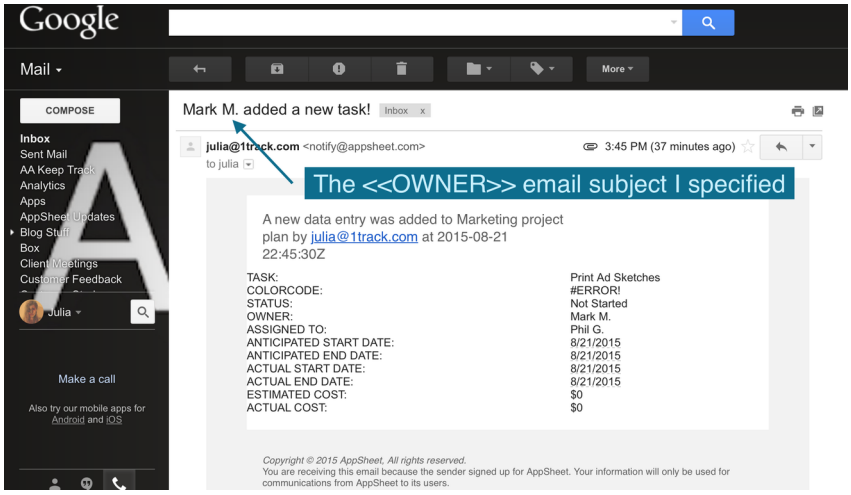
been added:



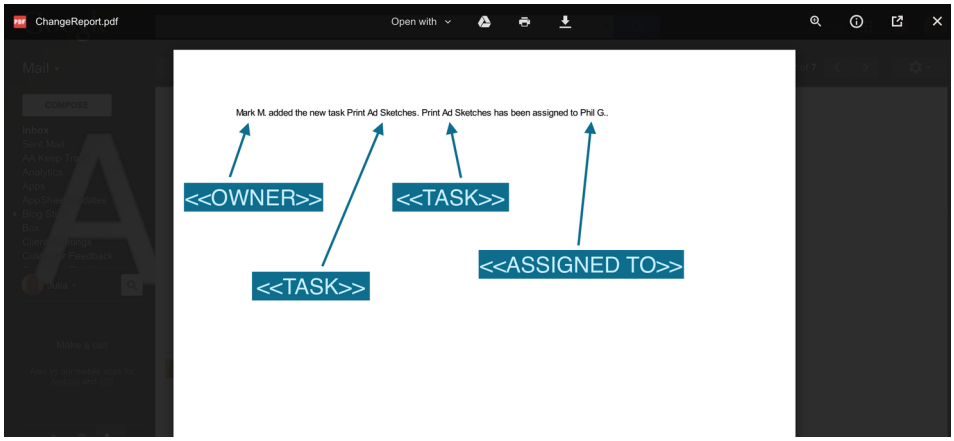
Let's see this in action. In the mobile app, my colleague Mark just added a new task: Print Ad Sketches.



When Mark synced the device, I received the workflow email:



Attached to the email is the PDF with the appropriate data replacing the placeholders:



We've focused on email in this article. Going forward, AppSheet will support a variety of other actions for workflow rules-- including text messages, push notifications, webhooks and more.

Please note that while the app is in test mode (i.e. it has not passed a *Deployment Check*), any messages sent from workflow actions will go only to the app creator.

Related articles

- *Change alerts and workflows*
- *How should I choose a key?*
- *How should I use multiple sheets in my app?*
- *Expressions*
- *Controlling data inputs with column constraints*

How should I use multiple sheets in my app? – AppSheet

You might want to use data from more than one spreadsheet in your AppSheet app. There are two ways to do this. *You can add multiple standalone spreadsheet files as AppSheet tables.* It is also possible to create AppSheet tables from different worksheets within the same workbook.

To be clear on terminology, a single dataset used by AppSheet is called a **table**. The source of a table is a single spreadsheet. A **workbook** is a file that contains one or more worksheets; a **worksheet** is a single spreadsheet that contains cells organized by rows and columns.

Either approach works just fine, depending on your circumstances. The performance of the two options may differ based on the specific content. The intuition is simple-- the AppSheet backend has to download the entire spreadsheet file but if the same file (workbook) is referenced multiple times (via multiple tables referencing individual worksheets in the workbook), the backend can be efficient and download it just once. On the other hand, multiple small files can be fetched and processed efficiently in parallel.

In most cases, you should pick whichever approach is convenient for you to manage. However, if you use only two worksheets out of a spreadsheet file with a hundred large worksheets in it, you'd be better off splitting those two off into their own single files.

Related articles

- *Using multiple spreadsheets*
- *How do I control the order of columns displayed in the app?*
- *Multi-page forms with conditional branching*
- *Who should use AppSheet and what kind of apps can it create?*
- *How do I design a secure app?*

Security and Reliability

Data Security

General concepts – AppSheet

App security matters because your data is exposed through your app. While we follow sound technical practices like encrypting all network communication, you have three simple and understandable mechanisms to control the security of your app.

- Distribution control-- AppSheet is not an app store. We do not let others search for your app or find it in any way. You distribute your app to the users that you choose.
- Access control-- you can require the users of your app to sign in, and can specify a whitelist of users who have permissions to access the app. If you do this, nobody else can access the app.
- Data control-- you can control the subset of data that is visible in the app, and control whether and how it can be updated. Even for the users who can access your app, this limits what they can see and do.

Related articles

- *Understanding authentication and authorization*
- *Understanding the secure flow of data*
- *How do I design a secure app?*
- *Run mode-- as app creator or app user*
- *Data access for different classes of users*

Understanding authentication and authorization – AppSheet

AppSheet uses the industry-standard OAuth protocol to secure access to your cloud file system. Let us describe the process when you sign up with AppSheet through your Google account. The process is similar if your account is through Dropbox, Box, etc.

On sign up on our website, you are redirected to a Google page where you are asked if you are willing to authorize AppSheet with a requested set of permissions. Here is what you are agreeing to and why:

- You let AppSheet verify your identity. This is so that we know can associate a unique identity with each user.
- You let AppSheet know your email. This is so that we can send you email as you create apps.
- You let AppSheet read your files and folders. This is so that you can select spreadsheets and images to use in the app.
- You let AppSheet make copies of data into your cloud file system and edit that data. This is so that you can make copies of apps.
- You let AppSheet read and write your spreadsheets. This is what it is all about after all.

We really do not like asking for permissions to your data and have worked to try to keep this minimal while still providing useful functionality.

Once you go through the auth process, Google and AppSheet use internal identifiers (called access tokens) to allow AppSheet software to act on your behalf. To explain this simply:

- Whenever one of your apps is used, AppSheet has to read and write your data from Google. It does this using your access token. Your access token is never used with anyone else's apps so your data is appropriately protected.
- You can completely revoke permissions by going to your Google Drive options and disabling AppSheet.

Related articles

- *Sign in*
- *Data access for different classes of users*
- *How do I design a secure app?*

- *Understanding the secure flow of data*
- *Controlling data inputs with column constraints*

Understanding the secure flow of data – AppSheet

AppSheet uses industry-standard security protocols and practices to secure your data. Users of your mobile app interact with a local copy of the data on the device. This local copy is maintained in a combination of 'HTML5 local storage' and the file system on the device. There are standard security mechanisms in place to ensure that unauthorized users cannot see this data.

When users sync their app, changes they make are sent to the AppSheet web service over an encrypted protocol (HTTPS). AppSheet then applies the changes to the backend spreadsheet (on Google Drive, Dropbox, etc). The latest version of the spreadsheet is read (from Google Drive, Dropbox, etc) and sent back to the mobile app.

Related articles

- *Where is my data cached?*
- *Understanding authentication and authorization*
- *Data access for different classes of users*
- *Run mode-- as app creator or app user*
- *Controlling data inputs with column constraints*

Where is my data cached? – AppSheet

Caching on the mobile device

By default, AppSheet does not cache spreadsheet data on the server. However, for read-only apps, server-side data caching significantly improves the speed perceived by app users during a Sync-- explained in simple terms, the mobile app can get the data directly from AppSheet's servers without having to wait for the data to be fetched from the backend cloud storage platform (Google Drive, Dropbox, etc).

By default, AppSheet does cache spreadsheet data on the mobile client in order to allow continued app usage despite transient loss of network connectivity (for example-- getting into an elevator).

However, full and seamless offline behavior needs to be explicitly enabled in the app definition, and these features require the Premium subscription plan. By default, AppSheet does not cache images and documents that are referenced by the spreadsheet data for offline access. You can explicitly require AppSheet to cache images and documents by setting the appropriate option in your app definition. After you do so, image downloads will happen asynchronously during initial app load, and subsequently the entire app and its image/document content is available offline.

By default, your device needs to be online to launch an AppSheet app from the homescreen icon. Of course, it can then function despite transient loss of connectivity. If you need the app to launch when offline, this is an option you can explicitly set.

These features are set from the Advanced Editor>Settings>Content.

Caching on the AppSheet Servers

Your data is never stored on AppSheet's servers.

The AppSheet web service is an intermediary between the mobile app and the backend spreadsheet. Importantly, it does NOT have a persistent copy of the spreadsheet so there is no danger of your data being compromised via the AppSheet web service. There are two caveats to this statement:

- You can optionally ask AppSheet to cache spreadsheet data in our service in order to improve performance. If so, this data is cached in-memory in our server for up to five minutes at a time.
- AppSheet caches resized copies of images used by the apps. Image resizing is important to conserve network bandwidth to mobile devices.

Related articles

- *Displaying images and documents*
- *Data access for different classes of users*
- *How do I design a secure app?*
- *Content caching on the AppSheet server*
- *Expressions*

Sensitive "Personally Identifiable Information" (PII) data policy – AppSheet

With our **Pay Per User "Pro" plan**, App Creators may designate columns as PII, so that AppSheet does not retain any information about those columns in our system logs.

For example, if you update a record with a PII column, our system logs would not record those columns. They get passed along to the backend sheet and whatever policies that it may have will apply.

To mark a column as **PII**, follow these instructions:

Related articles

- *Is it possible to create an incrementing number for each record?*
- *Other expressions*
- *Conditional formatting*
- *Column types*
- *Lists expressions and Aggregates*

Reliability

What if I lose connectivity? – AppSheet

If you expect your app to be offline for any non-trivial period of time, we strongly recommend configuring the app to indicate that it should work offline. AppSheet has platform features that explicitly handle this scenario.

However, even for apps that have not enabled this feature, AppSheet is designed to work through transient connectivity failures. Mobile apps work in occasionally connected environments, and network failures do occur.

- All changes are made locally and are queued up to send over the network when you choose to sync.
- During Sync, if any changes fail to be propagated, they are retried. Our system is designed so that the changes are 'idempotent'-- this means that even if we mistakenly try to apply the same change repeatedly, the result is still sensible.
- You can keep using the app until you get to a location with reliable connectivity and successfully sync your changes.

Related articles

- *Errors and retry*
- *Offline behavior*
- *AppSheet is NOT...*
- *App formulas*
- *Expressions*

Errors and retry – AppSheet

Mobile apps must account for failure to send updates to the backend. There are multiple reasons for failure:

- Your mobile device may have connectivity issues.
- We hate to say this, but AppSheet's servers could have occasional connectivity or downtime issues.
- Your cloud file system (eg: Google Drive) can have occasional connectivity or downtime issues.

These are the realities of working in a mobile-to-cloud environment where changes from the app have to be recorded at the backend cloud service. Yet any data captured by the app should never be lost.

AppSheet accomplishes this using three basic mechanisms (none of which you need to do anything to configure-- it just happens automatically):

1. All changes are recorded locally on the device. Even if the device shuts down and restarts, the changes are still available. Of course, if the device is lost or destroyed before you sync, those changes are gone forever.
2. When data is synced from the app, it travels to the AppSheet backend, and then on to the backend spreadsheet, and then the acknowledgment of success has to flow back all the way to the mobile app. If this does not happen in a timely fashion, the user sees an error message. Importantly, the change is queued up for a subsequent retry. If the user syncs again, the change gets sent again.
3. Of course, now we could have the situation where the same change is attempted multiple times (because perhaps the success acknowledgment failed to reach the app, though the update was successfully applied). All AppSheet updates are designed to be 'idempotent'-- i.e. you can apply them repeatedly without a change in behavior.

There is one other situation where errors occur-- when the app creator changes the app definition by adding or removing columns, but some user of the app still has the old version of the app and has unsynced updates on that app. Unfortunately, when there is a structure mismatch between the updates and the app definition, the updates fail. Retries will continue to fail. More details on handling this situation are provided *here*.

Related articles

- *Concurrent usage with multiple users*
- *Errors during Sync*
- *What if I lose connectivity?*
- *Errors accessing a spreadsheet*
- *Specifying a key*

Concurrent usage with multiple users – AppSheet

Apps built with AppSheet are meant for multiple concurrent users. This leads to the natural question -- how do we deal with multiple people updating the data?

The simple rule is that 'the last writer wins'.

- The changes of each user are isolated until a sync occurs. At this point, AppSheet attempts to replay the user's changes in order against the backend spreadsheet.
- The unit of change is a single row. Many applications partition well so that no two users update the same row.
- If two users happen to change the same row, the last user's change is what persists. In fact, someone could also update the data directly in the backend spreadsheet. This is treated no differently from any other data update.

There are three cases during sync when this behavior is exposed to users of your app:

- If the user adds a row but a row with the same key already exists, then that row is *updated* with the new data.
- If the user deletes a row but no row with that key exists, then that change is ignored.
- If the user edits a row but no row with that key exists, then that change is ignored because clearly some other user explicitly deleted it first.

Related articles

- *Data access for different classes of users*
- *Transferring app ownership to another user*
- *Specifying a key*
- *How do I design a secure app?*
- *Modifying column structure in the editor*

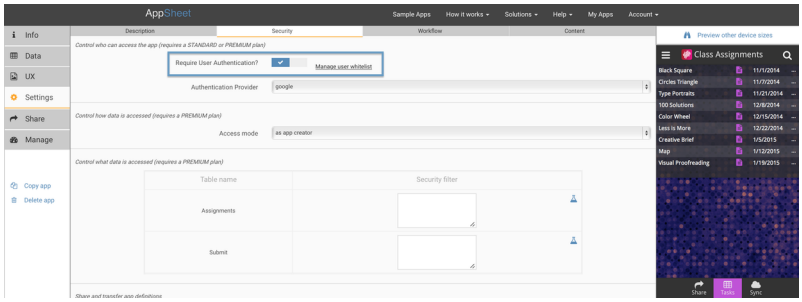
Access Control

Who can discover and install your app? – AppSheet

AppSheet does not have an app store or any other environment for others to discover your apps. This is by design. You decide to tell others about your app by sharing an install link. And your existing users can share the app install link with others as well. Of course, if someone decides to post your app's install link on Twitter, this could be a good thing or a bad thing for you!

If you want further controls on your app, enable the setting that requires users to sign in before using your app. When you do this, you have to explicitly add users to your app's user whitelist. Nobody else will be able to start the app. **You can do this in the Basic Editor>Settings tab, and the Advanced Editor>Settings>Security tab.**

Click the checkbox next to 'Require User Authentication'.

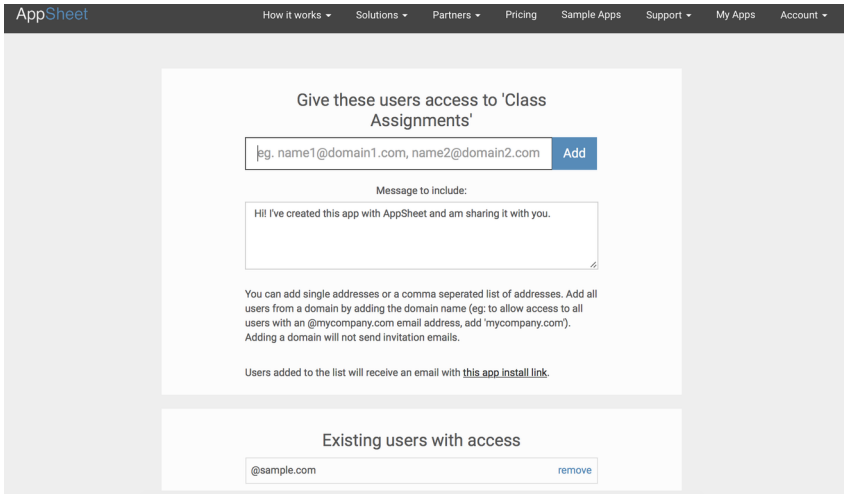


Then click the 'Manage user whitelist' link that appears. You'll be taken to a whitelist administration page where you can either enter individual email addresses or entire domain names that are allowed to access your application.

When you have a relatively small number of users, it is easiest to enter individual email addresses for each user.

When you have lots of users, they are all members of the same domain, and everyone in the domain is permitted to use the application, it is easiest to enter a domain name. Some authentication providers offer the option of creating and registering your own domain name. You can then assign email addresses using that domain name. For example, you might create a domain like "mycompany.com" and assign email address like "BobSmith@mycompany.com" and "MaryJones@ mycompany.com". By entering "mycompany.com" in the whitelist, you allow everyone having an email account with the domain name "mycompany.com" to access your application. You can learn more about creating and registering domain names and assigning email addresses by searching for "Google Apps for Work". Other authentication providers have similiar offerings.

For your safety, we prevent you from entering widely used domain names in the whitelist, such as "google.com".



Related articles

- *Data access for different classes of users*

- *Offline behavior*
- *How do I design a secure app?*
- *Advanced app customizations*
- *Expressions*

Data access for different classes of users – AppSheet

Many AppSheet apps allow users to update data. Please see *this article* that describes how you control update permissions at the granularity of the entire app.

There is sometimes a desire to limit the ability of a certain class of users to see and/or update some data. For example, the creator of an app for use in the workplace may want work assignments editable by managers, but not by employees.

You cannot limit some data in the app to some users and other data to other users. The simple answer in AppSheet is to build different apps for different classes of users.

It is easy to build multiple apps against the same underlying data, but with different permissions for each class of users. You can do this via the update permissions instructions found in *this article*.

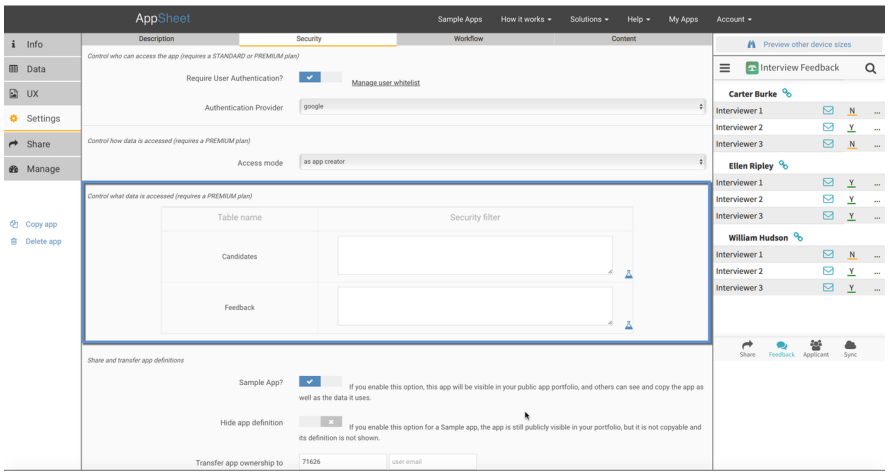
Related articles

- *Controlling data updates*
- *Run mode-- as app creator or app user*
- *Who can discover and install your app?*
- *How do I make sure each user sees only their own data?*
- *Maintaining an audit trail*

How do I make sure each user sees only their own data? – AppSheet

Security filters are yes/no expressions optionally associated with each table in the app. They will typically utilize some properties of the app user to limit the data shown to the app user.

See Advanced Editor>Settings>Security



Examples:

To limit access by User Email:

- [EmailColumn] = USEREMAIL()

To limit access by User Email Domain:

- CONTAINS(USEREMAIL(), [EmailDomainColumn])

It is possible to build more complex logic as well --- AND/OR/NOT/functions/derefs, etc.

Note: Both the USEREMAIL() and the USERNAME() functions require that the app has enabled user authentication (i.e. app users are required to sign in to use the app). It is preferable to use USEREMAIL() instead of USERNAME() because it is always guaranteed to be available for a signed-in user.

Related articles

- *Expressions*
- *How do I control the order of columns displayed in the app?*
- *Data access for different classes of users*
- *Run mode-- as app creator or app user*
- *How do I design a secure app?*

Run mode-- as app creator or app user – AppSheet

The mobile app that you build communicates with the AppSheet backend, and it is the AppSheet backend that communicates with your cloud data provider (eg: Google Drive) to fetch or update data.

For security reasons, the cloud data provider will not let AppSheet access your data unless AppSheet has the permissions to do so. Whenever a user or app creator signs in, he/she grants AppSheet the permission to act on their behalf.

As app creator

In the default run mode ('as app creator'), the AppSheet backend accesses data with the identity of the app creator. So if you are the app creator, and you have access to the spreadsheet that the app is based on, you can distribute your app to other people who do not have direct access to that data but the app still works fine.

There is a security setting in the Advanced Editor which allows you to change the run mode to 'as app user'. As the name suggests, in this mode, the AppSheet backend will try to access the app data with the identity of the app user. This will fail for apps that don't enforce sign in. And even if the user does sign into the app, it may still fail if the user doesn't have permissions to access the source spreadsheet and folder.

As app user

So what's the benefit of the 'as app user' run mode? It is a stricter security mechanism (ensures that the permissions model of the cloud data provider are strictly maintained). It is also an auditing mechanism, especially for cloud providers like Google Drive and Box that maintain audit histories of all updates made to documents. If the app runs 'as app user', each change to the app recorded in the audit history will carry the identity of the specific user who made the change. Please be aware that if the app user does not have permissions to access the underlying data (eg: the spreadsheet on Google Drive that holds the app's data), the app will fail for that user.

To avoid any confusion, this option has no bearing on the mode in which the app itself runs on the mobile device. The user who signs in is always the current user and expressions like @(_USERNAME) and @(_USEREMAIL) always reflect the current signed-in user, never the app creator.

Related articles

- *Maintaining an audit trail*
- *Data access for different classes of users*
- *How do I design a secure app?*
- *Controlling data updates*
- *Expressions*

Maintaining an audit trail – AppSheet

In some apps, it is important to know who made each update. A pre-requisite for this is, of course, that users must be required to sign in to use the app.

The simplest mechanism is for the app to maintain an extra column of type Email to hold the user's email. If you provide @(_USEREMAIL) as the Default value of this column, then every new row entered will automatically populate with the email address of the currently logged in user.

Some apps require a richer audit-trail, to maintain the history of changes made by different users. Typically, the backend cloud storage provider has an auditing mechanism. For example, Google Spreadsheets provide a Change History that shows a detailed list of changes made, when they were made, who made them, etc. In order to utilize this backend audit history, it is important that the AppSheet app run with the security setting '*As App User*'. Each potential app user should also be given permissions to access the backend spreadsheet.

Related articles

- *How do I design a secure app?*
- *Expressions*
- *Column types*
- *Plan upgrade required*
- *Printing your data*

Tutorial Content

How do I design a secure app? – AppSheet

Apps created with AppSheet are secure because of the architecture of AppSheet, because of the underlying security infrastructure of cloud and mobile technology, and because of security options in your control.

First, let's quickly summarize the AppSheet architecture from a security point of view:

- **Cloud data:** Your data is stored in your cloud data provider (Google Drive, Dropbox, Box, Smartsheet, Office365, etc). AppSheet does not store your data.
- **Authorization:** *As explained in another section*, when creating an application, you sign into AppSheet with your cloud storage account and give AppSheet the permission to read and write your cloud storage data on your behalf. The industry standard OAuth protocol is used in this process.
- **Apps get built based on the cloud data:** The application you create is saved to an application definition that is maintained by AppSheet in a secure cloud database. Communication between the AppSheet cloud service and your cloud storage provider uses secure web protocols (HTTPS).
- **The apps run on a mobile device:** Communication between the mobile device and the AppSheet cloud service uses secure web protocols (HTTPS).
- **Local data storage:** The application definition and all the necessary data is copied to the mobile device and securely stored locally on the device using HTML5 local storage.
- **Data synchronization:** When you add, update, or delete data via the AppSheet application running on your mobile device, all changes you make are saved to your cloud data provider. The AppSheet cloud service acts as a go-between to connect the mobile app with the cloud data provider used by your app.

You may have the following security questions about each of these aspects of AppSheet-- we answer each of them in this article:

- **Who can access my cloud data directly?**
- **Who can access my app and how do I control that?**
- **Do the users of my app also need to have access to my spreadsheet in cloud storage?**
- **What can users do with my app? Can they delete my data?**
- **Can I limit the data that specific users of the app can see?**
- **Can I limit the actions or capabilities that specific users of the app can do?**
- **Who can access the data on the mobile device?**
- **What happens if an app user leaves my organization and I want them to stop using my app?**

I will explain these topics using a fictional app created by a teacher to distribute and collect assignments from her students in a graphic design class.

Who can access my cloud data directly?

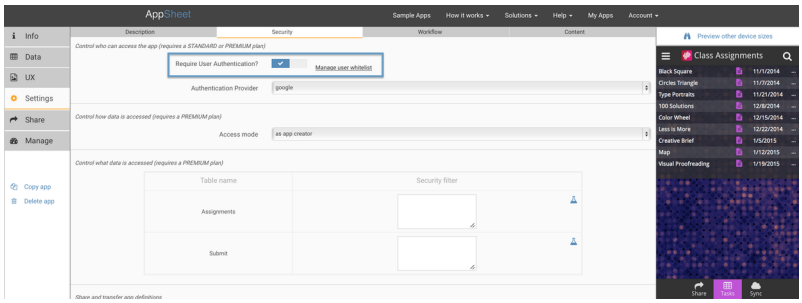
No other AppSheet user can access your cloud data directly-- that permission lives solely with you (and whoever else has your cloud account's login info).

Depending on the update permissions you have enabled in the app, the users of your app may be able to add, edit, or delete values that live in your cloud-hosted spreadsheets and that are exposed via the apps you build with AppSheet. You can control these permissions as described later in this article.

The AppSheet cloud service has permissions to act on your behalf to read and write your cloud data. These permissions are only used for the specific purposes of app creation and app execution. AppSheet does not save copies of your data. Furthermore, because of the way the OAuth protocol works, you can at any time decide to disable the access permissions granted to AppSheet.

Who can access my app and how do I control that?

You probably want to control who has access to your app-- I'll use the *Class Assignments* app to show how to do this. I want to ensure that only authorized students from my school can download and use the app. I can do this from the Advanced Editor>Settings>Security pane, by clicking the Require User Authentication checkbox:

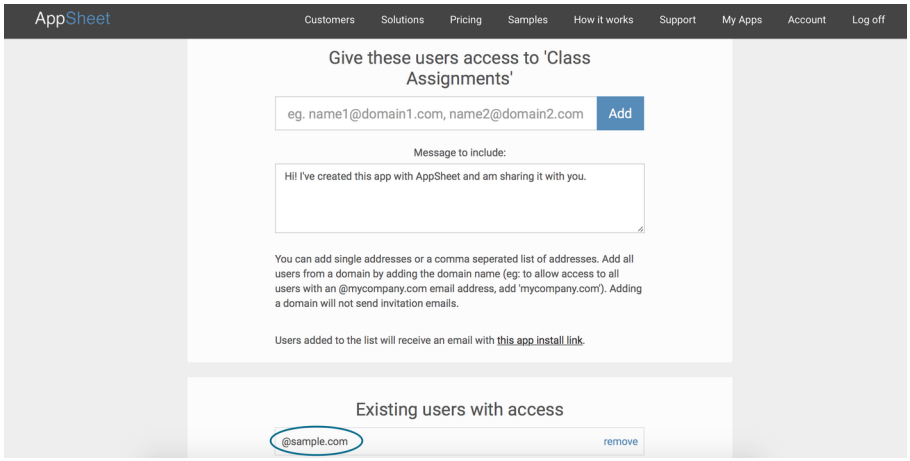


The whitelist is a list of users or a full domain you have permitted to access the app. You'll see that I've added a full domain (in this case, all students with a @sample.com subdomain), as well as a message that is included in the install link email that is sent when I click add.

AppSheet does not maintain its own user registry with usernames and passwords. Instead, your users will sign in using a standard cloud provider such as Google, Dropbox, Office365, Box, or Smartsheet. By default, we assume that your users will sign in using the same cloud provider that you use for your cloud storage. For example, if you use Google Drive, your app will be configured to ask your users to sign in with their Google accounts. AppSheet matches the email address of the application user's Google account with the list of allowed email addresses you specify in the white list. The user is permitted to use your app if the user's email address appears in the white list.

If you wish, your can require users to sign in using a different cloud provider that you use for your cloud storage. For example, you might require your users to sign in via Dropbox even though you use Google as your cloud storage provider. You can specify the sign in provider using the Advanced Editor. Open the Advanced Editor, select the "Settings" tab, select the "Security" tab, and select the appropriate authentication provider from the "Authentication Provider" drop down list.

If you wish, you can allow users to sign in using any of the cloud providers. For example, you may wish to allow users to sign in using either Google, Office365, or Smartsheet. To do this, open the Advanced Editor, select the "Settings" tab, select the "Security" tab, and select the blank authentication provider appearing at the top of "Authentication Provider" drop down list. This will enable your users to sign in with any of the standard cloud provider including Google, Dropbox, Office365, Box, and Smartsheet.



In the rest of this article, I'll assume that you are utilizing the whitelist to limit access to specific app users.

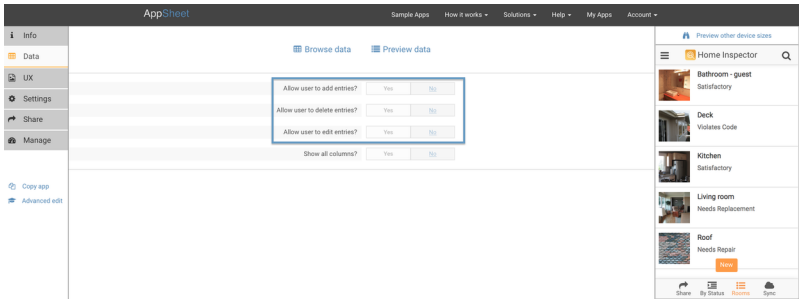
Do the users of my app also need to have access to my spreadsheet in cloud storage?

No. If you are the app creator, and you have access to the spreadsheet that the app is based on, you can distribute your app to other people who do not have direct access to that data. This works because the default execution mode ('as app creator'), instructs AppSheet to access data using the identity of the app creator. (Note: in certain advanced scenarios, you can specify the app to run as the app user. *Find out more in this section*).

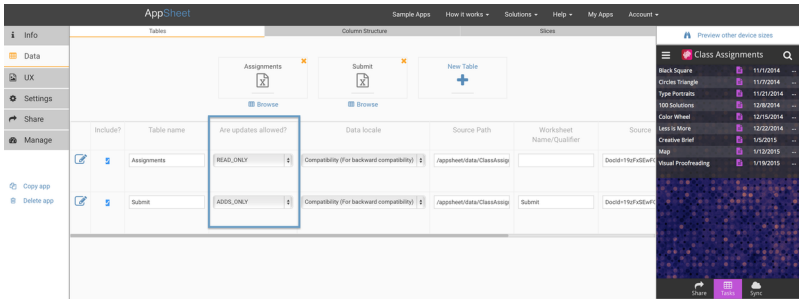
What can users do with my app? Can they delete my data?

As the app creator, you have full control over what kinds of changes users can make to the data in your app, or whether they can make changes at all. You may want a unique combination of permissions: for example, you might want users to be able to add and update data, but not delete data. Or, you may want users to have the ability to add new data, but not update or delete data. Any combination is allowed. You can do this in both the Basic and Advanced Editors.

Basic Editor



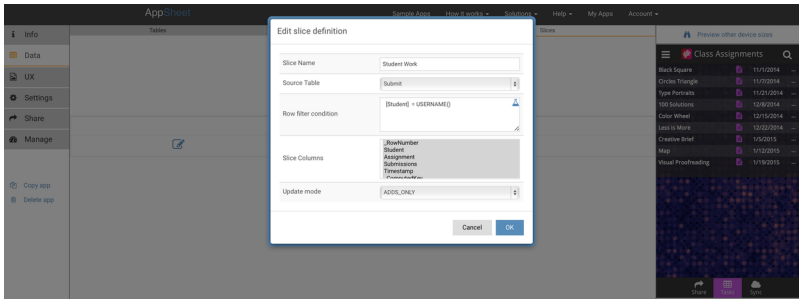
Advanced Editor



Can I limit the data that specific users of the app can see?

Users of your app can only see data that you choose to expose through the app. In fact, you might want certain classes of users to only have access to view specific subsets of data, and other users to different data. *You can do this using slices.*

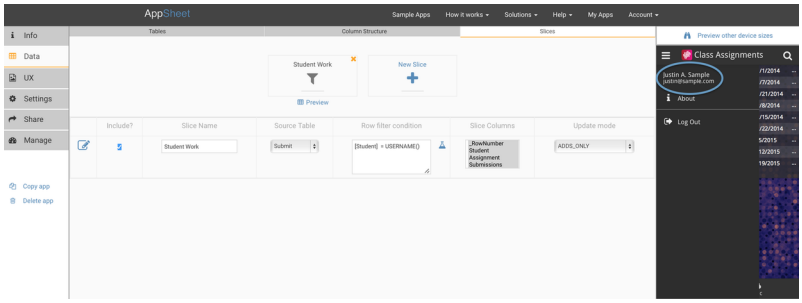
In the case of the Class Assignments app, I want to make sure that students can submit completed work and continue to view past submissions, but I don't want them to be able to see the completed work of other students. The best way to do this is to create a slice that shows only the assignments of the logged-in user. I'll do this from the Advanced Editor>Data>Slices tab, then clicking +Slice.



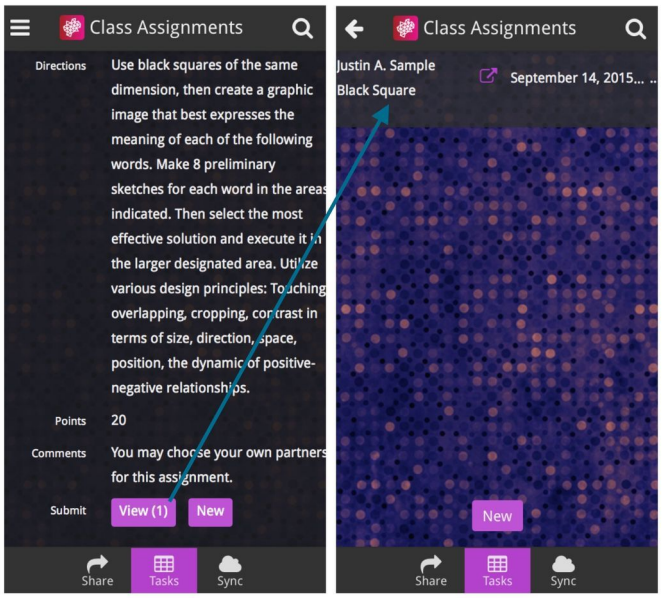
First I'll need to name my slice and then specify a few parameters. In this case I'll want to choose the Submit table so new submissions will end up there. I'll also want to make sure students can only add submissions, and not delete or edit-- so I'll choose ADDS_ONLY as the update mode for the slice.

In order to make sure logged-in students can only see their own present and past submissions, I need to filter based on the Student column. Since I required sign-in for the app, my filter condition should be Matches user name-- the app will then only show data that matches the current logged-in user's name.

You'll see then, in the hamburger menu, that the current logged-in user's name shows up.



You'll see below how the slice ensured Justin A. Sample would only be able to view his own past assignments.



Can I limit the actions or capabilities that specific users of the app can do?

All users *of the same app* will have the same editing, updating, or deleting capabilities. However, if you'd like different classes of users to have different permissions, the best approach is to create different apps for each user with the corresponding permissions levels. Each app would be tied to the same source spreadsheet. Creating a new app takes just a minute, especially if you start by copying an existing working app.

Who can access the data on the mobile device?

On the mobile device, AppSheet uses industry-standard "local storage" technology associated with web browsers (web browser technology is embedded inside the AppSheet mobile app and the actual implementation largely utilizes HTML5 technology). Since you require users to sign in to access your app, each user's data is isolated from any other user's data. Even were multiple users to use your app from the same device, their data would be isolated as long as they sign into the device as different users. In short, if you follow standard security practices on the device (users sign in with passwords), your app will be as secure as all other apps based on web technologies.

What happens if an app user leaves my organization and I want them to stop using my app?

At any time, you can return to the user whitelist and remove any users you've previously added. Any subsequent actions by the removed user from their mobile app will fail when the actions engage with the AppSheet cloud service (typically when synchronizing or saving data).

As an extreme measure, you could also make a copy of the app and delete the old one. The old one will stop working and then you can control access to the new one. You'll need to make sure to redistribute the new app link to the other users who still need access.

You can find out more about security in our Security and Reliability section.

Related articles

- *Defining and using slices*
- *How should I use multiple sheets in my app?*
- *Run mode-- as app creator or app user*
- *Change alerts and workflows*
- *Expressions*

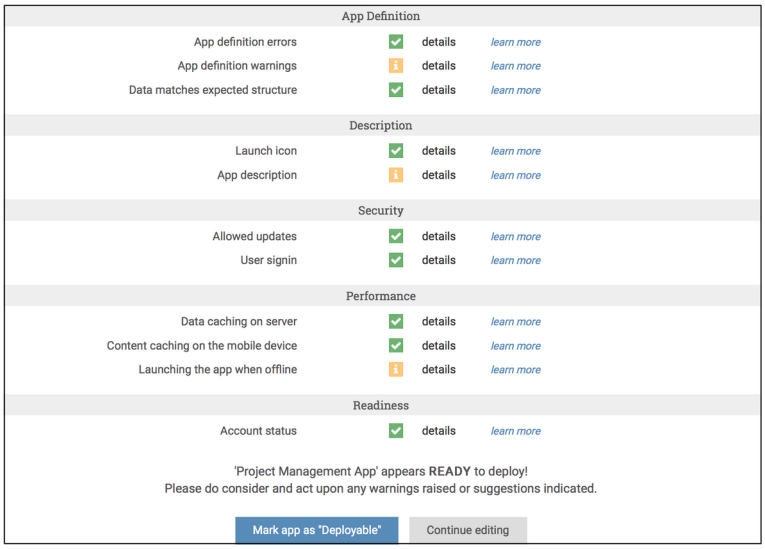
Managing Apps

Preparing for Deployment

Check your app for deployment – AppSheet

There are a few simple checks to make sure your app is ready for use and deployment.

On the app editor page, click the **Manage** button and run a Deployment check. AppSheet verifies that your app is ready to deploy-- if not, it informs you about steps you need to take.



- One class of checks is associated with the app definition and the underlying data. Any errors or warnings will be highlighted.
- An app must be assigned a launch icon and description before deployment.
- Because app security is really important, the deployment check includes a security review.
- Finally, AppSheet ensures that the features used by the app are compatible with your account type.

Related articles

- *Reflecting your brand*
- *Testing apps*
- *Plan upgrade required*
- *Offline behavior*
- *Multi-page forms with conditional branching*

Testing apps – AppSheet

During the app creation process, you may want to familiarize yourself with AppSheet's app building platform and all of the features it offers. Our goal is for you to be confident in the apps you create, the features they use, and their intended functionality before you decide which pricing plan fits your needs.

For these reasons, testing and prototyping apps are always free on AppSheet. You will not be restricted from using certain features that may otherwise be covered under paid plans if you are simply testing your apps and making sure they are ready to launch and deploy. Once you are confident your app runs as intended, and you know how many people will likely be using it, you can then decide which pricing plan is right for you.

How do I make sure my app is ready to deploy?

When you feel you're close to a working app, you will want to run a deploy check. You can find out more about how to do this *here*.

Related articles

- *Check your app for deployment*
- *Reflecting your brand*
- *Information for an IT department*
- *"Un-deployin"g your app*
- *Pricing, billing, and subscription plans*

Information for an IT department – AppSheet

If you are an AppSheet innovator at a larger company with an IT department, you may need to provide some information about AppSheet as you progress towards deploying your apps to others within the company. In this document, we've put together an "FAQ for IT". Many of the topics have more detailed information via separate articles in our Zendesk documentation site.

Architecture questions How are AppSheet apps created?

Apps are created on the AppSheet website. Each app is based on one or more 'tables'. Typically each table is a spreadsheet, but in general, each table is linked to a source of *structured data*. The rest of the app is defined via configuration options. There is no code involved, so apps can be created in minutes. **How are AppSheet apps distributed?**

Apps are distributed by sending an *install email to desired users with an install link*. Users open the install email on a mobile device, click on the install link, and can install the apps instantly. **What is the high-level architecture of AppSheet?**

Each mobile app is derived from and based on backend data, typically a spreadsheet hosted on a cloud storage system like Google Drive, Dropbox, or Office365. On the mobile device, a copy of the data is cached locally, in order to permit offline operation. Whenever changes are made in the app, or when the users explicitly choose to sync the app, the *changes from the app are moved to the backend spreadsheet, and vice versa*. All communication between the app and the backend data happens through the cloud-hosted AppSheet web service. **Where is AppSheet hosted?**

AppSheet's web service is hosted on Microsoft's Azure web hosting infrastructure. **Can AppSheet be hosted on-premise within the company?**

No, AppSheet is hosted in the cloud. It is not an on-premise service. **Is AppSheet reliable?**

The AppSheet web service is highly reliable with 99.99% monthly uptime. The reliability derives from both the redundant cloud architecture as well as the reliability of the underlying cloud infrastructure services like Microsoft Azure. The greatest source of reliability concerns for a mobile app stem from *connectivity issues*. AppSheet is explicitly designed for environments with bad or intermittent connectivity, and the data synchronization technology handles various kinds of failures seamlessly without data loss. **Is each app separately deployed via the iOS or Android app store?**

No. Mobile apps run on the device within a common 'host' app called AppSheet. This host app is already available in the iOS and Android app stores. This allows any new app to be *instantly deployable*.

Data architecture questions What sources of data can AppSheet access?

AppSheet apps can access spreadsheet data from Google Drive, Office365, Dropbox, Box, and Smartsheet. More data sources are being added actively. Connectors to Salesforce and SQL databases will shortly be available for early adopters. **Do we *have* to use data in a spreadsheet?**

AppSheet is meant for non-developers to easily build apps, which is why we have emphasized access to spreadsheets. Over the next few months, we will also be supporting more traditional enterprise data sources as well. **Can you connect to our corporate database or resources behind a firewall?**

Not directly. The AppSheet architecture permits our service to interact with data sources via a web service that acts as a proxy to the actual data. If the actual data is behind a firewall, you (or your IT department) will need to create a web service that acts as a proxy for the data. This is a relatively straightforward task for a web developer and we can help enterprise customers with sample code for this proxy service.

Mobile platform questions What mobile platforms do you support?

AppSheet apps run on iOS and Android devices. We encourage iOS version 8.0 and higher, and Android version 4.3 and higher. Lower versions of these operating systems can cause subtle bugs or performance problems in certain scenarios. **Can the apps run in a browser?**

Yes. While we primarily focus on mobile devices, many AppSheet customers also run their apps in a web browser, usually within an iframe. In fact, on unsupported mobile platforms like Windows Phone, we recommend running within a browser. Most of the app functionality is supported in a browser (except offline image caching and bar code scanning). We recommend a modern browser and the apps work best on Chrome. AppSheet does not work on IE versions lower than V10.0.

Functionality questions Do you support feature X?

We add features to AppSheet at a rapid rate-- it is not uncommon for us to unblock a customer by adding a new feature in a couple of weeks. There is an active *user community* that drives and prioritizes our feature investments. All features are initially announced via the community for trial and then promptly added to the Zendesk documentation.

Security questions Where can I learn about the security model?

We have written a short article describing different aspects of *the AppSheet security model*. We are happy to share further detail with IT experts. **What data does AppSheet store on its servers?**

The AppSheet service records user signup information and access tokens in order to be able to access backend spreadsheet data. The definition of each app is also stored. *The actual data used by each app is not copied to the AppSheet service*-- instead it passes through the AppSheet service as it flows between the mobile device and the backend cloud storage system chosen by the customer.

AppSheet does record usage events and also logs data updates in order to support diagnostics, debugging, analytics, and billing. We are introducing an option for customers to limit the duration for which this information stays in the system logs. **Has AppSheet passed security and/or privacy audits?**

As a startup company, we have focused for the last year on building a secure and scalable architecture. We have designed for security and for data privacy. Over the next year, we will be working with established auditors to gain official compliance certifications.

Support and services questions What is the response time for product support questions?

Our user community forum is the venue for most support questions. Not only is every member of our team actively engaged with user questions, we also have power users who are community moderators who also help all members of the AppSheet community with questions and issues. Most issues are responded to within hours and resolved within the day. We also offer premium support for customers in higher subscription plans. **Do you provide professional services for app development?**

We try to focus on core platform development. For some key strategic engagements, we can help customers with app development. The preferred model is for our customers to work with one of our solution partners-- these are professionals who are experts on the AppSheet platform. Depending on your needs, we can identify suitable solution partners.

Related articles

- *Pricing, billing, and subscription plans*
- *Launching AppSheet apps from other AppSheet apps*
- *Check your app for deployment*
- *Application crashes / errors*
- *Automated usage summary emails*

Pricing, billing, and subscription plans – AppSheet

AppSheet is a subscription service.

We have structured our plans and pricing to encourage adoption with minimal friction and barriers to use. If none of our plans are appropriate for your scenario, we encourage you to contact us.

By default, every user account is free until a specific subscription plan is chosen. Please review the list of available plans at our *Pricing page*.

You can modify your plan via the Billing tab on your account page.

Appropriate usage of each type of plan

AppSheet is free only for prototyping and for personal projects. While you are evaluating AppSheet, you are not required to pay as you evaluate all of the platform features. In fact, we encourage you to try all the features of the platform. However, as you deploy your apps to others and start to use them in your organization, you should change your account to a paid plan. When you take your app through the 'Deploy Check' process, you will be warned if your account is not on a paid plan and that is the preferred stage at which to sign up for a paid plan.

Per-user Plans

When you build apps for users within your organization, you probably need to identify specific people who will use the app (perhaps members of your team or department). This is an ideal case for a per-user subscription plan. In a per-user plan, you purchase licenses for every user of your apps. Each licensed user can use many of your apps. So for example, if there are ten people in your team, you should purchase ten licenses and you can create one app, two apps, or any number of apps that are used by these users.

Per-app plans

If you build apps to send to a large base of occasional users (for example, for a community event, or to share your product catalog with your customers), a per-app subscription plan is probably appropriate. Per-app plans differ from per-user plans in that you cannot limit the app to specific users. They are meant for situations where data security is not a concern. In a per-app plan, you purchase licenses for each deployable app in your account. So for example, if you had two apps that you want to deploy widely, you would purchase two app licenses.

Multiple plans within the same account

Each AppSheet account has a single subscription plan. It is not possible to have some apps in an account associated with one plan and other apps associated with a different plan. We recognize though that this is a requirement for some of our customers and will release updates to accommodate such need. For now, you can create additional AppSheet accounts to add apps to a different type of plan.

Changing plans and licenses

You can change your subscription plan or the number of licenses on your plan at any time. Your charges are immediately applied and any charges are pro-rated to reflect the changes.

For example, assume you have five users of your app and have purchased five user licenses with a per-user plan. If you need to send the app to more users, you should simply increase the number of licenses via the Billing tab on your account page. Likewise, you can decrease the number of licenses to reflect lower expected usage or change plans.

It is uncommon to switch from a paid subscription plan to no plan unless you expect your apps to see absolutely no usage at all. It is usually more appropriate to modify the number of licenses to reflect periods of lower expected usage.

Enforcement of plans and licenses

We try to ensure that app usage is never compromised because of short-term discrepancies in plans and licenses. If the actual number of users for your app crosses your license limits, we will still serve the app to those users. Likewise, if you are on the wrong plan or there is a problem with your credit card, we will continue to serve the app to your users. In both cases, we do this for a reasonable interval so that you have the time to modify your plan appropriately.

Billing

You may choose and adjust your Subscription plan until you are confident that you have selected the correct plan for you or your business. Next, enter your Payment Method. Your credit card is charged on a monthly schedule based on the particular plan and license count you have chosen. AppSheet processes credit card payments through Stripe, a popular and reliable platform for credit card payments.

If you make changes in the middle of the month, the pricing is pro-rated to the time of the change. You may see adjustments to the next month's billing to reflect these changes. For example, if you lower the number of licenses, you will see credit reflected in your next month's bill. Please contact us at any time if you have questions about your billing.

As a paid subscriber, you get a monthly email with a summary of the last payment directly from Stripe, please use this receipt as your bill. Unfortunately we cannot produce individual invoices other than the emails sent to you directly from Stripe. Large Accounts may contact us directly for specific invoicing procedures.

Credit card failures

When there is a failure processing a credit card payment, you will receive an email notification indicating that the credit card information should be updated. You can go to your account Billing page and update your card information. In the interim, your app status will remain active and the access to your app won't be affected for up to 15 days from the date the email notification was sent.

Our credit-card charges are USD-based, and cards based in countries outside the US may have security constraints that disable foreign charges. Unfortunately, we are unable at the moment to charge directly in currencies other than USD.

Receipts and invoices

Every time your credit card is charged via Stripe, you will receive an invoice email from Stripe showing what you have been charged by AppSheet and the details of the invoice. If you have not received these invoice emails, please check your email folders in case the mail has been routed to a Promotions or Junk mail folder. If you still cannot find the invoice, we can manually retrieve the invoice for you.

Related articles

- *Plan upgrade required*
- *Deploy your app*
- *Launching AppSheet apps from other AppSheet apps*
- *Including PDF's in your application*
- *Information for an IT department*

Deploying and Sharing Your Apps

Launching AppSheet apps from other AppSheet apps – AppSheet

If you have ever used AppSheet's App Gallery in your mobile device, you'll realize the value of having a list of apps that you an easily access from a single App.

You can create an AppSheet app that can link to other AppSheet apps, helping you create suites of solutions that fits the needs of your users, this also simplifies the deployment process for multiple apps as you can now send one app and simply change the back-end spreadsheet to update to a new version of the app or add a new service.

Introducing app launchers (beta)

An app Launcher has a very similar behavior as the App Gallery in the AppSheet mobile app. To achieve this you need to follow a few instructions:

1. Structure the table

   The App Launcher is based on a table that looks like this:

App Name	Image	App Link
Consultant	Sample app Launcher_Images/ ConsultantImage121017.png	Consultant-10305
Reporter	Sample app Launcher_Images/ ReporterImage.121109.png	Journalist-10305
Assignment Manager	Sample app Launcher_Images/ Assignment Manager.Image.120718.png	SchoolWorkManager-71626
Class Assignments	Sample app Launcher_Images/Class Assignments.Image.120811.png	ClassAssignments-71626
Extracurricular Schedule	Sample app Launcher_Images/ Extracurricular Schedule.Image.121042.png	ExtracurricularSchedule-71626

   - App Name: You can give any name to the app
   - Image: It has to be a public url for the image or enter a reference to a file in your cloud provider. *How to add Images*
   - App Link: Follow the instructions below
2. In the app link field, enter the app name. Each app has a unique App Name, you can find it in the link to the app. For example: *https://www.appsheet.com/template/showdef?appName=ClassAssignments-71626* In this case, the App Name is: ClassAssignments-71626
3. 3. Create the app Add it via one of our add-ons or directly from My Apps > Make New App.
4. 4. Make sure the field types are as follows: Change the field type by going to Advanced Editor > Data > Column Structure App Name: Name Image: Image or Thumbnail App Link: App. The last field to change is App.
5. 5. Make the available view a gallery That's it, the linked apps should load when you click/tap on the image. This is pre-beta so we might still have lots of behaviors to hone.

Related articles

- *Expressions*
- *Column types*
- *Information for an IT department*
- *Managing App Versions*
- *Pricing, billing, and subscription plans*

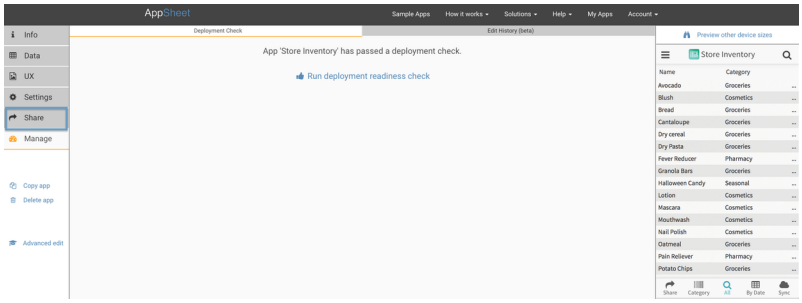
Sharing and distributing your app – AppSheet

You distribute apps by sharing an installation link. Users who have not yet downloaded AppSheet's mobile app from the app store *will be prompted to do so before they can install apps created with AppSheet onto their devices*.

There are a few ways to distribute apps.

Sharing apps via the Editor

When you're ready to launch your app, click the 'Share' button at the left of either Editor.



You'll be taken to a page where you can add email addresses with whom you would like to share your app. An email with the install link will then be sent to those email addresses.

Other ways to distribute apps

Upon creation of an app, you (the creator) immediately receive an email with the installation link. You can also distribute your apps via this email, in a variety of ways:

- You could forward that email to other users.
- You could copy the link onto a website to share.
- You could post it to Twitter and ask your followers to install the app.
- You could send the link via an SMS message.

You get the idea!

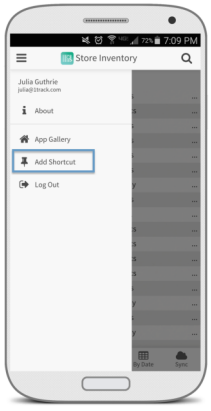
At anytime, you can always go back to the Editor and click 'Share/Install'.

Homescreen shortcut installation on Android

On Android devices, you or your users can install app shortcuts from AppSheet's app gallery without needing to use the installation link. If you have downloaded the AppSheet app from the app store, and then open the *app gallery*, click into the app you want on your homescreen. You will be able to install apps from there to your homescreen in just one step.

Simply click the hamburger menu at the top left and then click "Add Shortcut". The current app's icon should then appear on the homescreen.

If you'd like your users to use this method, just make sure you've *shared the app* with them. They will see your app in the "Shared" tab and can follow the above instructions.



Related articles

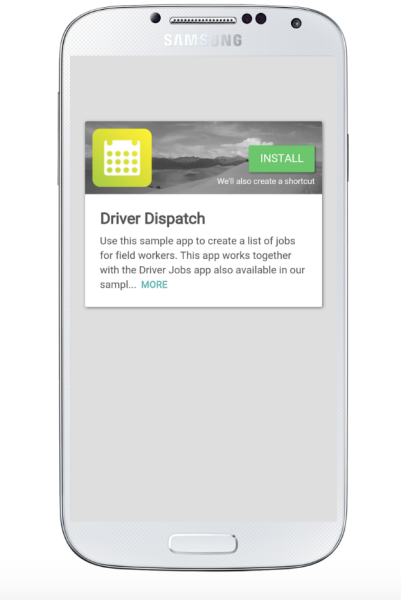
- *Deployment from an install link*
- *Deploy your app*
- *Launching apps shortcut from the homescreen*

- *Making your app definition public to troubleshoot issues*
- *Pricing, billing, and subscription plans*

Deployment from an install link – AppSheet

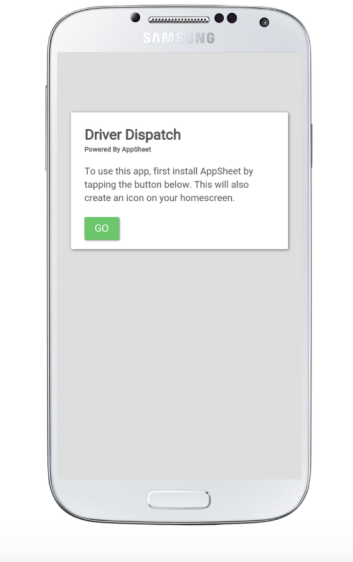
How does a user install your app with the install link? The link points to a page with instructions depending on the combination of platform and web browser that they are using.

Chrome and Firefox on Android: Chrome has features which allow us to detect whether or not AppSheet is installed on a user's device, which greatly simplifies the installation process. First, they are taken to a page that looks like this:



If AppSheet is already installed, tapping the Install button will open AppSheet to the app that you shared (in this case Driver Dispatch). Additionally, it will create a shortcut on the user's homescreen which also points to your app.

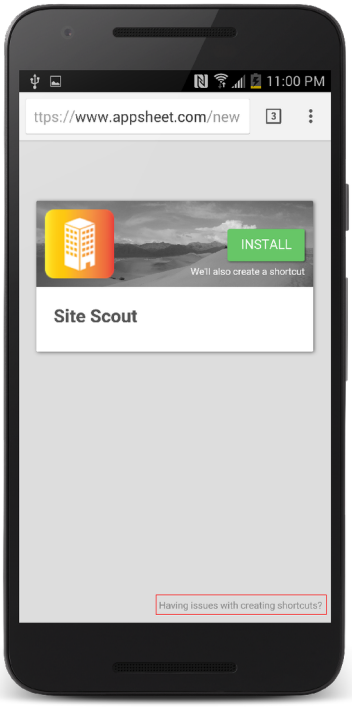
However, if AppSheet is *not* installed, the user will be taken to the following page:



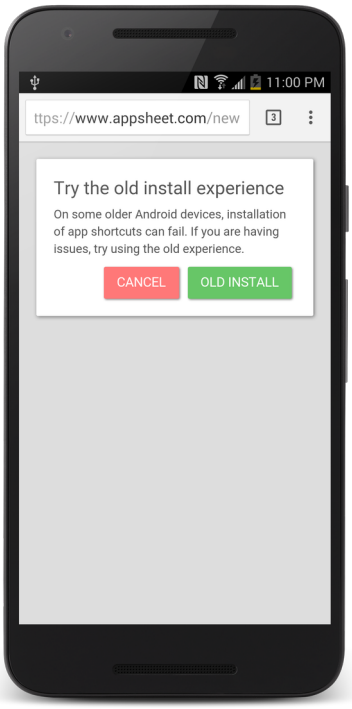
Tapping Go will redirect to the Play Store, where the user can install AppSheet. Once they open it for the first time, they will be taken to the correct app (in this case, Driver Dispatch), and a shortcut will be placed on their homescreen.

Chrome Fallback to Old Install Experience

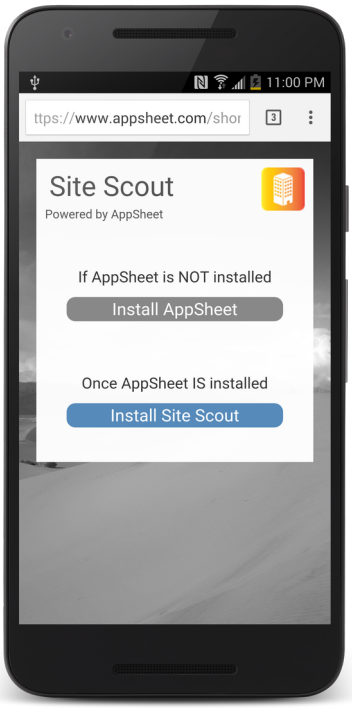
On some older phones, AppSheet cannot automatically create shortcuts for you, so you have the option to create it using a different approach. On the install page, in the bottom right, tap on the gray text that says "Having issues with creating shortcuts?" (outlined in red below)



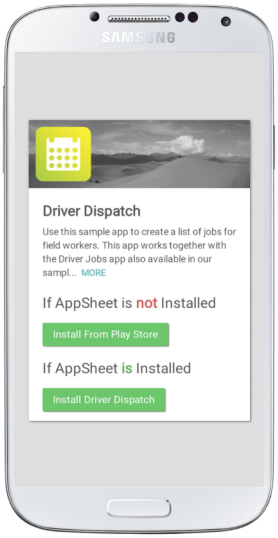
Then, tap on the green "Old Install" button.



After that, you should see a page that looks like the one below. Follow the old instructions to create a shortcut (they can be found here: https://appsheethelp.zendesk.com../appsheethelp.zendesk.com/hc/en-us/articles/205803927-Deployment-from-an-install-link-old-)



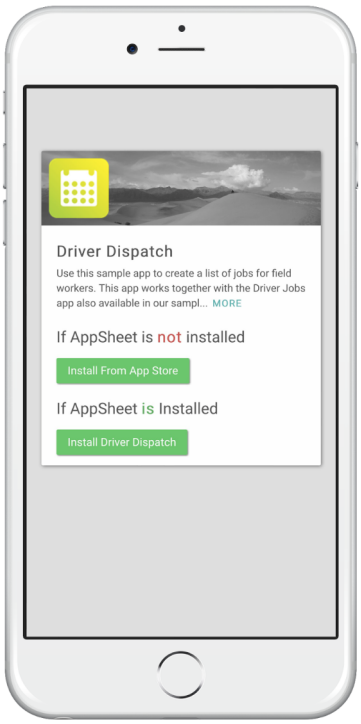
Any other browser on Android: Browsers other than Chrome are less sophisticated, so this process is a bit more complicated. Once the user you shared your app with opens the link, they will be taken to a page that looks like this:



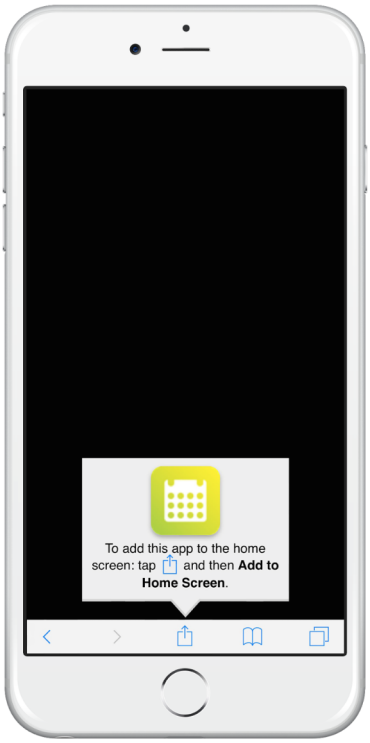
The user is presented with two options: one for when AppSheet is installed, and one for when it is not installed. If the user does not have AppSheet, they should install AppSheet from the Play Store. When they open it, they will be taken to the app that you shared (in this case Driver Dispatch), and a shortcut to the app will be placed on their homescreen.

However, if AppSheet is already installed, tapping the second button will launch AppSheet with the correct app and create a shortcut on the homescreen.

Safari on iOS On iOS, Safari is the only supported browser at the moment. When the user opens the share link sent to them, they will see the following page:

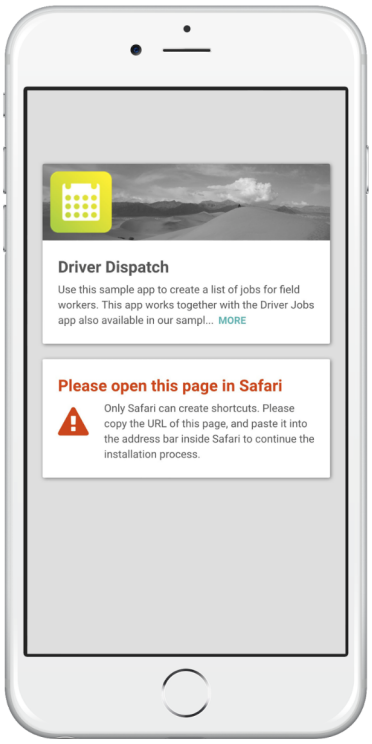


Tapping the first button will install AppSheet. Then they have to come back and tap the Install Driver Dispatch button, which will then instruct them to create a shortcut on their homescreen. It will look like this:



Any other browser on iOS

Only Safari is currently supported on iOS. The following page is the one that opens up when you open the page in a browser other than iOS:



Related articles

- *Deploy your app*
- *Advanced app customizations*
- *Controlling data inputs with column constraints*
- *Expressions*
- *Making your app definition public to troubleshoot issues*

Launching apps shortcut from the homescreen – AppSheet

What happens when the user clicks on an app shortcut to launch the app?

Remember that your app is actually hosted on the *AppSheet mobile runtime*. So when the user launches your app, it first loads AppSheet (which you do not see) and then starts your app. The whole process only takes a few seconds, after which the user can start using your app. Your app looks and behaves exactly as it did in the emulator on our website.

You can find out how to install your app to your homescreen *here*.

Related articles

- *Deployment from an install link*
- *Sharing and distributing your app*
- *Multi-page forms with conditional branching*
- *Choosing and adding data views*
- *Run mode-- as app creator or app user*

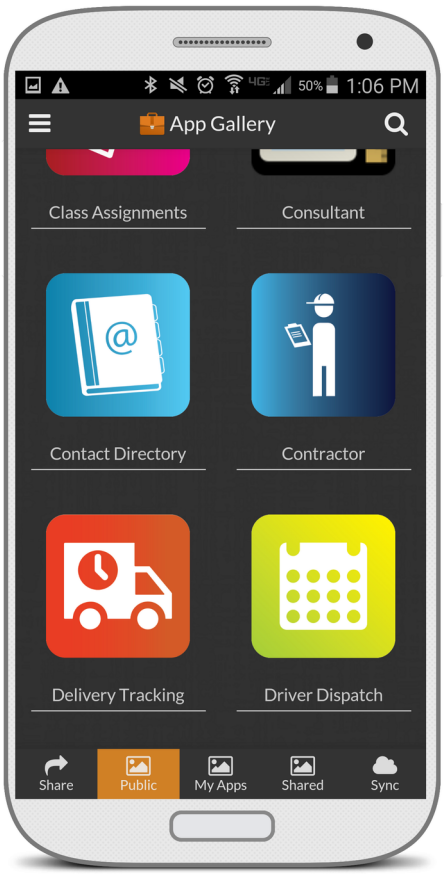
Launching apps from AppSheet's mobile app gallery – AppSheet

We encourage you to install your apps' icons to your homescreen when you're ready to deploy them. However, there are a handful of reasons you may want to use apps without having to first install them to your phone.

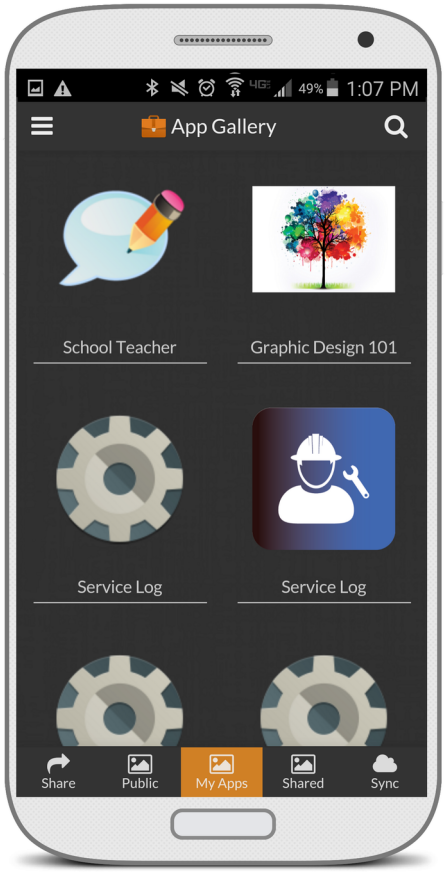
All of the apps you create with AppSheet are available via AppSheet's mobile app that you *downloaded from the app store*. Likewise, AppSheet's app also features a gallery of public sample apps and the apps that others have shared with you. You may want to use these apps without having to install them. Here's how to do it:

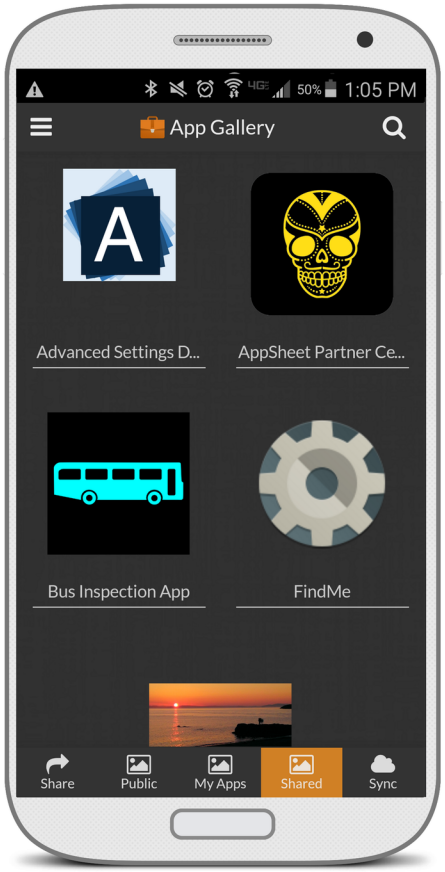
1. Launch the AppSheet mobile app on your mobile device. It will ask you to log in. Do so with the same account you used for app creation.

2. You can choose to view "My Apps", "Public", and "Shared". Choose a view, then click 'Sync'



to download the most updated version of each app.





3. This deployment approach is best suited for development and testing, especially if you build many apps.

Note: once you are using the app you have launched from the app gallery, and if you are using an Android device, you have the option to add that app's icon to your homescreen for future ease of use. To do so, *please refer to this article*.

Related articles

- *Sharing and distributing your app*
- *Deployment from an install link*
- *Deploy your app*
- *Reflecting your brand*
- *Viewing external content*

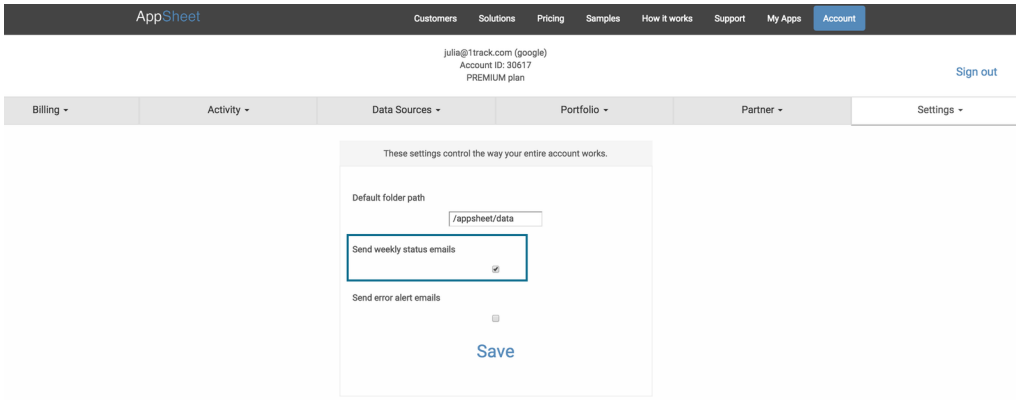
Once You Have Users

Automated usage summary emails – AppSheet

For any deployable app (one explicitly checked and marked as deployable), we will send you a weekly status summary email. The content of the mail is identical to the contents of the Activity tab on the Account page. There are links in it to further details about each of the apps.

These will give you a 10,000 foot view of your apps on a regular basis without becoming too noisy.

You can disable these emails from your Account page on the Settings tab:



Related articles

- *App updates*
- *How can I receive (or send) an email alert when data changes?*
- *Expressions*
- *"Un-deployin"g your app*
- *Pricing, billing, and subscription plans*

App updates – AppSheet

As the creator/owner of an app, you can make changes to the app definition at any time. These changes take effect immediately. For example, if you add a change workflow rule to your app, it starts to work the moment you save the change.

Most of your changes to the app definition involve changes in the actual mobile app. These changes are automatically picked up by your users the next time they Sync their app from their mobile device.

As described in the section on Sync, it is possible that an update to your app definition can disrupt your app users. So you should be thoughtful in making certain app updates. Here's the scenario. Your app user has one version of your app and captures some new data. Before they Sync the data, you make a change to the app definition (let's say you

change some columns in the spreadsheet and regenerate the app). Now when your user tries to Sync the change, there is a mismatch and AppSheet shows the user an error message. The only remedy is for the user to discard the changes (via the Reset Changes menu option).

Related articles

- *"Un-deployin"g your app*
- *Errors during Sync*
- *How do I make sure each user sees only their own data?*
- *Conditional formatting*
- *Who can discover and install your app?*

"Un-deploying" your app – AppSheet

If you want to permanently remove an app, you can simply delete it. The next time any existing users sync the app, it will no longer run.

If you just want to remove specific users from an app that requires user sign in, then removing them from the app whitelist suffices. The next time they sync, the app will prompt them for signin and that will fail.

Related articles

- *Transferring app ownership to another user*
- *Including PDF's in your application*
- *How do I design a secure app?*
- *Making your app definition public to troubleshoot issues*
- *Column types*

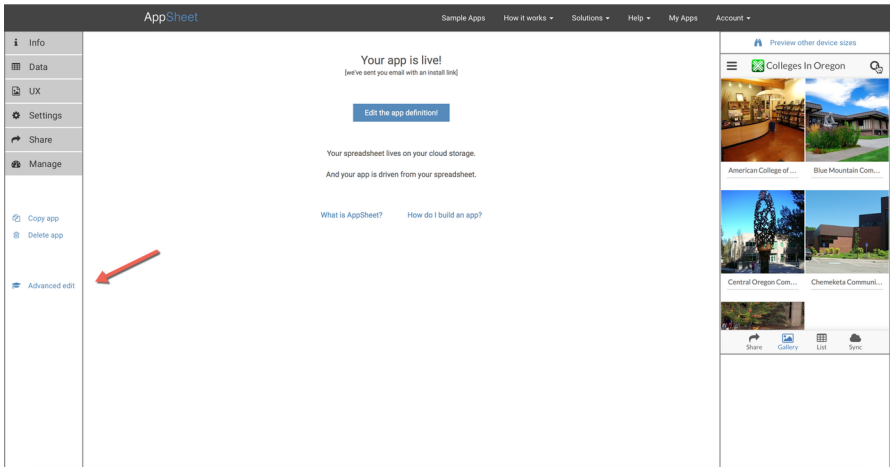
Application Lifecycle

Team collaboration, shared authoring of apps – AppSheet

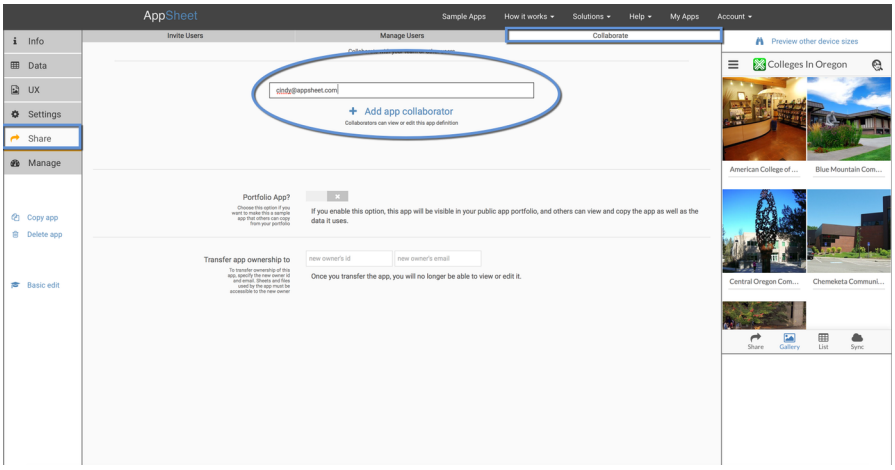
Oftentimes, app creators would like to share app authoring with other team members. With team collaboration, app creators may enable app view or edit rights for other team members. The app creator remains the app owner and may grant or remove access rights for others. View App Collaborators may view the app definition and the supporting spreadsheets, while Edit App Collaborators may edit the app definition and the supporting spreadsheets.

Please be aware that it is wise to have only one user edit an app at any point in time. If two users are editing the same app at the same time, attempts to save conflicting edits will fail. App Editors are not notified if there is another editor changing the app.

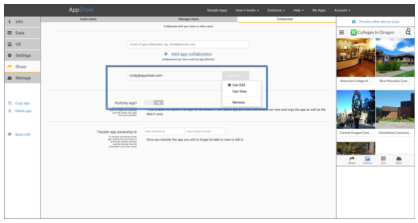
1. To enable this feature, select the **Advanced Editor** from the left sidebar.



2. Next, choose **Share** > **Collaborate,** enter the email addresses of your collaborators and select "**Add app collaborator**."



3. Once added, you may choose to either allow your collaborators to edit or view the app. The default selection is **"Can Edit."**



Note: **Collaboration rights apps** will be displayed in the **My Apps** section, but currently are indistinguishable from **owned apps.** In addition, neither the app owner's name nor the specific right (view or edit) is displayed.

Related articles

- *How do I change a Column Type?*
- *Printing your data*
- *Many identical apps*
- *Making an "AppSheet-friendl"y spreadsheet*

Managing App Versions – AppSheet

As you edit your app definition in the basic or advanced editor, AppSheet maintains earlier versions of the app. This is useful if you need to recover edits that you've overwritten. The Edit History is accessible from the Manage pane in the app editors.

HOW DOES IT WORK?

Every time you save any information via the basic or the advanced editor, AppSheet saves away the current version in an edit history log. Sometimes these changes are explicit. Sometimes they are doing automatically by the system (eg: when checking the consistency of your app).

WHAT CAN YOU DO WITH AN OLDER VERSION?

You can open and view any older version. This opens the editor in a new browser tab and displays that version of the app in read-only mode. The emulator shows a preview as well. You can choose to "Restore" the app --- i.e. make this version of the app the new latest version. The Restore action is also available via the Manage pane in the app editor when you have opened the specific app version.

WHAT CAN'T YOU DO WITH AN OLDER VERSION?

You cannot edit a specific version of the app! You can only edit the latest (i.e. current) version. Further, if the underlying data spreadsheet has changed its structure since then, the app itself may not run in the emulator.

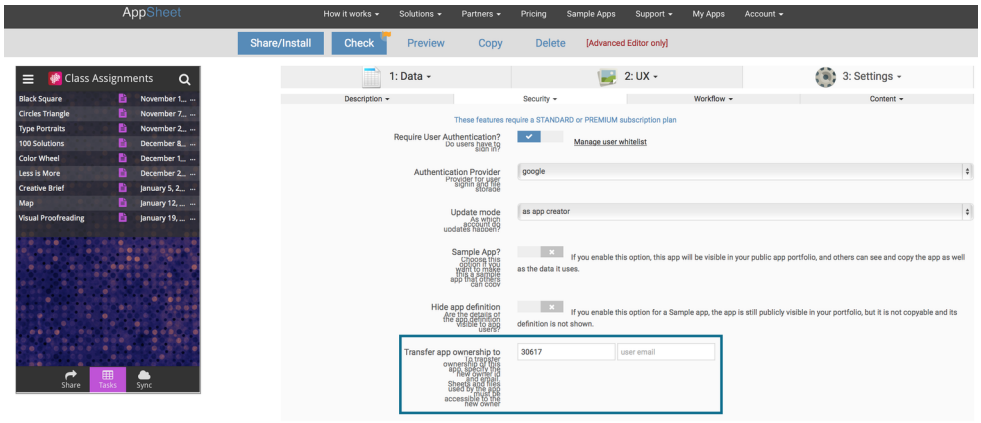
Related articles

- *Transferring app ownership to another user*
- *Information for an IT department*
- *Slices*
- *Plan upgrade required*
- *Team collaboration, shared authoring of apps*

Transferring app ownership to another user – AppSheet

You might want to transfer your ownership of any app to another user for a variety of reasons. This means the new user will have full control of the app's permissions and setup.

To transfer ownership to another user, use the Advanced Editor>Settings>Security tab. You will need to specify the person's user number and email address (both are shown on the Account page). By requiring both pieces of information, we ensure that you must communicate with a user before transferring the ownership of an app to him/her. Once you hit 'Save' the new user will have full ownership.



It is very important that the new user also have access to the files (sheets, images, PDFs, etc) and folders used by the app. Otherwise the app will stop working the moment ownership is transferred. If you are working with Google Sheets and are transferring to user B, make sure to share the sheet(s) with user B. It is also important that user B accepts the shared sheet via "Add to my Drive".

Related articles

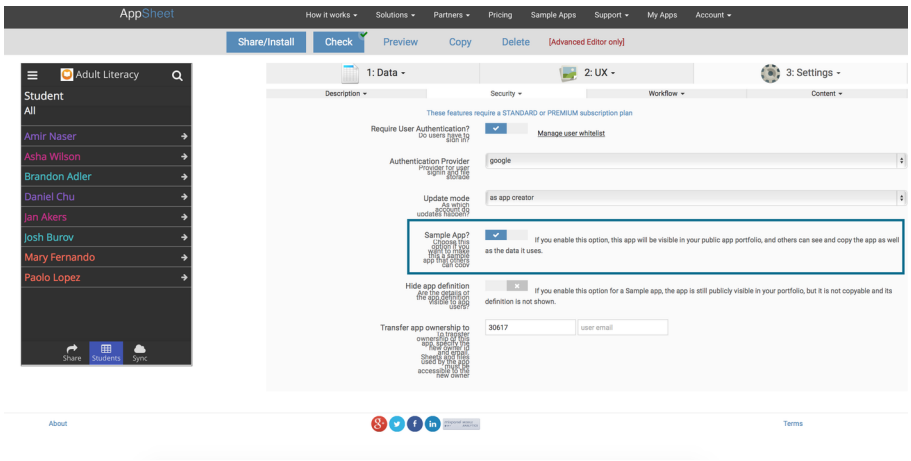
- *Making your app definition public to troubleshoot issues*
- *Change alerts and workflows*
- *Expressions*
- *Check your app for deployment*
- *Sharing and distributing your app*

Using your portfolio – AppSheet

You can choose to make your apps public and display them on your AppSheet portfolio in order to share them with others. Your public apps will act as samples and users who choose to view or copy them will not have write access to your data nor will they be able to save edits with your app. They will simply be able to clone your apps or view them for instructional purposes.

Please be aware that making your app public also means that the data within the app is also visible to a public audience. Do NOT do this if you have confidential data.

To make an app public, you'll need to use the Advanced Editor>Settings>Security tab, and click the 'Sample App?' check box.



How to find your portfolio and edit its description

Your portfolio can be found at the following link: https://www.appsheet.com/portfolio/your user ID number

You can add a description, a photo, and even an external URL that navigates to your personal website. When you're logged into AppSheet, click the Account tab at the top, then Settings. There you'll be able to add the pertinent information.

Related articles

- *Making your app definition public to troubleshoot issues*
- *Deployment from an install link*
- *Launching AppSheet apps from other AppSheet apps*
- *Controlling data inputs with column constraints*
- *Change alerts and workflows*

Troubleshooting

Getting and giving help

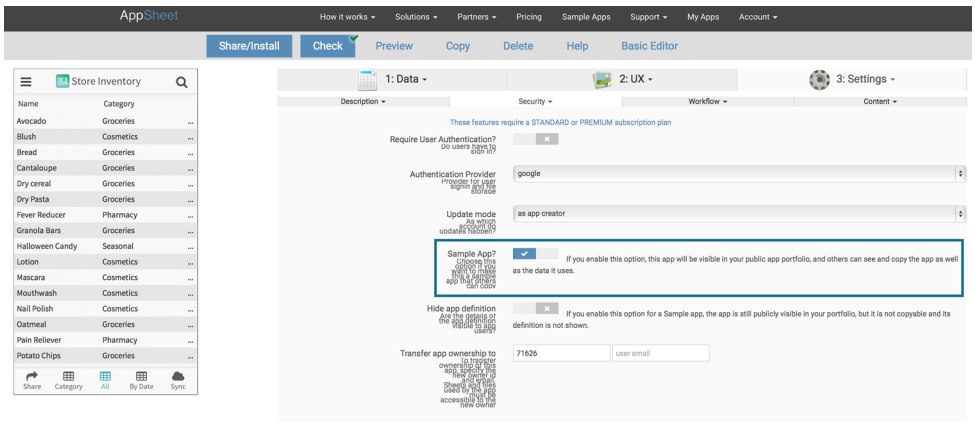
Making your app definition public to troubleshoot issues – AppSheet

Often, there is a need to show someone else your app's definition. Another common use case is when you want someone else to see your app definition and make suggestions for improvements (eg: when posting questions to the user community). You can do this in two quick steps:

1. Mark your app as a 'sample' to appear in your *AppSheet portfolio*. See information about this below.
2. Copy a link to the app definition (from your browser url bar) and share it with whoever needs to help with the troubleshooting (usually, add it in a post on our user community forum).

Please be aware that making your app public also means that the data within the app is also visible to a public audience. Do NOT do this if you have confidential data.

In the Advanced Editor>Settings>Security tab, click the 'Sample App?' check box. The app definition now becomes publicly visible via your portfolio page: https://www.appsheet.com/portfolio/<userid>. Or you can simply share/email the browser link to the app editor page.



Related articles

- *Using your portfolio*
- *Sharing and distributing your app*
- *Virtual columns*
- *Transferring app ownership to another user*
- *Expressions*

General

Errors accessing a spreadsheet – AppSheet

AppSheet can experience errors reading or writing spreadsheet data from any cloud storage provider.

- Error 401 -- this occurs when AppSheet does not have permission to work on behalf of the user.
- Error 403 (permission denied) -- this occurs when AppSheet tries to access your spreadsheet, but the storage provider responds that AppSheet does not have the permission to access it.
- Error 404 (cannot access the spreadsheet) -- this occurs when AppSheet tries to access your spreadsheet but the storage provider responds that no such document exists. Perhaps you have deleted the document? Perhaps you never had access to it?
- Timeout -- occasionally, the remote storage provider has a timeout and fails to provide the spreadsheet data to AppSheet despite repeated retries. Although fewer than 1% of our requests encounter such issues, it is still significant enough that you may encounter such a problem. You may experience this problem as an unexpected delay during Sync, or even a sync failure. Typically a refresh or retry after a few seconds rectifies the problem.
- Error 500 (internal server error) -- very occasionally, the remote storage provider has an internal error and fails to provide the spreadsheet data to AppSheet. Typically this is a transient problem and often is hidden from our users because we automatically retry the request.

Related articles

- *Make a spreadsheet*
- *Google Drive*
- *Plan upgrade required*
- *Deploy your app*
- *Using a scanner*

Errors and warnings in the editor – AppSheet

AppSheet automatically extracts structure from a spreadsheet so that your app can be generated. While most spreadsheets are simple tabular structures, there are also ways to use spreadsheets that are incompatible with AppSheet. So you may see errors, warnings, and informaton messages in the app editor when AppSheet reads your spreadsheet.

1. Errors with spreadsheet structure -- these errors typically occur when AppSheet is unable to properly identify the column headers. Often this is because the first worksheet in your spreadsheet has non-tabular data like charts or pictures.
2. Errors when switching spreadsheets -- when you switch to a different spreadsheet, AppSheet tries to retain as much as possible of your existing app. However, the new spreadsheet may have a totally different column structure. In this case, you may see errors that ask you to change the other parts of the app definition appropriately.
3. Warnings about column structure mismatch -- this is the most common warning seen during the app creation process. As users modify their spreadsheets, they add or remove columns. Each time you do this and tell AppSheet to refresh the column structure, you are warned if the old column structure and new structure do not align. AppSheet always adjusts to the new structure, so this is a warning, not an error.
4. Warnings about formulas -- we discuss formulas in a separate topic, but in brief, there are only certain kinds of spreadsheet formulas that make sense to use in your data.
5. Information about the key column -- as described earlier, this information tells you which column is chosen as the key. The key also plays an important part in the presentation of data as described in the UX section. If AppSheet cannot find a good key column, it tries to combine pairs of columns to find a composite key. If that also fails, it settles for the row number (an implicit column). This is typically not a good choice unless you have an app that is purely showing information and prevents all edits. The number of a row in a spreadsheet changes dynamically based on the rows above it, so clearly the row number identifies a row only as long as no rows are being added or deleted to the spreadsheet by the app, or directly.

Related articles

- *Reflecting your brand*
- *Plan upgrade required*
- *Offline behavior*
- *How should I use multiple sheets in my app?*
- *Making an "AppSheet-friendl"y spreadsheet*

The app is not runnable – AppSheet

You may see this error in the app emulator (in the editor web page) or your users may see this error on a mobile device.

It indicates that the app definition is invalid (most likely) or corrupt (rare).

The most likely reason for an app definition to become invalid is that the structure of the spreadsheet used to create the app changed. For example, someone added or deleted a column. Now the spreadsheet is no longer compatible with the app. Every time you open the app in the app editor, or refresh the app editor web page, AppSheet rechecks this and alerts you with error messages if the app definition is invalid. In many cases, AppSheet wil automatically adjust the app to make it runnable again. However, this does not happen unless the app creator actually opens the app editor --- i.e. it requires an intentional action by the app creator.

AppSheet will also consider an app definition as invalid if the underlying spreadsheets are no longer readable (if the data cannot be read, then clearly the app cannot be verified let alone run).

On a mobile device, an app definition may be corrupted if a network error occurs while the app definition is being downloaded (initially or during a sync). This is a relatively rare event and usually is resolved with a restart or resync. In rare cases, you may need to uninstall and reinstall AppSheet.

Related articles

- *Application crashes / errors*
- *Errors accessing a spreadsheet*
- *Plan upgrade required*
- *Deploy your app*
- *Make a spreadsheet*

Plan upgrade required – AppSheet

AppSheet has different subscription plans as described on the *Pricing page*.

All basic app functionality is available for free for prototypes and purely personal use. All use in business and professional settings requires a paid plan. All the same, every app feature and behavior can be tested for free until you are ready to deploy. In short, your app will function even if you haven't yet selected a plan. **If features of your app do not work, it is not because of the plan you are on.** Of course, it is good karma to be on the right plan, and good karma always helps. :-)

The app editor indicates that certain app behaviors (eg: security, offline, content optimization, team collaboration, etc.) always require paid plans. When your app definition enables these features, AppSheet knows for sure that you are not building an app as a prototype or for personal use. The use of these features may lead to a warning message that tells you that a plan upgrade is required. Even if you are not using these features, if your app is used in a business setting, you should be on a paid subscription plan.

The per-user subscription plans involve a variable cost component depending on the number of unique users of the app. AppSheet monitors this usage and you, the app creator, can see approximate historical usage statistics for each of your apps on the Activity tab of your Account page. At the current moment, AppSheet does not warn you when the user limits are reached, but will do so in the future.

We do not ever plan to prevent access to the app if the user limits are reached -- rather, we expect to record and warn the app creator of this situation. However, the preferred route is for the app creator to anticipate and purchase the appropriate plan based on the expected usage.

Related articles

- *Reflecting your brand*
- *Errors and warnings in the editor*
- *Offline behavior*
- *Expressions*
- *Column types*

Errors during Sync – AppSheet

As described earlier, Sync has three stages:

1. Send changes from the device to the backend. When this stage starts, the Sync progress bar is about 1/4 complete.
2. Fetch the latest app definition from the backend. After this stage, the Sync progress bar moves to 3/4 complete.
3. Fetch the latest data from the backend.

Most errors in Sync are observed in the first stage (i.e. when the progress bar is 1/4 complete).

1. If there are network errors, the step is retried a few times, and if the failure persists, the Sync is halted. The solution is to retry later when connectivity is better.
2. If you have poor connectivity or if large images have to be sent as part of the Sync, this step can take a long time and can potentially timeout. The solution is to retry later when connectivity is better.

If the app definition has changed on the backend (for example, new columns have been added to the data), the local changes from the device may no longer be executable. In this case, an error message is shown to the app user during Sync. It usually takes the form of an error indicating that a value being inserted is incompatible with the type of data expected (because the columns have changed!). If you use the advanced menu in the app (top left) and choose 'About', you will see the basic app information including the version number. The version number indicates the accurate version of the app that is being run. The creator of your app can look at the latest app version number via the app editor's Manage pane. This is the best way to determine if there is a mismatch between the app version on your device and the app version expected by the AppSheet backend during sync. There are two possible remedies to this situation:

1. Restore the old version: the app creator can restore the required version of the app. This will work only if the underlying data sources (spreadsheets) have not changed their column structure. Once the old version is restored, the sync should proceed successfully.

2. Manual recovery: if all else fails, you will have to abandon the changes on the device via "Reset Changes" (in the advanced menu at the top left of the app). However if you cannot afford to lose these local changes, first click on the "Show Changes" menu option. This will give you the option to send yourself a copy of the changes via email, so that you can manually process this data.

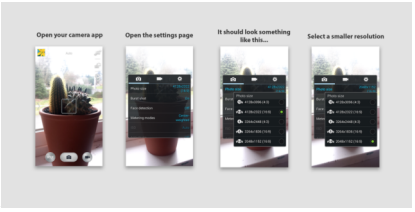
Related articles

- *Application crashes / errors*
- *Sync-- between the app and the backend*
- *Controlling data inputs with column constraints*
- *How should I choose a key?*
- *App formulas*

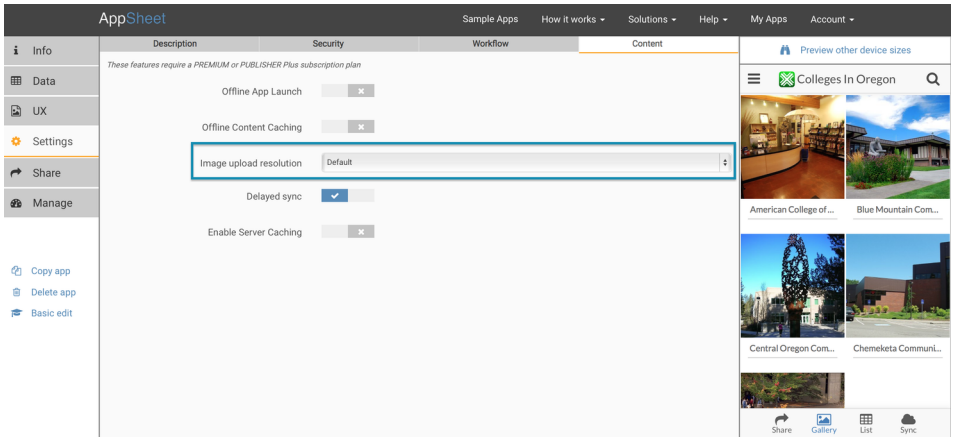
Mobile device issues

Application crashes / errors – AppSheet

If you are seeing application crashes there are a couple of steps you can try to resolve these issues:

1. Close High memory applications - application crashes on a mobile device are usually due to high memory usage, and closing background applications can help free memory. For Android devices there are some step by step instructions at: *http://blog.laptopmag.com/how-to-close-android-apps* and for iOS at: *https://support.apple.com/en-us/HT201330*
2. Reboot the device - rebooting can also free system memory, and may resolve the issue.
3. Reinstall the appsheet app - fully remove the appsheet app and reinstall it from your app store (Google Play or the Apple App Store). Note that this will delete any local data, so don't do this if you have data changes on your device that still need to be synchronized to your spreadsheet.
4. Update your OS - there may be an operating system update that resolves the issue. For Android devices you'll need to check with your device manufacturer (Samsung, LG, HTC etc.). For iOS there are instructions at: *https://support.apple.com/en-us/HT204204*
5. If that didn't work, and your crashes are related to photos on Android, try reducing the photo resolution in your photos application. Following is an example of how to do it on a Samsung Galaxy phone. Other camera apps on other phones work differently, though most should have the option to change resolution. 

If you are capturing a large number of photos you can also try lowering the Image Upload Resolution setting on the Settings \ Content page of the Advanced Editor.



If the steps above haven't resolved your issue please contact us at support@appsheet.com. When you contact support please let us know which device you are using and what OS version you're running. For example "Samsung Galaxy S6 running 5.1.1" or "iPhone 6 Plus running iOS 8.3".

It will also help us to know if the crash reproduces in a web browser on your device. To test this open Chrome on your Android device or Mobile Safari on iOS. From the browser on your device login to *appsheet.com* and in the "My Apps" section of the website click "Preview" for your app then "Fullscreen". This should start your app in your mobile browser. Use your app as you normally would and let us know if you see the problem while running in the browser as well.

Related articles

- *Errors during Sync*
- *Google Drive*
- *App formulas*
- *Make your list into an app*
- *Controlling data inputs with column constraints*

Issues with specific providers

Google Drive – AppSheet

Users of Google Drive can sometimes see Error 403 (permission denied) or Error 404 (not found). There are a few common reasons for this:

1. You are not the owner of the document but this is a shared document owned by a different Google account. Check your document sharing permissions on Google.
2. You have multiple Google accounts (let's call them Acct1 and Acct2); your spreadsheet is owned by Acct1 but you sign into AppSheet using Acct2 to access the spreadsheet. Either use the same account, or share the document across accounts.
3. You had access to the file at one point, but the access has been removed.
4. This can sometimes occur if you have a corporate Google Docs account that has an admin policy that prohibits all users from enabling third party software like AppSheet (so you may need to talk to your account admin).

Click here to learn more about enabling Google permissions.

If you use on of the AppSheet add-ons to integrate with Google Forms, Sheets, or Docs, please also read *this article* to learn more about specific limitations in those integrations.

In particular, if you utilize Google AppScript onEdit triggers on your Google Sheet, or a third-party addon that utilizes these triggers, those triggers are not fired when AppSheet makes a change to the data in the spreadsheet. This is an unfortunate limitation imposed by Google on all third-party solutions like AppSheet. The best workaround is to utilize a timed AppScript trigger rather than an onEdit trigger.

The Google onChange "Simple Trigger" is fired when AppSheet makes a change to the spreadsheet. When Google Sheets fires the onChange trigger, it passes an "event object" to the invoked onChange script that contains information regarding the onChange event. See the Google documentation for "App Scripts" for more information regarding scripting.

Also, do not use filters on your Google Sheets --- it can make the filtered rows invisible to updates. Instead, use filter views (this is a Google Sheets feature that lets each user create and use their own filters without affecting other users of the sheet).

Related articles

- *Plan upgrade required*
- *Customizing input forms*
- *Dropbox*

- *How do I design a secure app?*
- *Deployment from an install link*

Dropbox – AppSheet

Related articles

- *Sign in*
- *OneDrive*
- *Google Drive*
- *Controlling data inputs with column constraints*
- *Reflecting your brand*

OneDrive – AppSheet

Related articles

- *Integrating with Excel and Office 365*
- *Google Drive*
- *Box*
- *Locale support in AppSheet*
- *Sign in*

Box – AppSheet

Box.com users tend to have corporate accounts and see a greater incidence of two specific kinds of errors. One error occurs when copying a sample app or making a new app. By default, AppSheet creates a folder for each app under the /appsheet/data path, so that the sample spreadsheet and data can be copied into it. In many corporate Box installations, users do not have permissions to create a folder in this path. Instead, each user (say 'joe') has permissions to a path that may be '/users/joe'. So you will need to change the default folder path for your AppSheet account appropriately by accessing the 'Account' tab on the top bar of our website. The other error reported by some of our Box users is that Excel spreadsheet files seem to be automatically converted into CSV files on some Box installations. Users can check this by just trying to download the Excel file from Box to see if its format has been modified -- if so, please consult with your Box admin.

Related articles

- *Make a spreadsheet*
- *Smartsheet*
- *Who should use AppSheet and what kind of apps can it create?*
- *How do I make sure each user sees only their own data?*
- *Including PDF's in your application*

Smartsheet – AppSheet

Related articles

- *Integrating with Smartsheet*
- *Google Drive*
- *How do I design a secure app?*
- *Grouped views and data filtering*
- *Virtual columns*

Frequent App Design Questions

Common App Usage Scenarios

Including PDF's in your application – AppSheet

There are a three ways you can include PDF's in your application.

1. Link to the PDF using a relative path

Using relative paths, you can create links to files relative to the spreadsheet that your app is using. This works in a similar way to links to image. For example, if the PDF is in the same directory as your spreadsheet, you can create a link to it just by putting in its name in your spreadsheet, eg. âÄòfoo.pdf'.

2. Link to the PDF using a direct URL

Using direct Url's you can link to a pdf that is somewhere on the internet. Just like you would link to an image, you can link to a pdf, for example. âÄòhttp://www.mywebsite.com/path/super_pdf.pdf'

3. Use a Google Drive URL

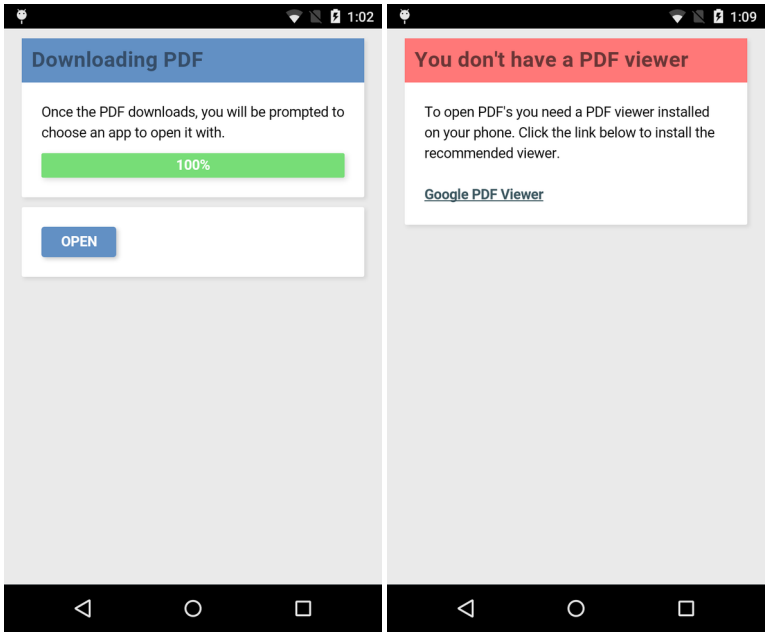
These are url's that you copy from your address bar in your browser when you are previewing a pdf in google drive, eg. âÄò*https://drive.google.com/view?id=1234*'

Note that these are not direct or relative links, which means you are not supplying us with the actual PDF file. This means that you won't be able to take full advantage of AppSheet's PDF support.

How does it work?

iOS - Regardless of the type of URL, the pdf will be opened in Safari. However, only relative and direct URL's are guaranteed to be cached for offline use.

Android - If you link to the PDF using a direct URL or relative path, the PDF will be downloaded to your phone, and you will be able to choose a viewer with which to view it. If you do not have a PDF viewer installed on your phone, we'll give you a link to the recommended viewer.



Google Drive URL's will open correctly, however you will not be able to open them with a PDF viewer on your phone, and they are not guaranteed to be cached for offline use.

Related articles

- *Viewing external content*
- *Printing your data*
- *Advanced app customizations*
- *Using spreadsheets with pivoted data*
- *Limits on data size*

Make your list into an app – AppSheet

Everyone has a list in a spreadsheet-- a client list, an item list, etc. Here are some suggestions for building useful list apps.

Consider multiple grouped list views instead of or in addition to the default tabular view. Grouped views help you divide your data into different categories.

If you have a list of people:

- Their phone numbers should be touch-callable in the app. If not, it is probably because your telephone number format was not automatically recognized by AppSheet. No problem, you can explicitly set the type of this column to Phone.
- If there are physical addresses for the entries in your list, consider adding a map view. Your addresses should be fully specified (i.e. if you live in Las Vegas and all your addresses are in Las Vegas, don't just leave out 'Las Vegas, NV, USA' from your addresses). While the shortened address clearly makes sense in your context, our mapping software does better with full addresses.

If you have a list of products:

- Consider adding an image to each product and add an image gallery view. Images usually look beautiful. Play with the different image presentation sizes. They completely alter the feel of the app. Don't worry if you have high resolution images-- we downsize them appropriately for a mobile device.
- If you want to steer users to an external e-commerce site, you can add an action button to each image. Take a look at the *ECommerce Store sample.*

If you have a list of numbers (sales reports, budgets, etc.):

- Consider using one or more chart views-- they will help you visualize the data.
- AppSheet supports different kinds of charts and we are adding more based on customer requests.

If you have a list of instructions:

- Consider adding an image to each instruction and using a List view with a Deck presentation.
- Take a look at the *Repair Manual sample.*

Related articles

- *Apps using dates*
- *Displaying images and documents*
- *How should I use multiple sheets in my app?*
- *Make a spreadsheet*
- *Expressions*

Apps using dates – AppSheet

If your data uses dates or timestamps, there are some special considerations to keep in mind.

Dates in spreadsheets

Date formats vary widely across languages and locales. So it is really important that you use the cell formatting capability of the spreadsheet to indicate that all the cells of a column are Date values. You can utilize dates and times in your local format.

If your data column in the source spreadsheet is formatted as a Date or Time with a specific format (eg: DD/MM/YYYY) we respect that format when your changes are stored back in the spreadsheet.

Many spreadsheets use automatic mechanisms to add a timestamp in the first column. A timestamp is generally a poor choice for a key-- though it uniquely identifies the row, the timestamp isn't really meaningful to the rest of the app. So if the timestamp is chosen as your key, you may need to change that explicitly.

Dates in apps

Dates and times in AppSheet apps are shown in the appropriate locale-specific format.

If you're seeing the wrong date format on your mobile device or browser, make sure your system and browser language is correct. US English date format is different than non-US English format.

Change your language on Android: *https://support.google.com/accounts/answer/32047?source=gsearch&hl=en*

Change your language on iPhone / iPad: *https://support.apple.com/en-us/HT204031*

Change your language on Chrome: *https://support.google.com/chrome/answer/173424?hl=en*

Related articles

- *Using formats and data validation rules*
- *Expressions*
- *Locale support in AppSheet*
- *Controlling data inputs with column constraints*
- *Presentation types*

Using spreadsheets with pivoted data – AppSheet

Most spreadsheet data uses field names as column headers and data entries in each row, but what should your data is pivoted with field names down the first column and data entries in the other columns? For example, if employee names are used as column headers and employee attributes as rows.

AppSheet doesn't work with pivoted data. This is because apps are generated from the column structure of the data. The column structure tells the app the "shape" of the data that it needs to process. This structure needs to stay stable as the app is being used. New data should take the form new rows and changes to data should take the form of row edits or row deletes.

Luckily most data can be re-pivoted so that the attributes of each data item form column headers and the values of each data item are represented in a row. This is very easy to do in a spreadsheet. In Google Spreadsheets or Excel, you first copy the data and then use 'Paste Transpose'.

Related articles

- *Many identical apps*
- *Expressions*
- *Presentation types*
- *How do I control the order of columns displayed in the app?*
- *Including PDF's in your application*

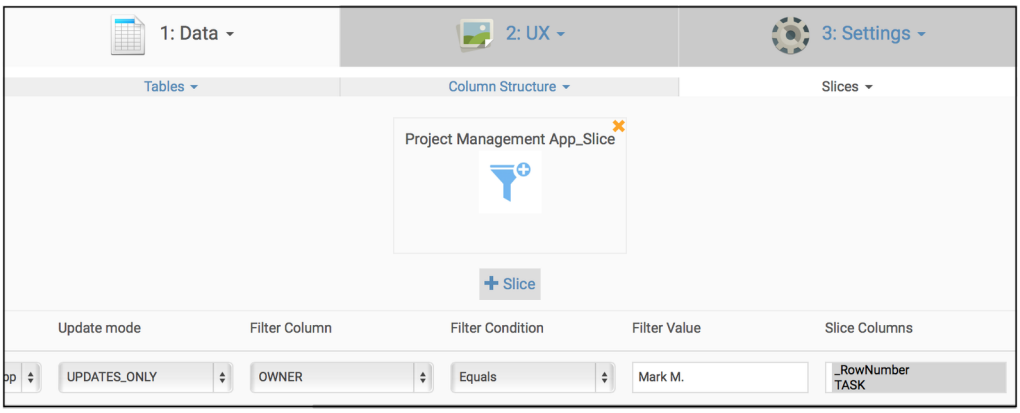
Many identical apps – AppSheet

Do you need to make many similar apps-- identical in structure, but just with different data for each?

Teams may need to have similar apps, but with different data. There are two ways you can achieve this outcome: by making an app for each user, or making one app with only the information pertinent to each user.

- Make app copies using multiple spreadsheets: copy the app repeatedly and switch it to a different spreadsheet for each user. This is simple and gives you a lot of control. You can make changes to each app if you want. But of course, it can be tedious if you have many apps to create and maintain.

- Use row filters for a single app using a single spreadsheet: row filters can achieve a similar effect with a single app. Different users of the app can see different subsets of the data. In particular, consider adding slices in the Advanced Editor using the 'Matches the user name' or 'Matches the user email' condition to filter the right data to the right user. Then send the same link to each user.
- Use row filters for multiple apps using a single spreadsheet: row filters can achieve a similar effect with multiple apps. You can slice the data as in the above example, but instead of using multiple slices, just use one slice per app per user. Then send each user their unique app's link.



Apps with different update permissions for different classes of users

You can also build multiple apps with the same data, but different update permissions for different classes of users. To do this, see *this article*.

Related articles

- *How should I use multiple sheets in my app?*
- *App formulas*
- *Controlling data inputs with column constraints*
- *Expressions*
- *Conditional formatting*