Table of Contents

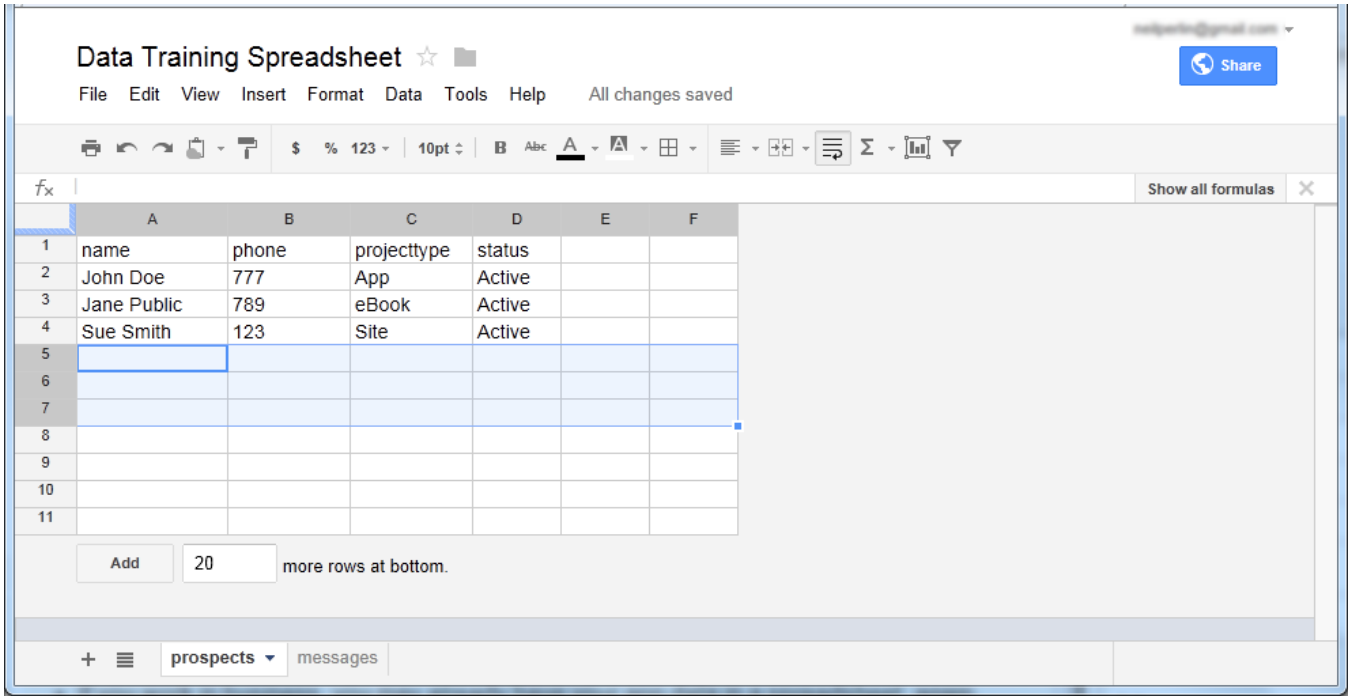
Table of Contents 1
What Google Docs Buys You 2
  What Is Google Docs? 2
Overview of Data Management Operations 3
Spreadsheet Concepts and Syntax Rules 3
  Concepts 3
  Syntax Rules 4
What Is A Query 5
Introduction to Database Commands 6
Select From - Examples 6
Select From - Concepts 10
Select From - Examples 12
Insert Into - Concepts 15
Insert Into - Concepts 17
Insert Into - Examples 19
Update - Examples 24
Update - Concepts 29
Update - Examples 32
Delete From - Examples 38
Delete From - Concepts 38
Delete From - Examples 41
Delete From - Examples - Predefined Value Deletion 42
Delete From - Examples - User-Specified Value Deletion 47
Introduction to Conditions in ViziApps 53
Conditions With Options 53
Conditions On Commands 54
Feedback for Data Operations 60
How Images Are Stored 64
How Maps Are Stored 67
  Manually Creating the URL 69
  Automatically Creating the URL 72
Best Practices for Using Spreadsheets 72
  Planning and Design 72
  Development 72
Google Apps for Business 73
  Availability 74
  "Subcontracted IT" 74
  Security 74
  Accessing ViziApps Through Google Apps for Business 74

77
Example - Sorting Data In a Table 77
Example - User-Added Images 80
Take, Annotate, and Store 81
View a List of All Pictures 84
Get A Larger View of One Picture 87
Example - User-Added Text 90
Add a Restaurant to the List 91
View a List of the Restaurants 94
97

What Google Docs Buys You

You're probably accustomed to using spreadsheets on your PC - the desktop - and manually entering data into them. Google keeps the familiar spreadsheet model but puts it on the web and lets your app users store and retrieve data in it. This makes a Google Docs spreadsheet a data-base.

The image below shows an example from a spreadsheet used to store sales lead information for a simple lead tracking app used elsewhere in this online guide.



What Is Google Docs?

Google Docs is a free, web-based office suite that competes with Microsoft Office. Google Docs has spreadsheet, word-processing, presentation, drawing, forms, and data storage modules. Google Docs is web-based, so users can work and collaborate online with other users in real time.

Overview of Data Management Operations

Using Google Docs spreadsheets to manage your app data is straightforward and uses concepts that you may already know from other business tasks. Here's the overall process, with links to other topics that explain the concepts and steps in more detail.

Here's what you need to do in two high-level steps:

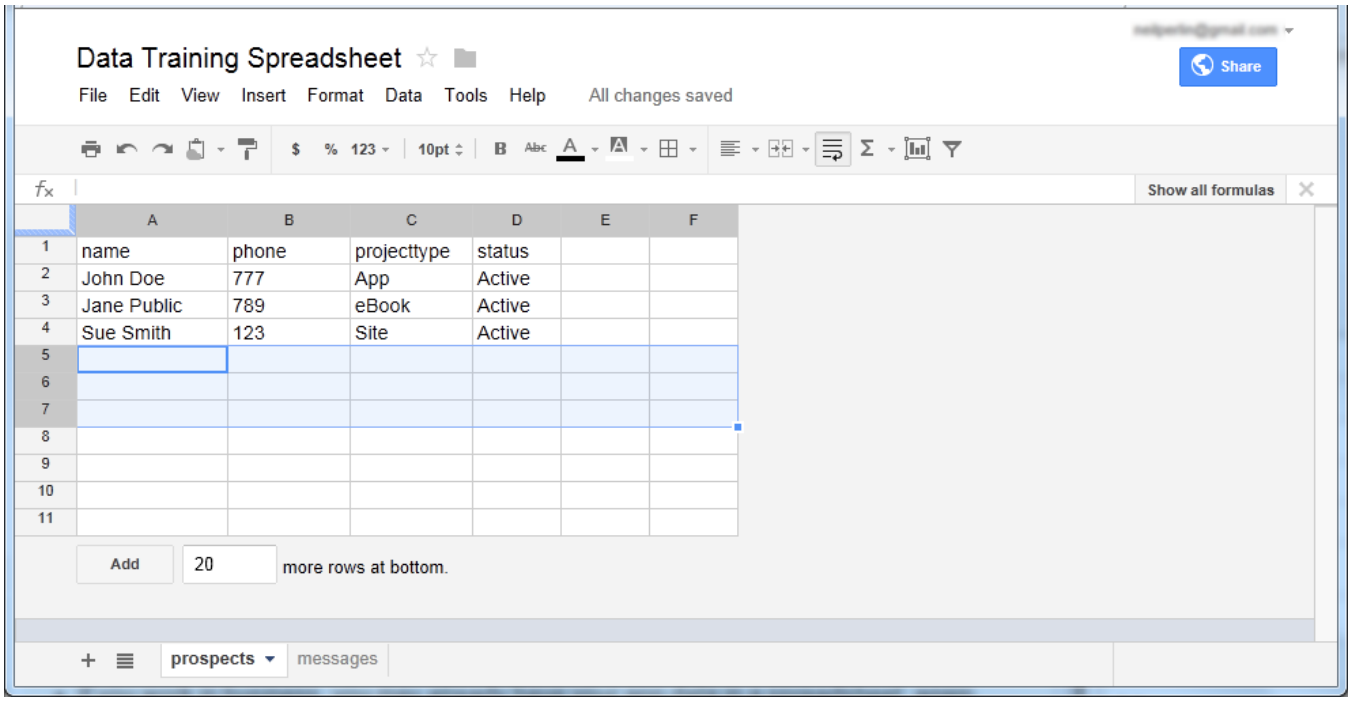
1. Create and structure a Google Docs spreadsheet and worksheets to contain your data. To get started, read these two topics in this order.
   - See "Spreadsheet Concepts and Syntax Rules" on page 3
   - See "Best Practices for Using Spreadsheets" on page 72
2. Use ViziApps' Data Management window to perform "queries" that write data to or read data from the spreadsheet.
   - See "What Is A Query" on page 5
   - See the topics in the Database Commands section for details about the four commands you can use.
   - See the topics in the Conditions section for information about refining your command settings.

Spreadsheet Concepts and Syntax Rules

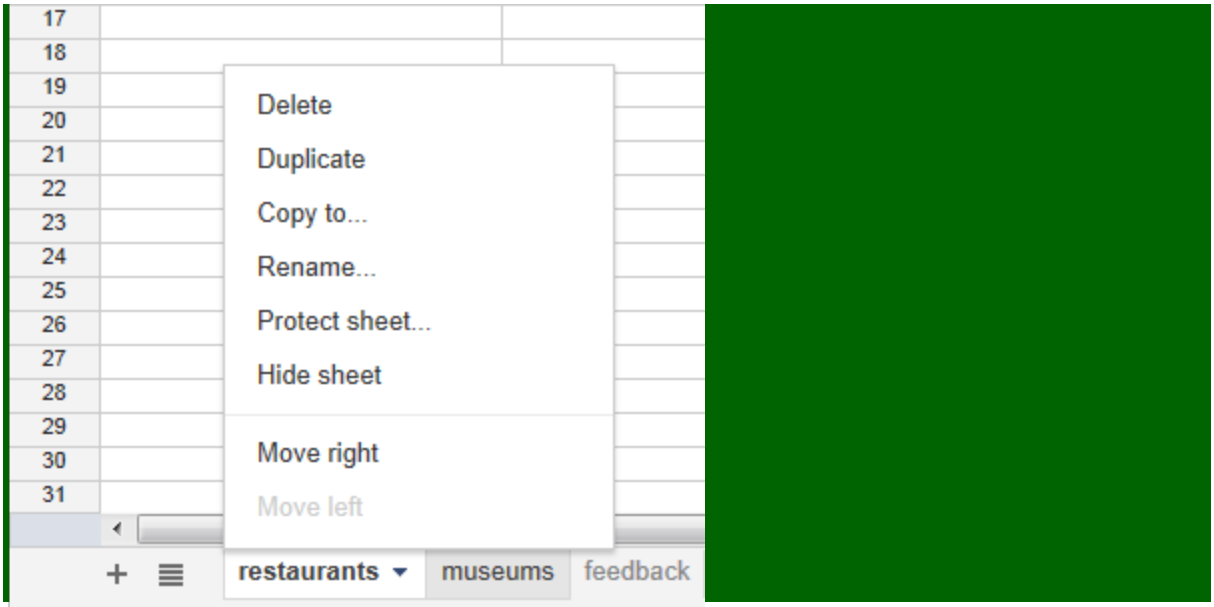
A Google Docs spreadsheet has a few simple concepts and rules.

Concepts

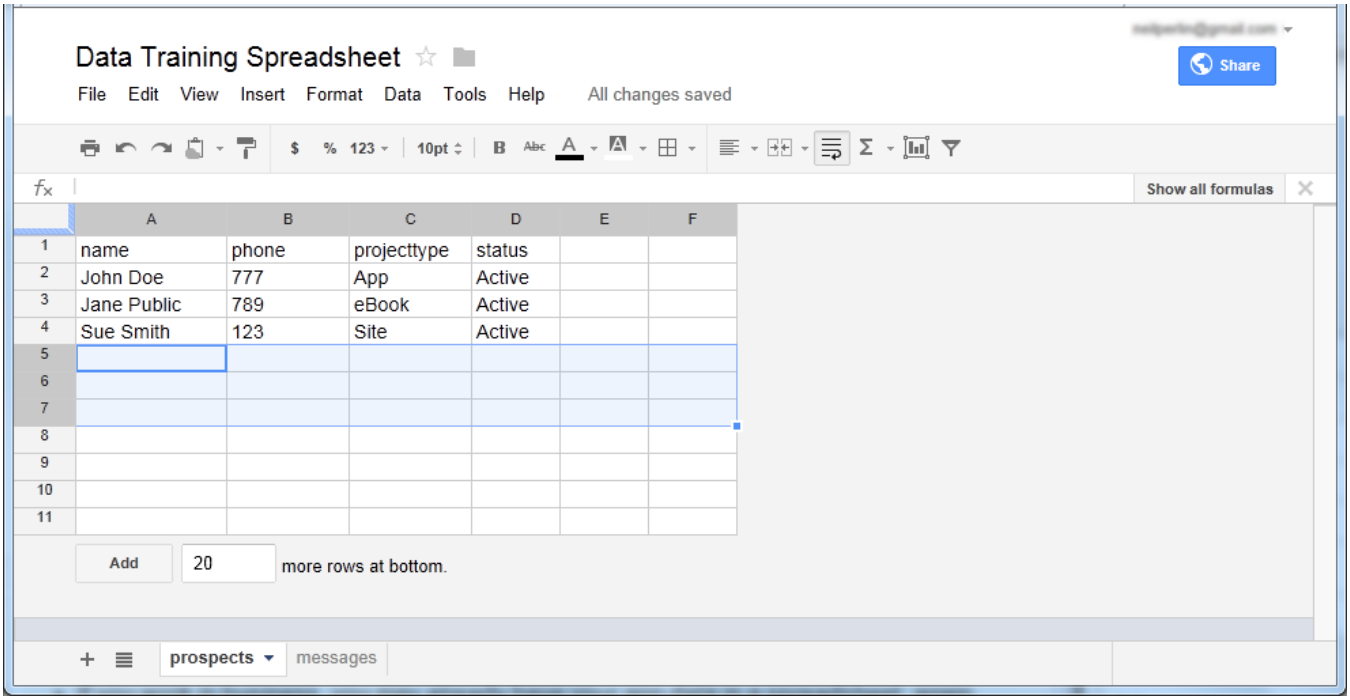
- A spreadsheet must have at least one worksheet. Simple data may just need one worksheet. More complex data may call for several. It's up to you, based on how you want to organize your data in order to create an easily understood and maintained structure. For example, the spreadsheet shown below has two worksheets. They're listed on the tabs at the bottom left - *prospects* and *messages*.



- To manage a worksheet, click the pulldown on its tab. The image below shows the options.



- Each worksheet column is a "field" that contains one type of data. The field names are in the first row, field values in subsequent rows in the same column. The example below has four fields whose names, in row 1, are *name*, *phone*, *projecttype*, and *status*. The field values are in row 2 and down for each column, such as John Doe, Jane Public, and Sue Smith as values for the *name* field.



Syntax Rules

There are a few simple rules when you create a spreadsheet or add fields.

- Type multi-word field names with no spaces. For example, enter "project type" as "*projecttype*" as shown in the example above.
- Field names are case-insensitive. Google Docs treats "*name*" and "*Name*" the same. But to ensure consistency, enter field names in lower case - "*name*" rather than "*Name*".
- When adding data for a field, try to not use "-" or "_" because Google may not read them.
- When you create a new worksheet and fields, each field must contain at least one row of data before you can use that worksheet in your app. If it doesn't, you'll get a Google error message. You can always enter a row of placeholder data just to start using the worksheet, such as "abc" in each column.

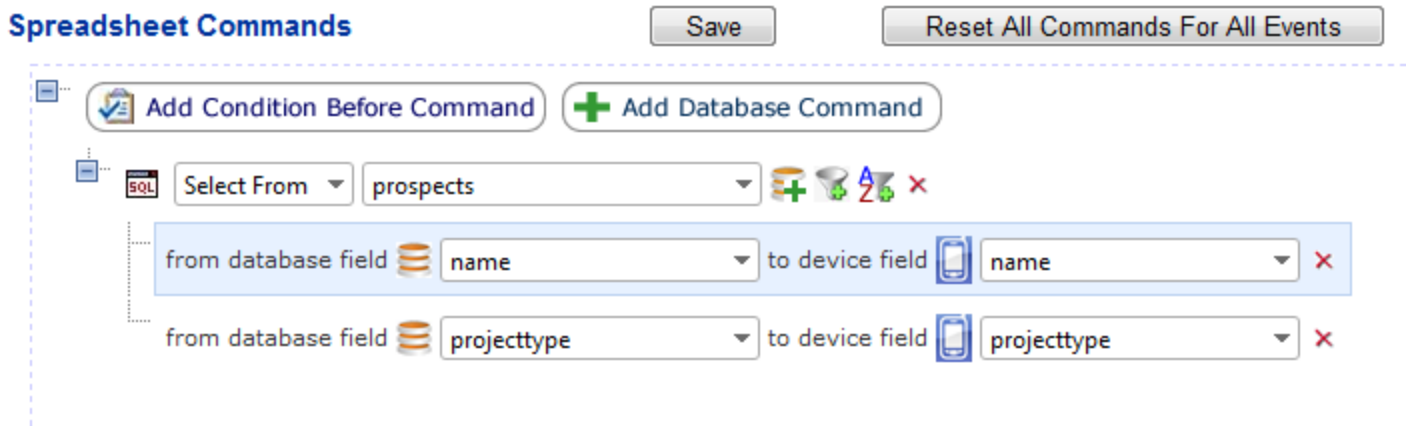
What Is A Query

The word "query" implies a request for information. However, in the computer world, a "query" is an action that affects data in a database - writing or reading data to or from a database, changing data, or deleting data. In ViziApps, these actions are controlled by four commands:

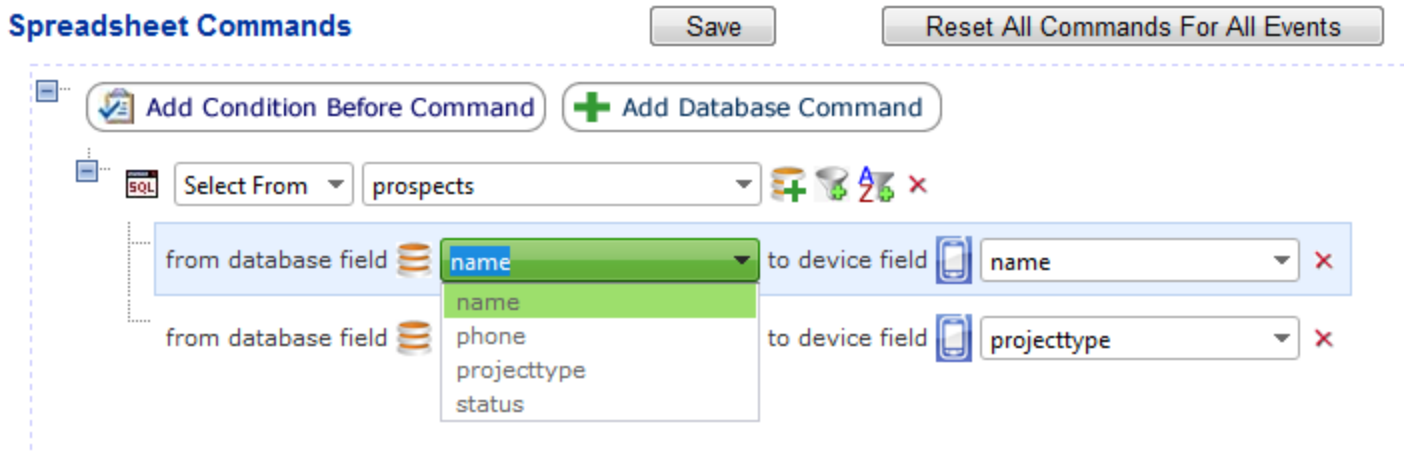
- Delete From
- Insert Into
- Select From
- Update

There are three general ways to create queries. ViziApps' Google Docs spreadsheet approach uses the easiest - choosing parameters from a menu. ViziApps simply presents a list of options from which to choose.

For example, the image below shows the completed entries for a query that tells ViziApps to select data from the prospects worksheet's *name* and *projecttype* fields and write that data to the *name* and *projecttype* fields on the mobile device for display to the users.



To specify the fields from which to select data, just click the pulldown for each field. The image below shows all the available fields on the *prospects* worksheet. They're displayed by clicking the pulldown on the first "from database field" option.



Creating queries this way is simple because you don't have to be a programmer.

Introduction to Database Commands

This section introduces the concepts and mechanics behind the four ViziApps commands for working with data in a Google Docs spreadsheet. We'll illustrate the mechanics with examples from a simple app, a contact management app that you'd use to keep track of leads at a trade show.

Select From - Examples

How did the example in the Select From - Concepts topic work? See "Select From - Concepts" on page 10

The image below shows the spreadsheet for this project.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	978-654-3310	Site	Active		
6						
7						
8						
9						
10						
11						

Add 20 more rows at bottom.

Notice the field (column) names - *name*, *phone*, *projecttype*, and *status*. (We'll ignore the *status* field in this example.) This spreadsheet also has two worksheets - *prospects* and *messages*.

We'll use *prospects* in this example.

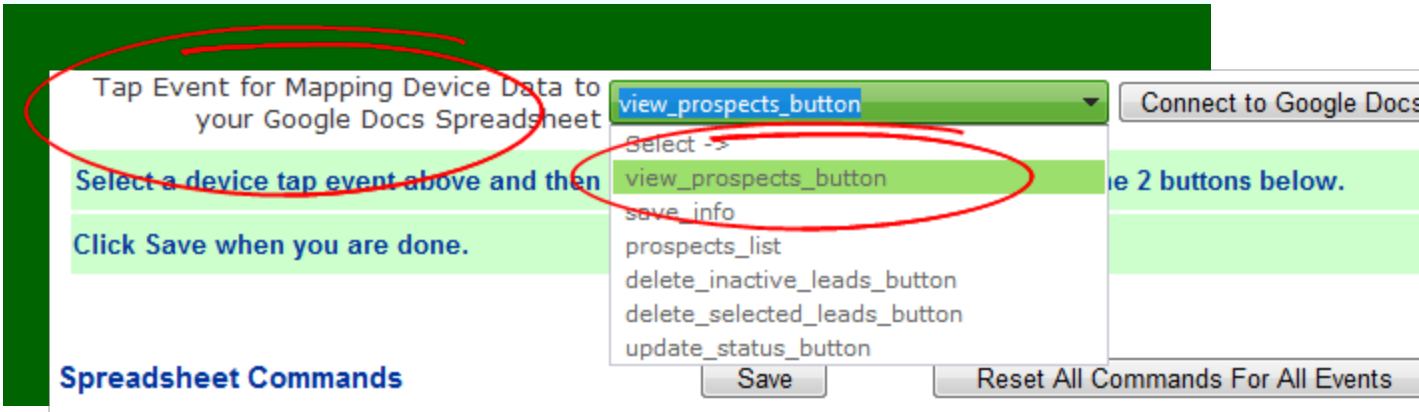
An app should contain fields that are to contain the data extracted from the spreadsheet and that correspond to the field names in the spreadsheet. In this example, we want to extract the users' names and project type of interest and display that information in the app. (We're also doing this for all users listed in the spreadsheet, rather than trying to display a specific user's information.) So there should be fields in the app that correspond to the name and project type fields and that will contain and display the data extracted from the spreadsheet. They're called *name* and *projecttype*.

Tip: Give fields names that make it clear which fields in the app connect to which fields in the spreadsheet. For example, it might be better to name these two table fields *name_column* and *projecttype_column*. (The actual names aren't crucial as long as they're clear.)

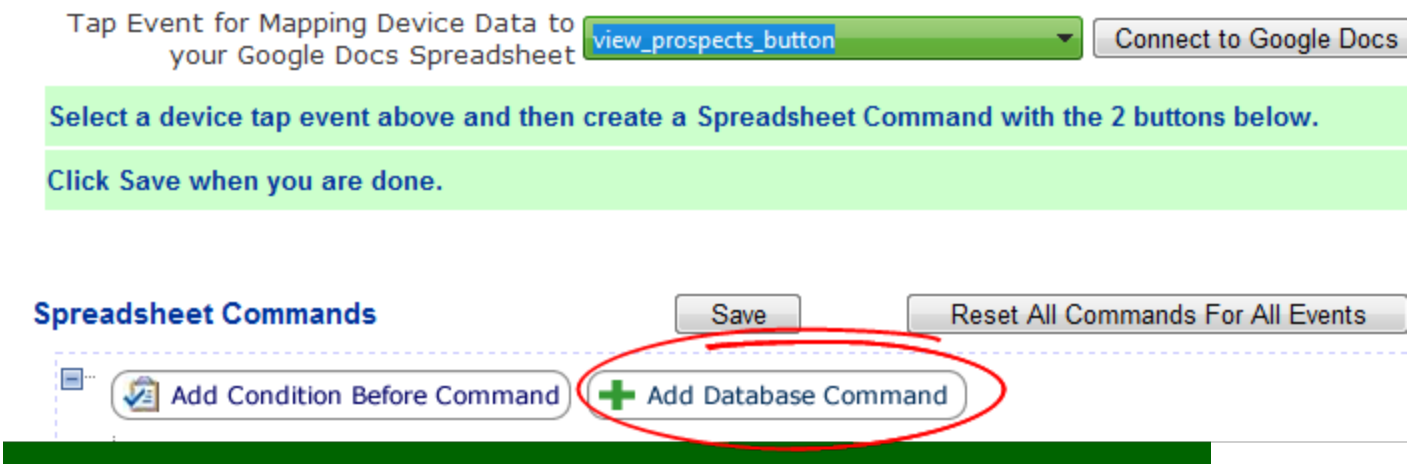
The last thing we need to know is the name of the tap event, the button or control that, when tapped, runs the data management operation. The name of the tap event for this example is view_prospects_button. We can determine this by hovering over the View Existing button, as shown below.

With this information, we're ready to go.

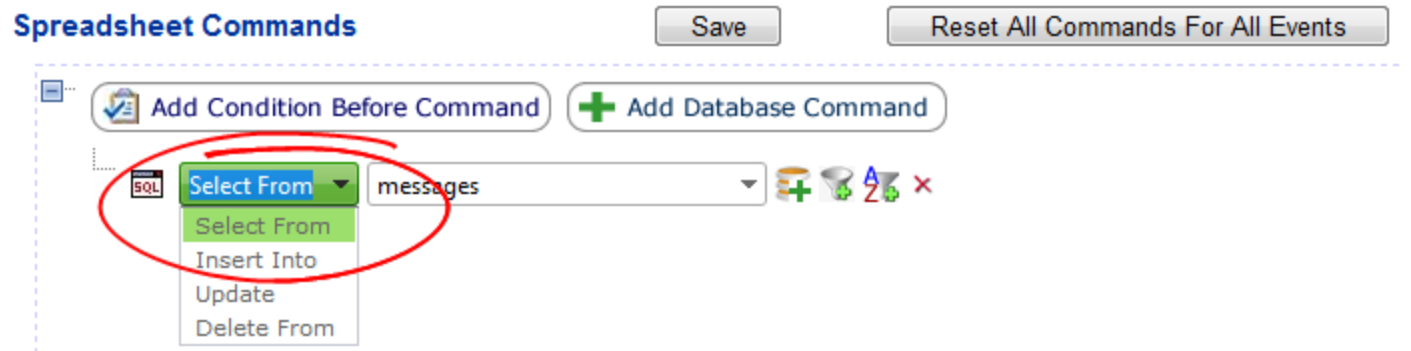
1. Go to the Manage Data page in ViziApps and select *view_prospects_button* from the Tap Event... field.



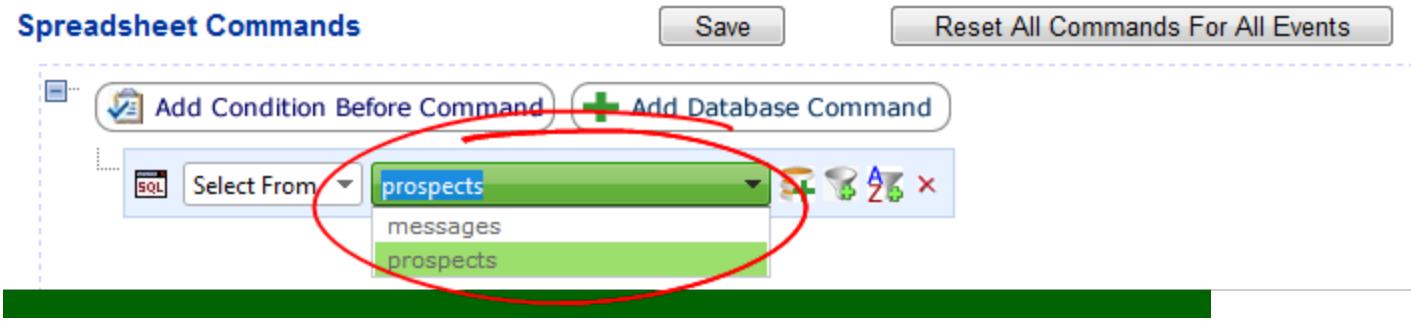
2. Click the Add Database Command button.



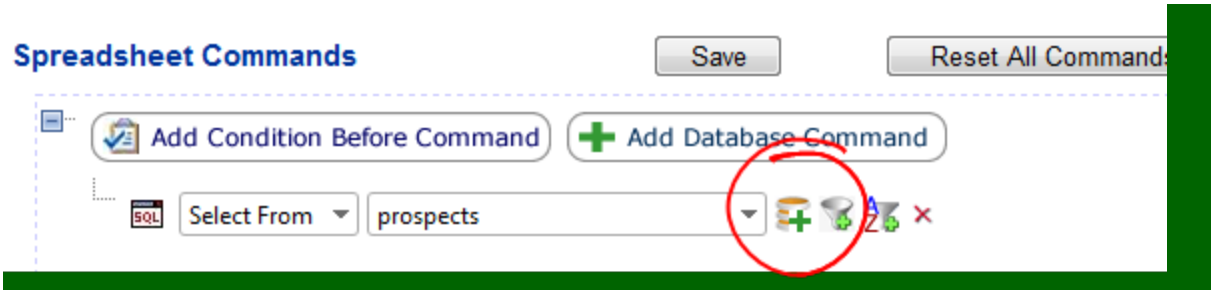
3. Select the command, in this case Select From, from the command pulldown.



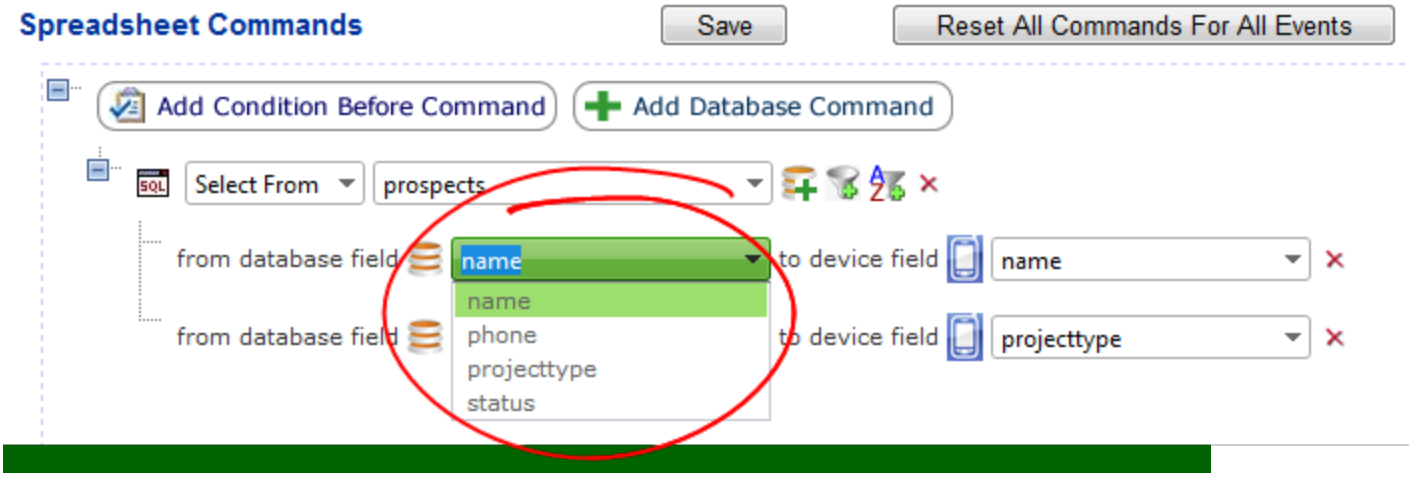
4. Select the worksheet from the worksheet pulldown.



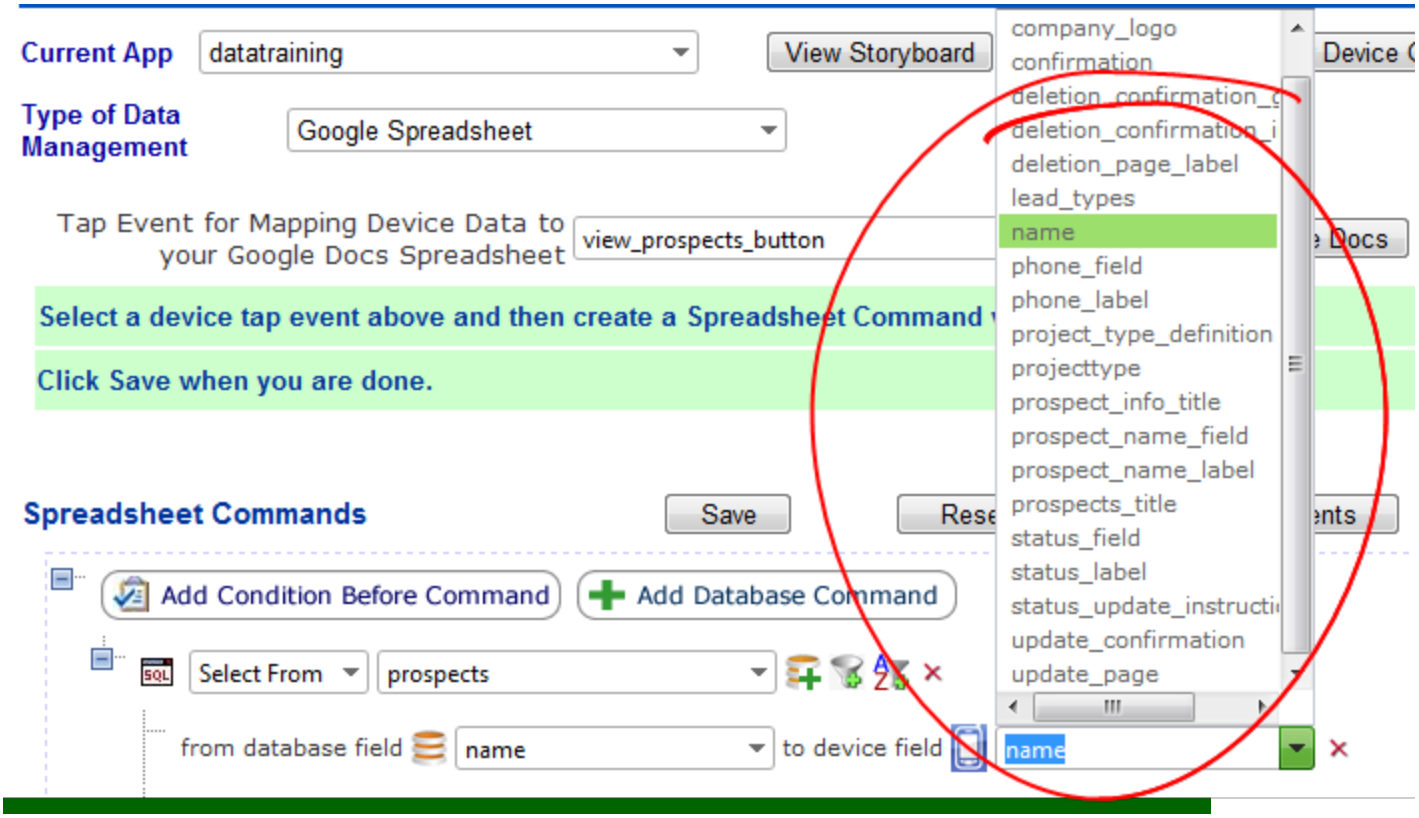
5. Click the Add a Field icon.



6. Click the pulldown for the spreadsheet field and select the desired field name, *name* in the example below.



7. Click the pulldown for the device field and select the desired field name, *name* in the example below.



That's the first entry.

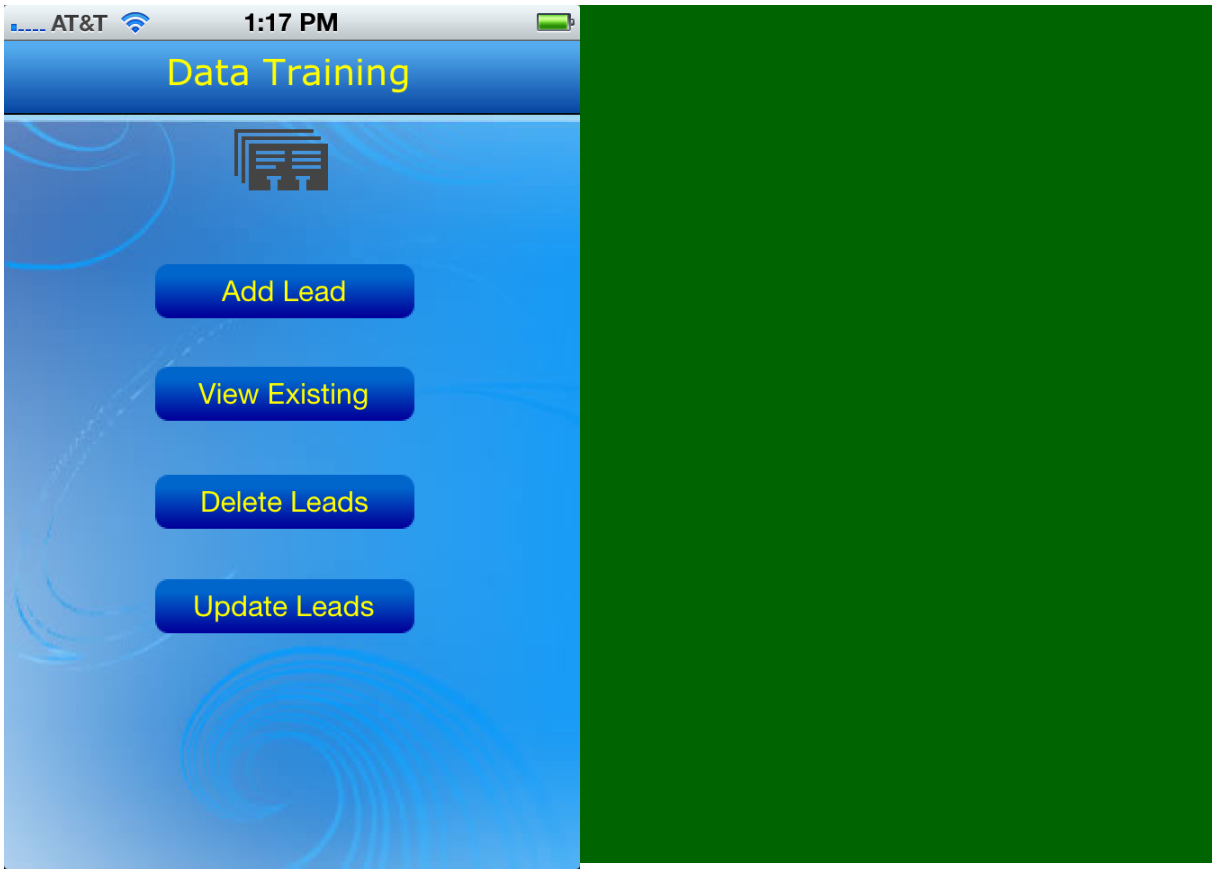
8. There's one more field on that app screen, so we need to repeat the process. Here's the result.

The app maps the selected spreadsheet fields to those two fields in the table on the app page when the user taps the View Existing button on the app page.

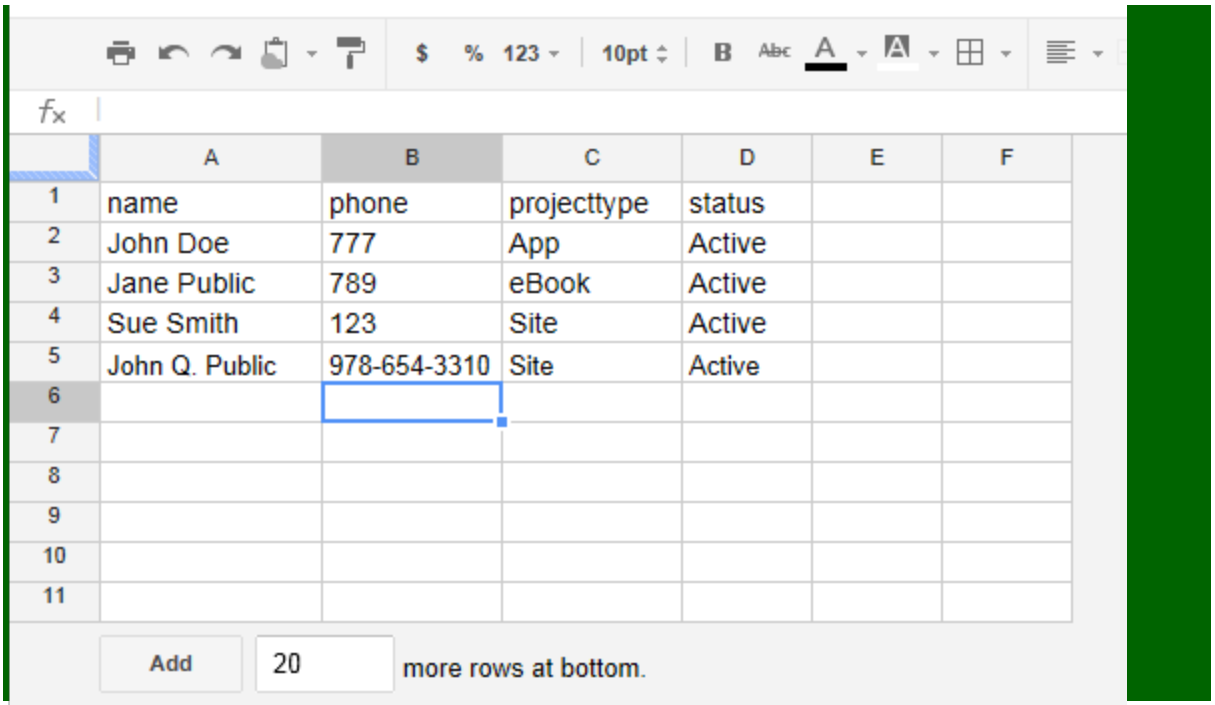
Select From - Concepts

The Select From command takes data from fields on the spreadsheet and inserts it in fields on the mobile device.

For example, tapping the View Existing button on the page below runs a set of Select From commands that access the spreadsheet, collect each prospect's name and area of interest, and insert that data in fields in a table in the app.



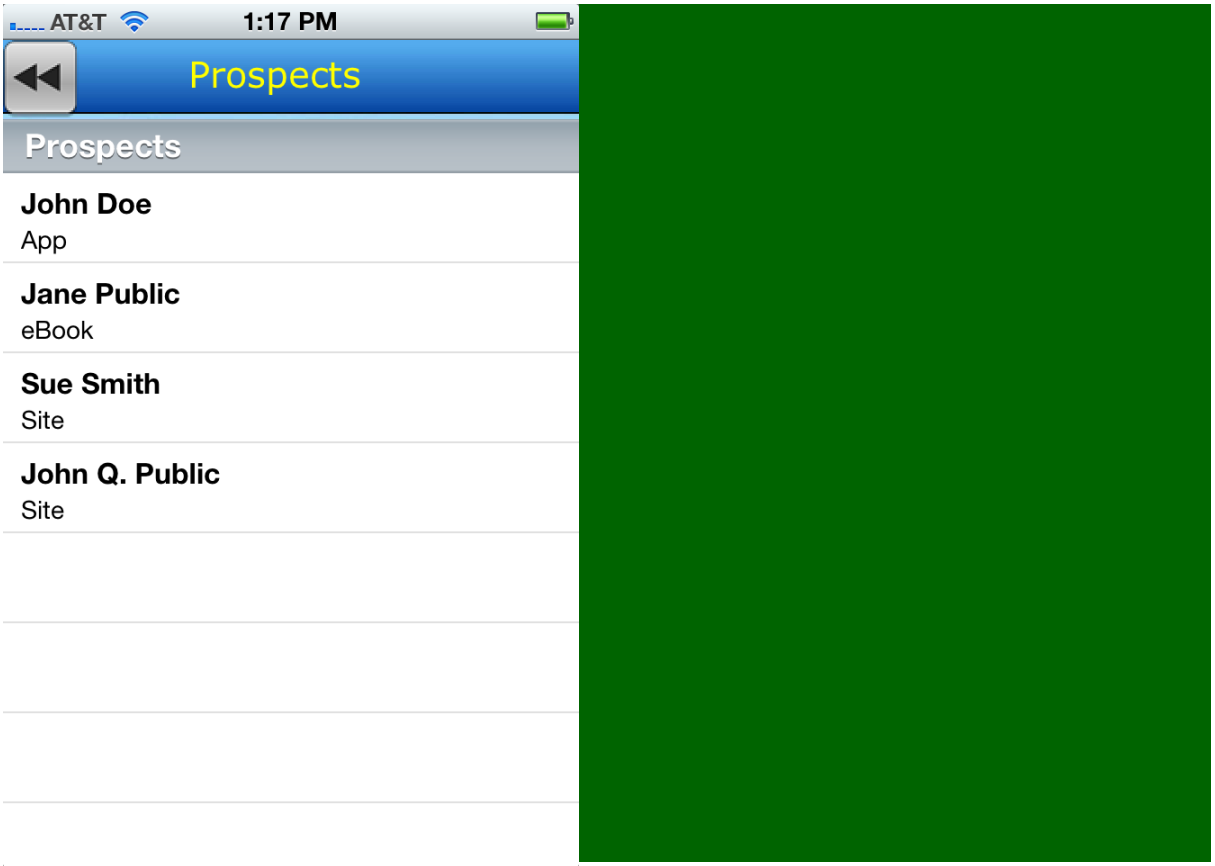
The screen below shows the spreadsheet and the data.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	978-654-3310	Site	Active		
6						
7						
8						
9						
10						
11						

Add 20 more rows at bottom.

The screen below shows the app page containing the table with the results.

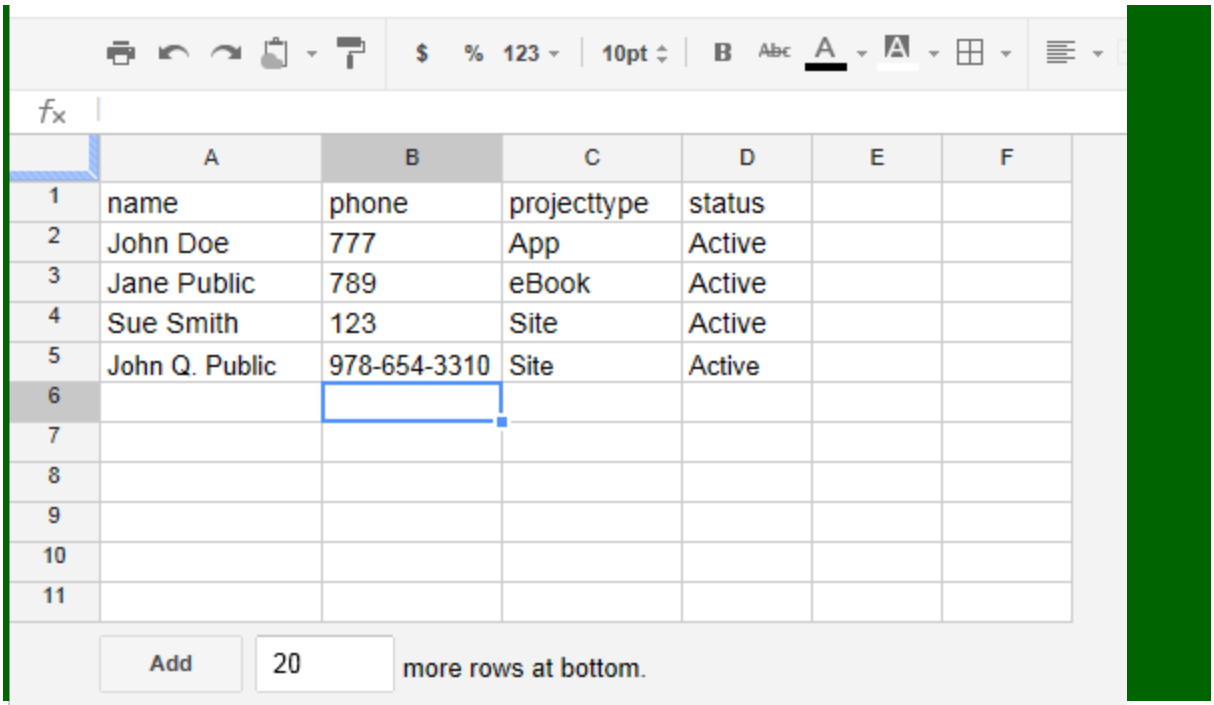


For an example of the mechanics of the Select From command - See "Select From - Examples" on page 12.

Select From - Examples

How did the example in the Select From - Concepts topic work? See "Select From - Concepts" on page 10

The image below shows the spreadsheet for this project.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	978-654-3310	Site	Active		
6						
7						
8						
9						
10						
11						

Add 20 more rows at bottom.

Notice the field (column) names - *name*, *phone*, *projecttype*, and *status*. (We'll ignore the *status* field in this example.) This spreadsheet also has two worksheets - *prospects* and *messages*. We'll use *prospects* in this example.

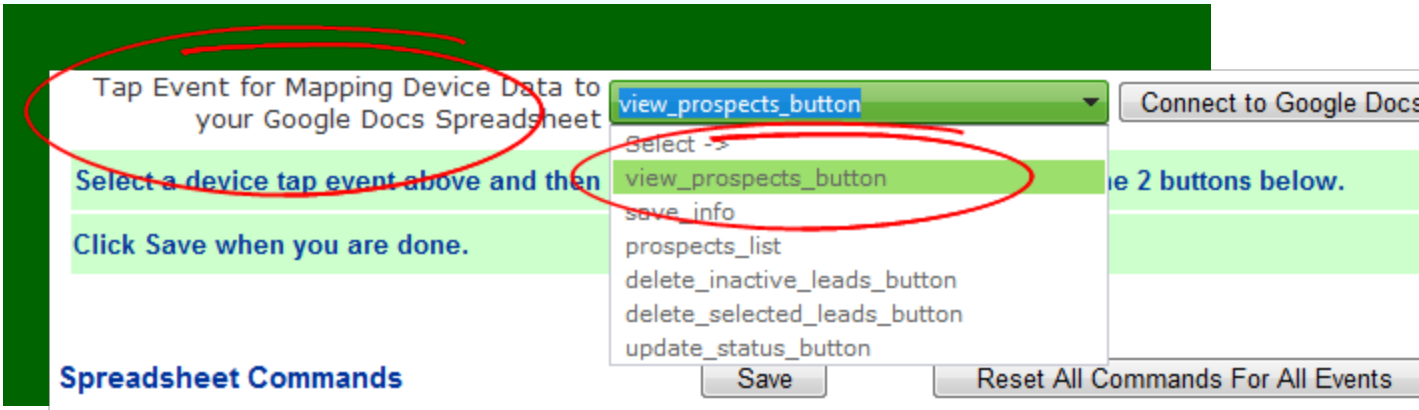
An app should contain fields that are to contain the data extracted from the spreadsheet and that correspond to the field names in the spreadsheet. In this example, we want to extract the users' names and project type of interest and display that information in the app. (We're also doing this for all users listed in the spreadsheet, rather than trying to display a specific user's information.) So there should be fields in the app that correspond to the name and project type fields and that will contain and display the data extracted from the spreadsheet. They're called *name* and *projecttype*.

Tip: Give fields names that make it clear which fields in the app connect to which fields in the spreadsheet. For example, it might be better to name these two table fields *name_column* and *projecttype_column*. (The actual names aren't crucial as long as they're clear.)

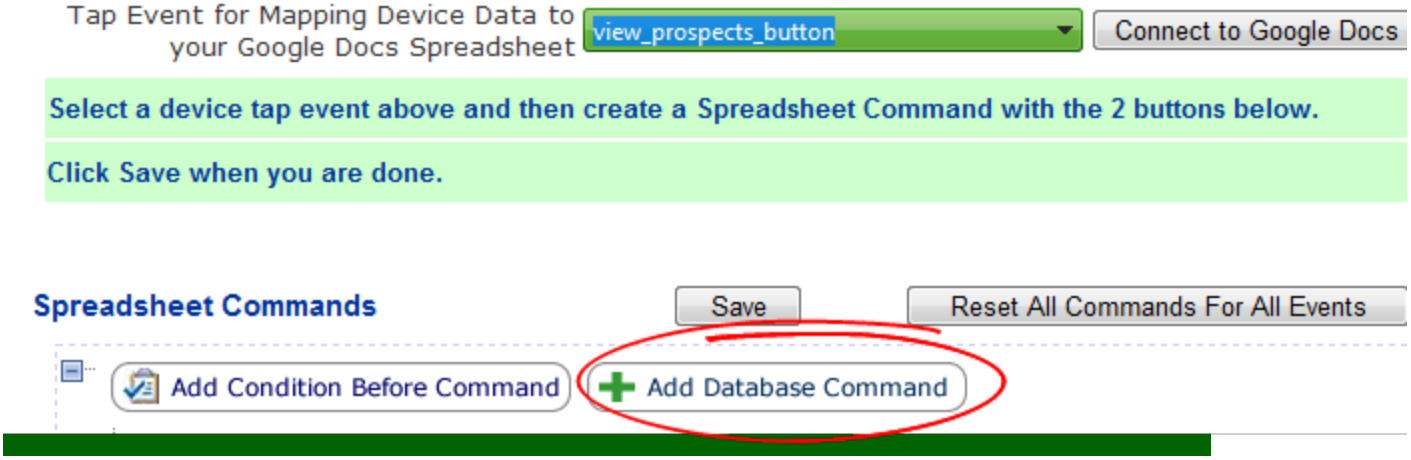
The last thing we need to know is the name of the tap event, the button or control that, when tapped, runs the data management operation. The name of the tap event for this example is view_prospects_button. We can determine this by hovering over the View Existing button, as shown below.

With this information, we're ready to go.

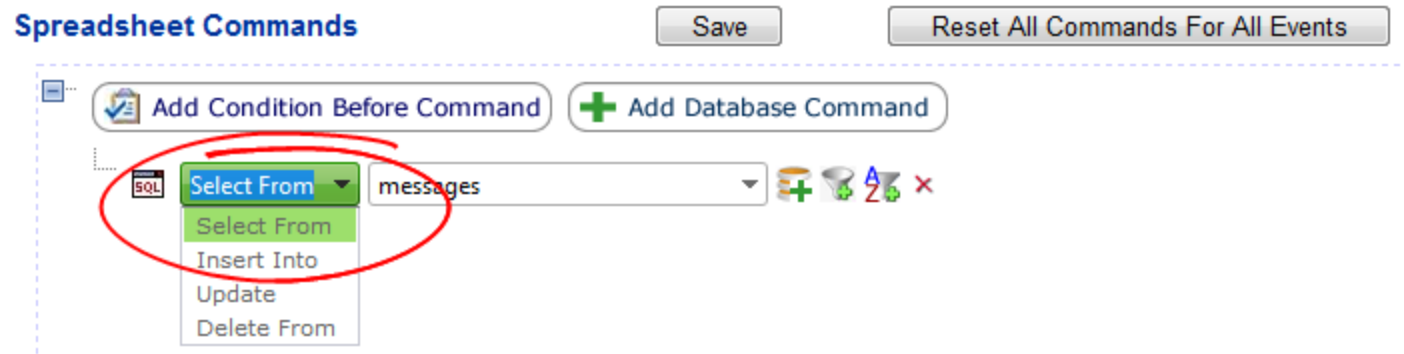
1. Go to the Manage Data page in ViziApps and select *view_prospects_button* from the Tap Event... field.



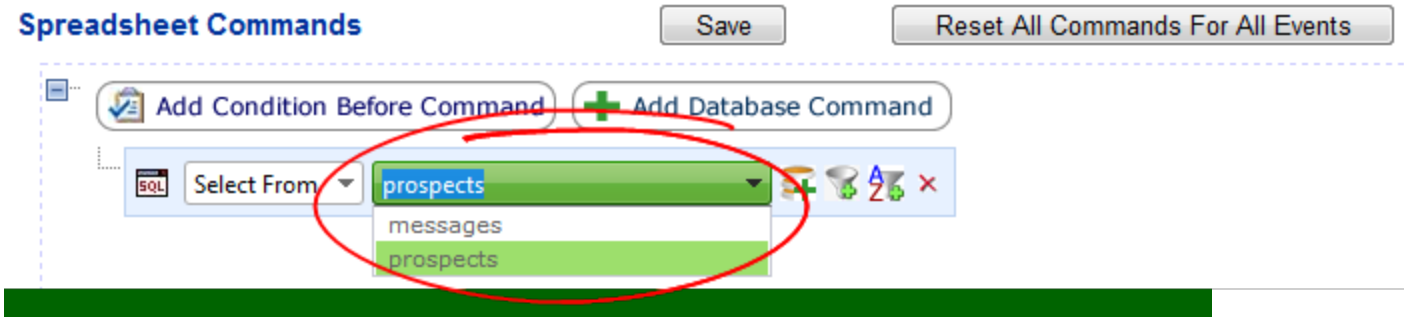
2. Click the Add Database Command button.



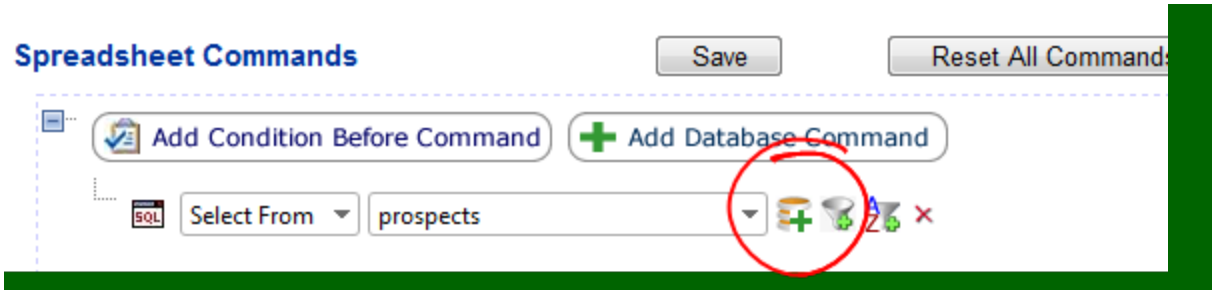
3. Select the command, in this case Select From, from the command pulldown.



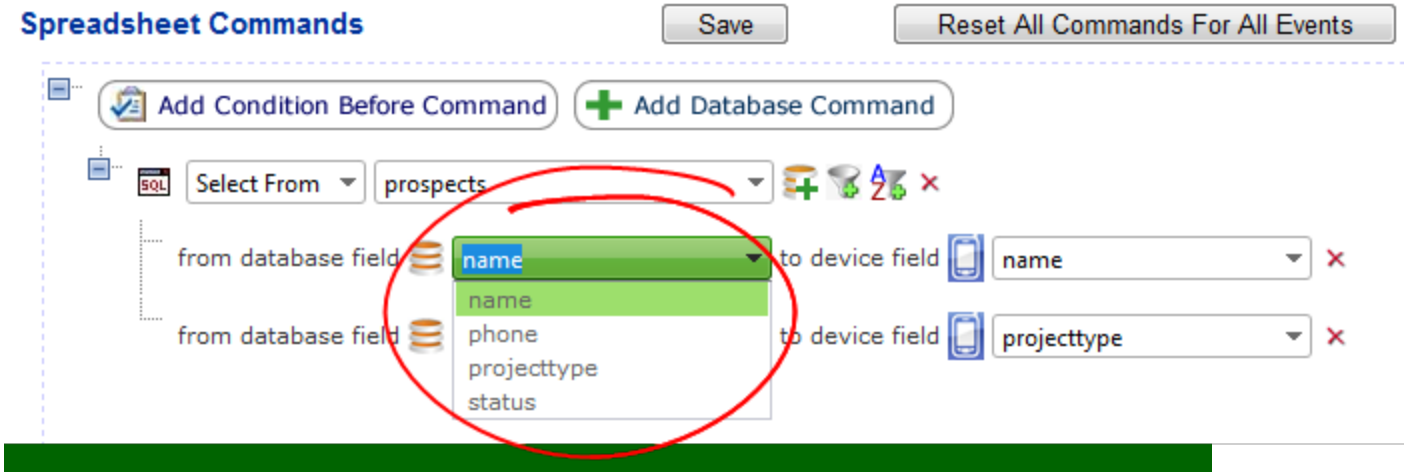
4. Select the worksheet from the worksheet pulldown.



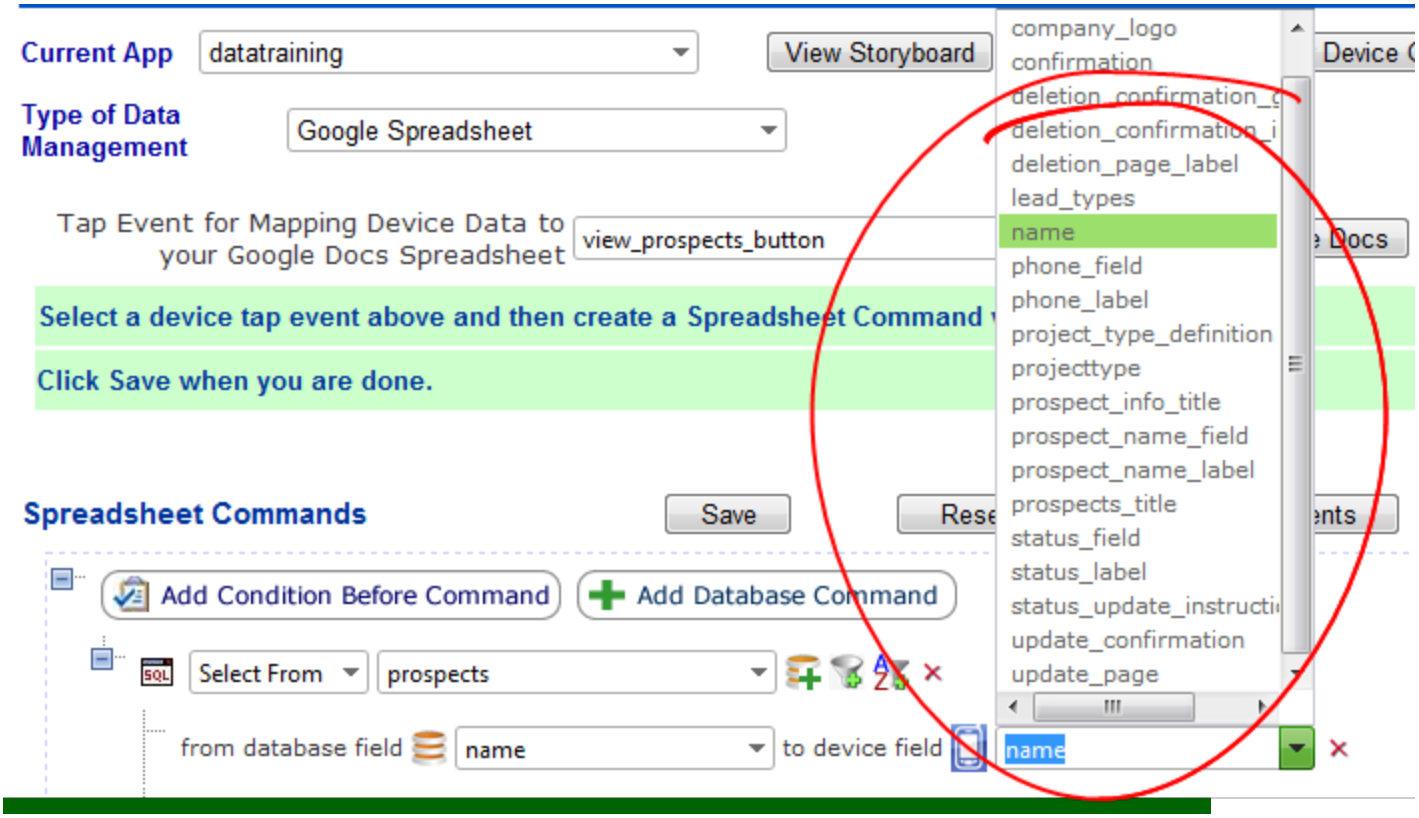
5. Click the Add a Field icon.



6. Click the pulldown for the spreadsheet field and select the desired field name, *name* in the example below.

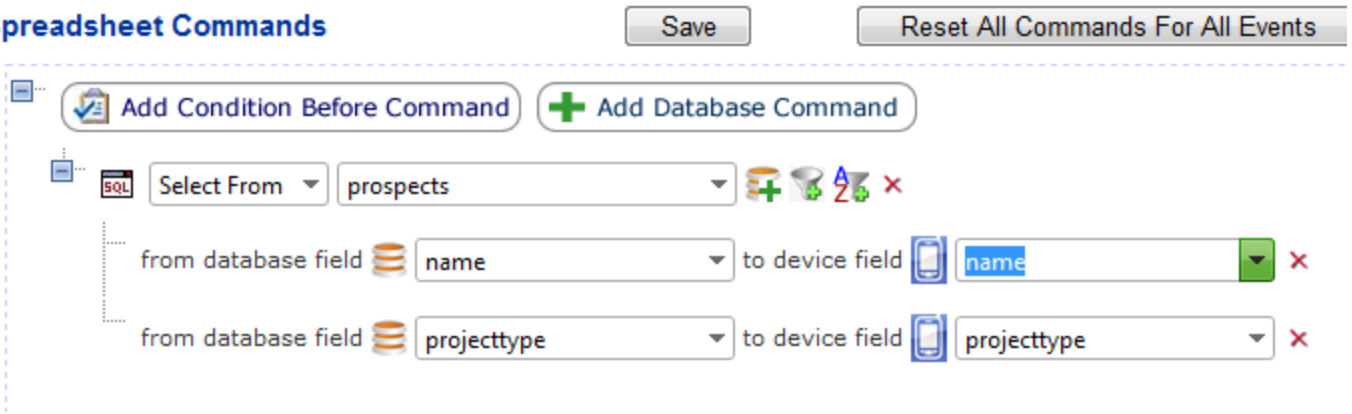


7. Click the pulldown for the device field and select the desired field name, *name* in the example below.



That's the first entry.

8. There's one more field on that app screen, so we need to repeat the process. Here's the result.



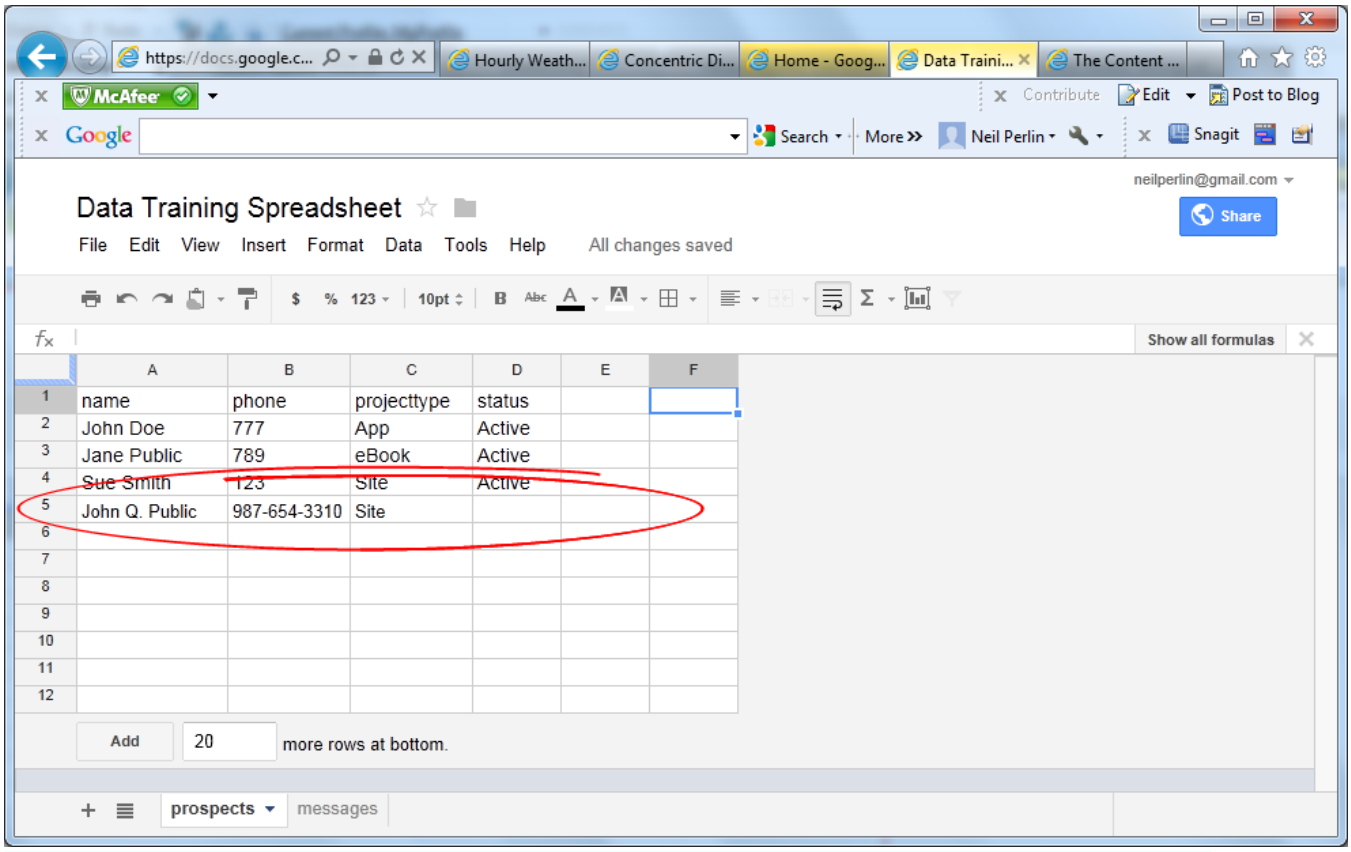
The app maps the selected spreadsheet fields to those two fields in the table on the app page when the user taps the View Existing button on the app page.

Insert Into - Concepts

The Insert Into command takes data entered in fields on the mobile device and inserts it in fields on the appropriate row of a spreadsheet.

For example, tapping the Save button on the page below runs a set of Insert Into commands that insert a prospect's name, phone number, and area of interest into the spreadsheet.

The screen below shows the spreadsheet with the added information, in row 5.



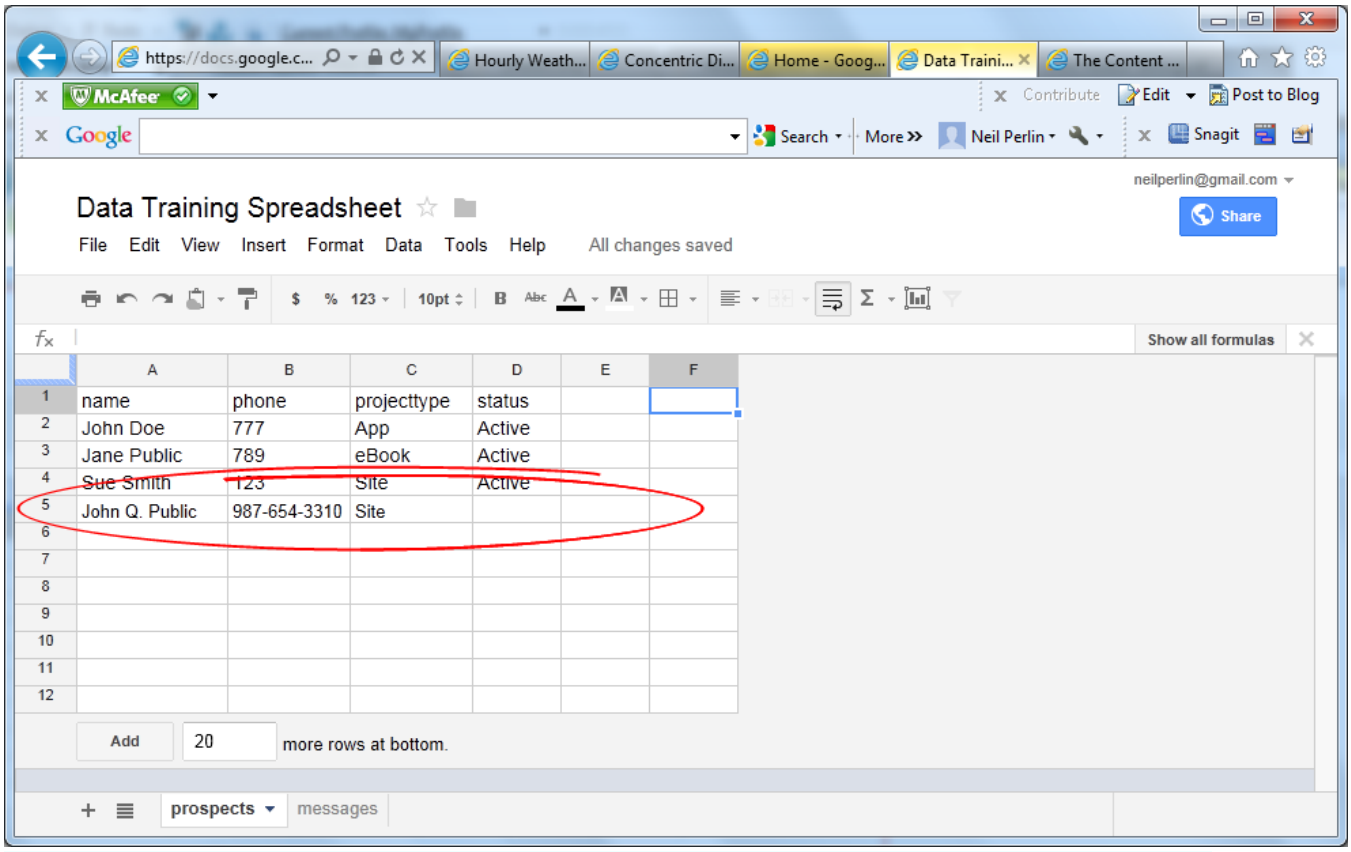
For an example of the mechanics of the Insert Into command - See "Insert Into - Examples" on page 19.

Insert Into - Concepts

The Insert Into command takes data entered in fields on the mobile device and inserts it in fields on the appropriate row of a spreadsheet.

For example, tapping the Save button on the page below runs a set of Insert Into commands that insert a prospect's name, phone number, and area of interest into the spreadsheet.

The screen below shows the spreadsheet with the added information, in row 5.

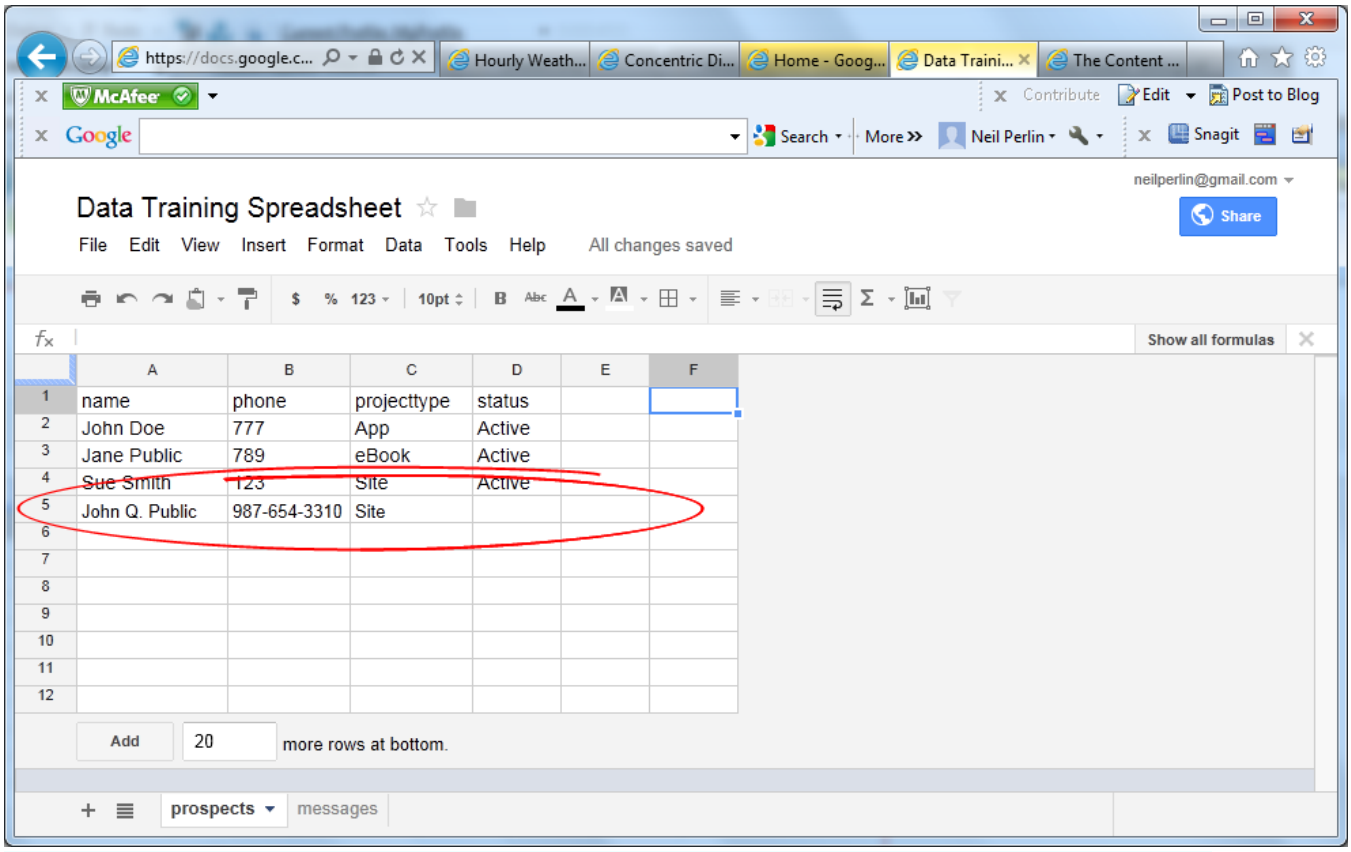


For an example of the mechanics of the Insert Into command - See "Insert Into - Examples" on page 19.

Insert Into - Examples

How did the example in the Insert Into - Concepts topic work? See "Insert Into - Concepts" on page 17

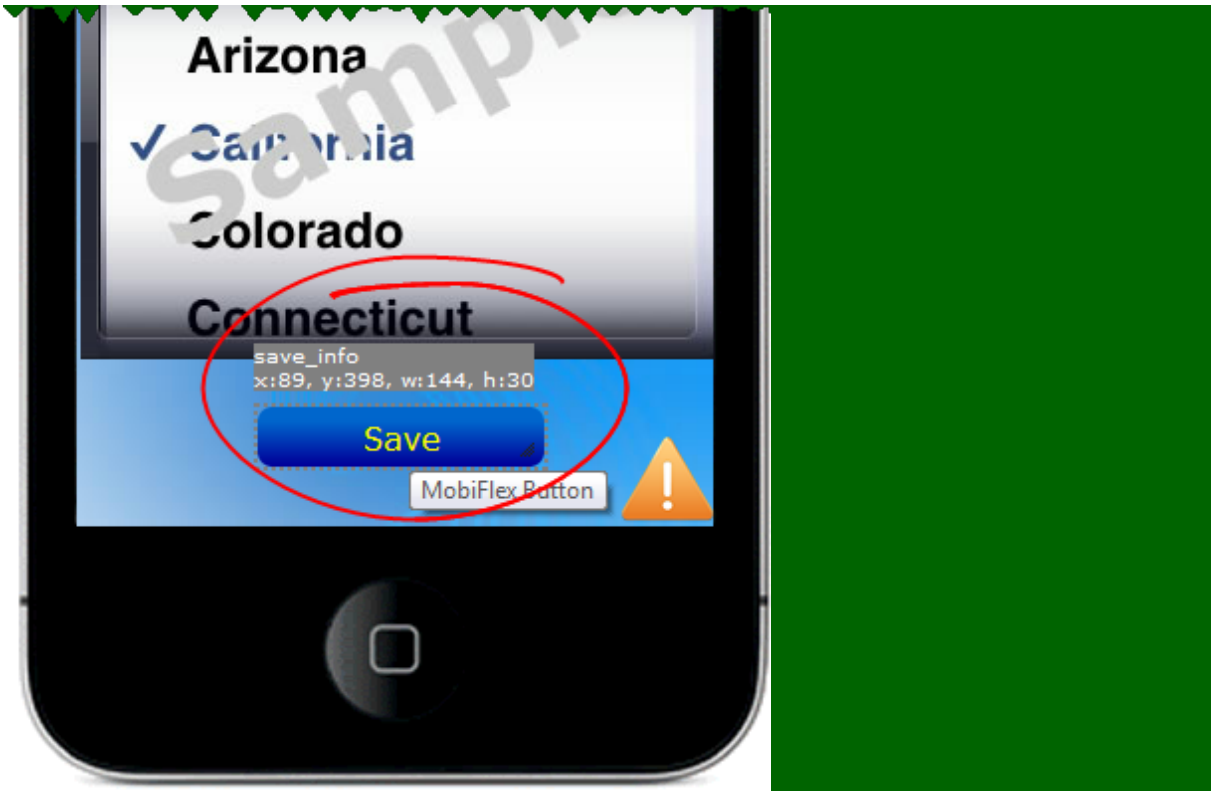
The image below shows the spreadsheet for this project.



Notice the field (column) names - *name*, *phone*, *projecttype*, and *status*. (We'll ignore the *status* field in this example.) Notice also that this spreadsheet has two worksheets - prospects and messages. We'll use prospects in this example.

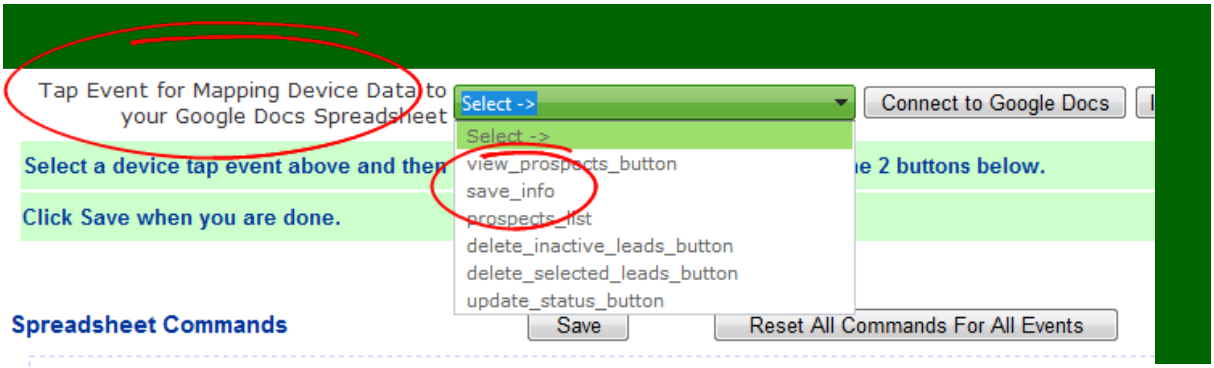
An app should contain fields that correspond to the field names in the spreadsheet. For example, there should be a field in the app that corresponds to the *name* field. There is - it's called *prospect_name_field*, and two others called *phone_field* and *project_type_definition*. (The actual names aren't crucial as long as they're clear.)

The last thing we need to know is the name of the tap event, the button or control that, when tapped, runs the data management operation. The name of the tap event for this example is *save_info*, which we can determine by hovering over the Save button, as shown below.

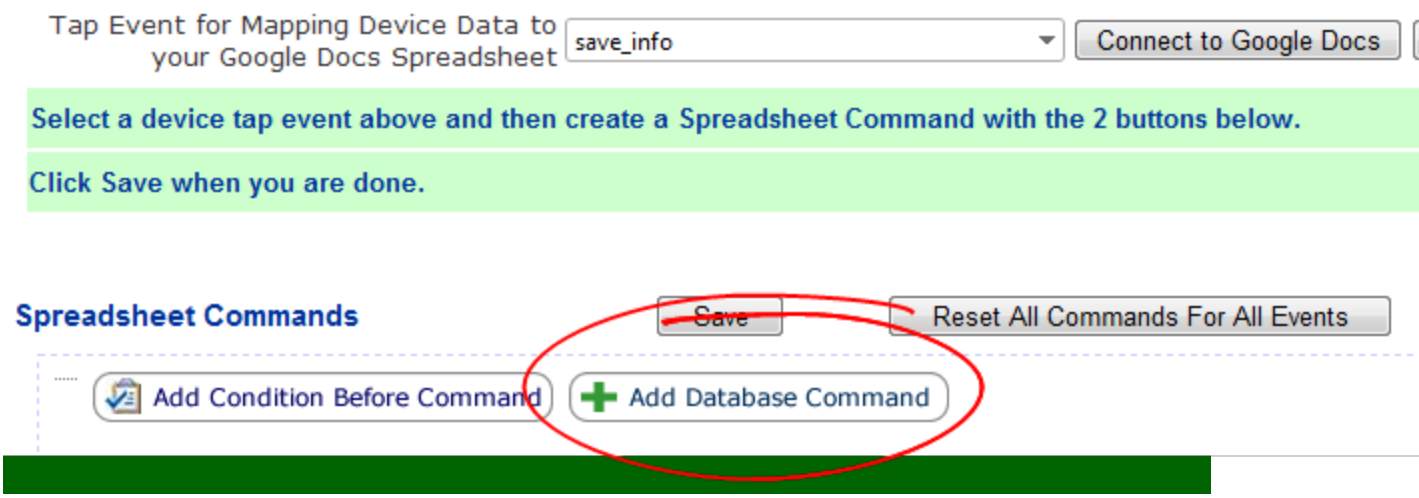


With this information, we're ready to go.

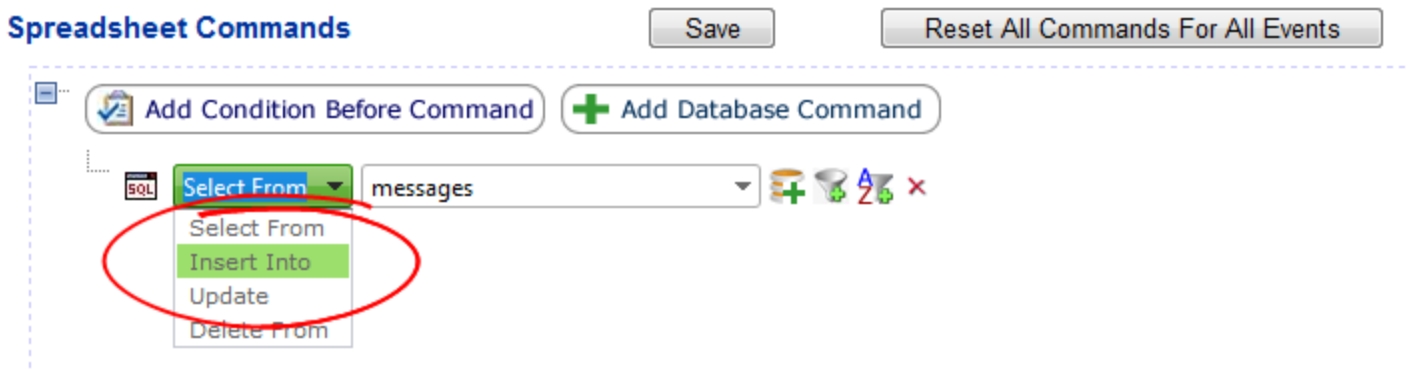
1. Go to the Manage Data page in ViziApps and select *save_info* from the Tap Event... field.



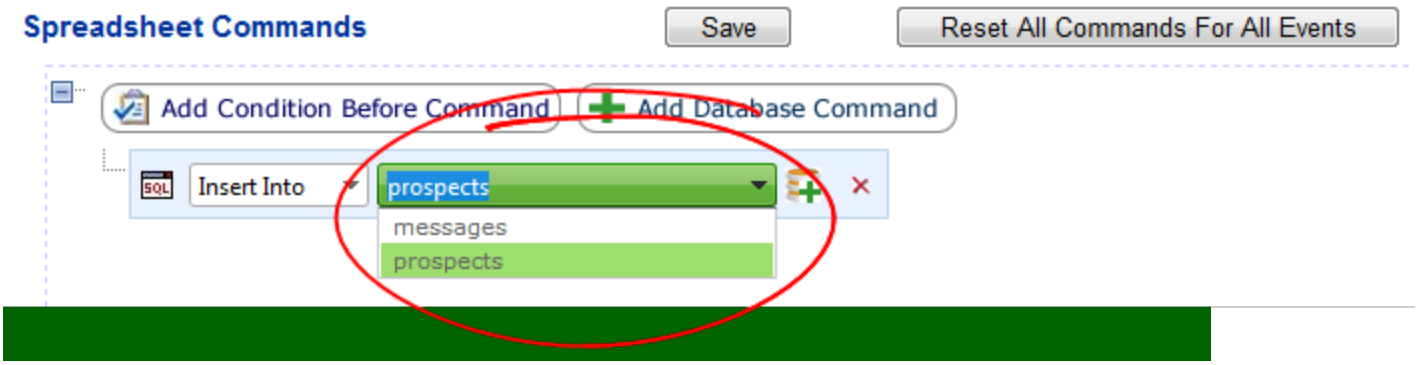
2. Click the Add Database Command button.



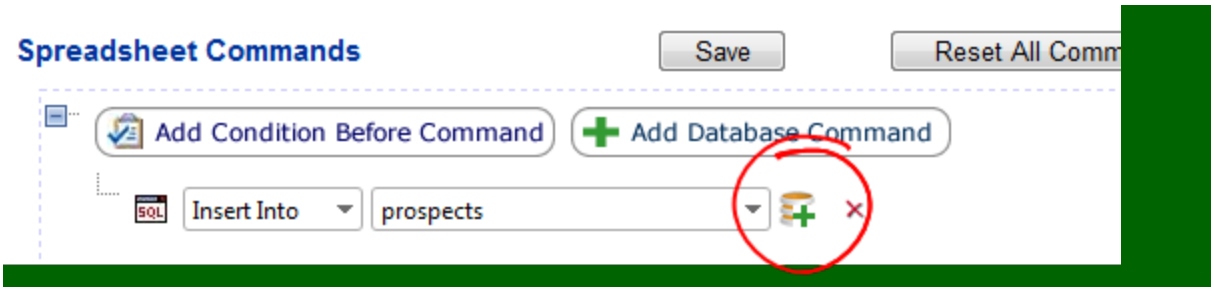
3. Select the command, in this case Insert Into, from the command pulldown.



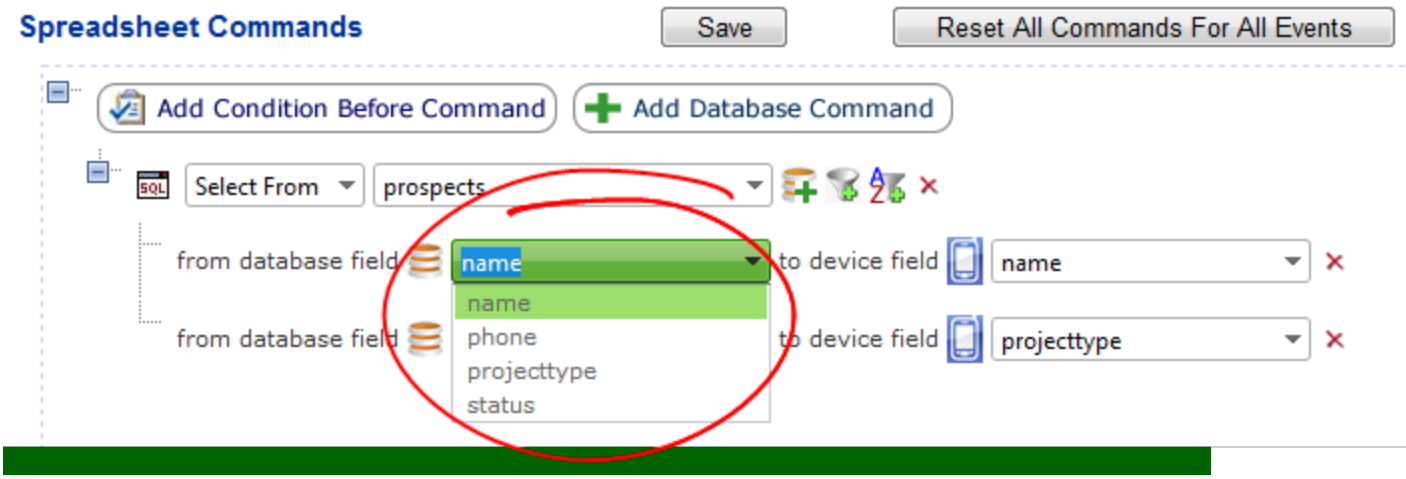
4. Select the worksheet from the worksheet pulldown.



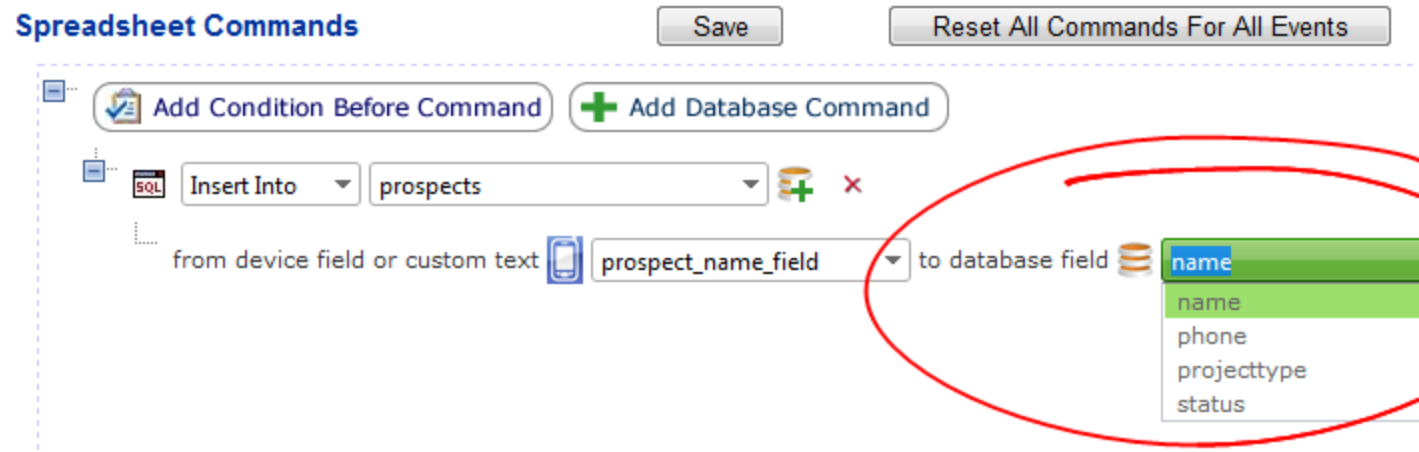
5. Click the Add a Field icon.



6. Click the pulldown for the device field and select the desired field name, *name* in the example below.

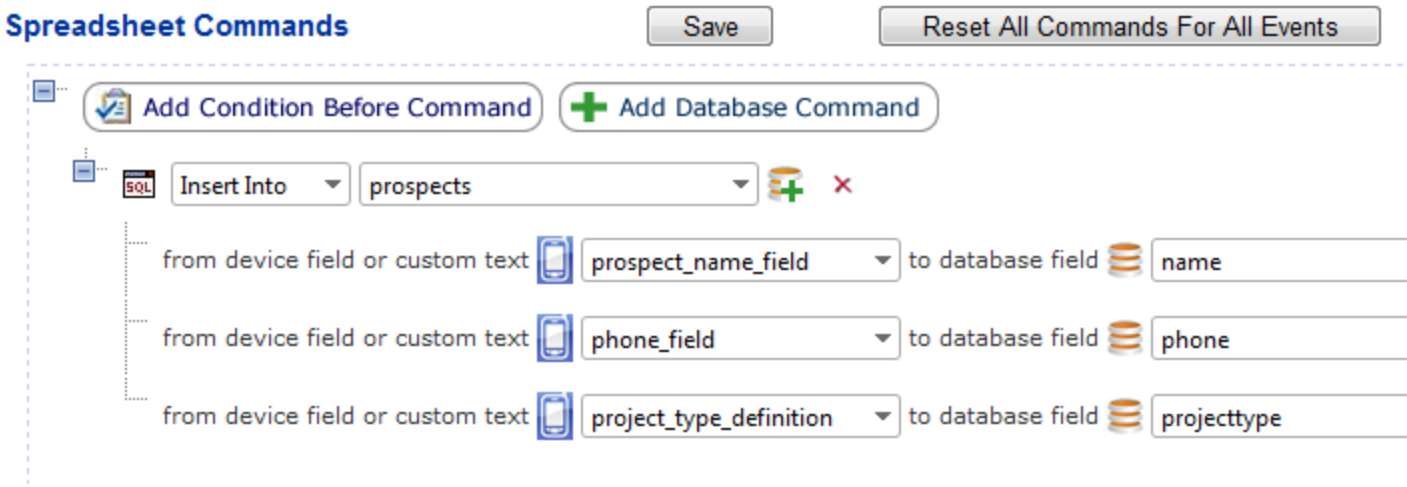


7. Click the pulldown for the database field and select the desired field name, *name* in the example below.



That's the first entry.

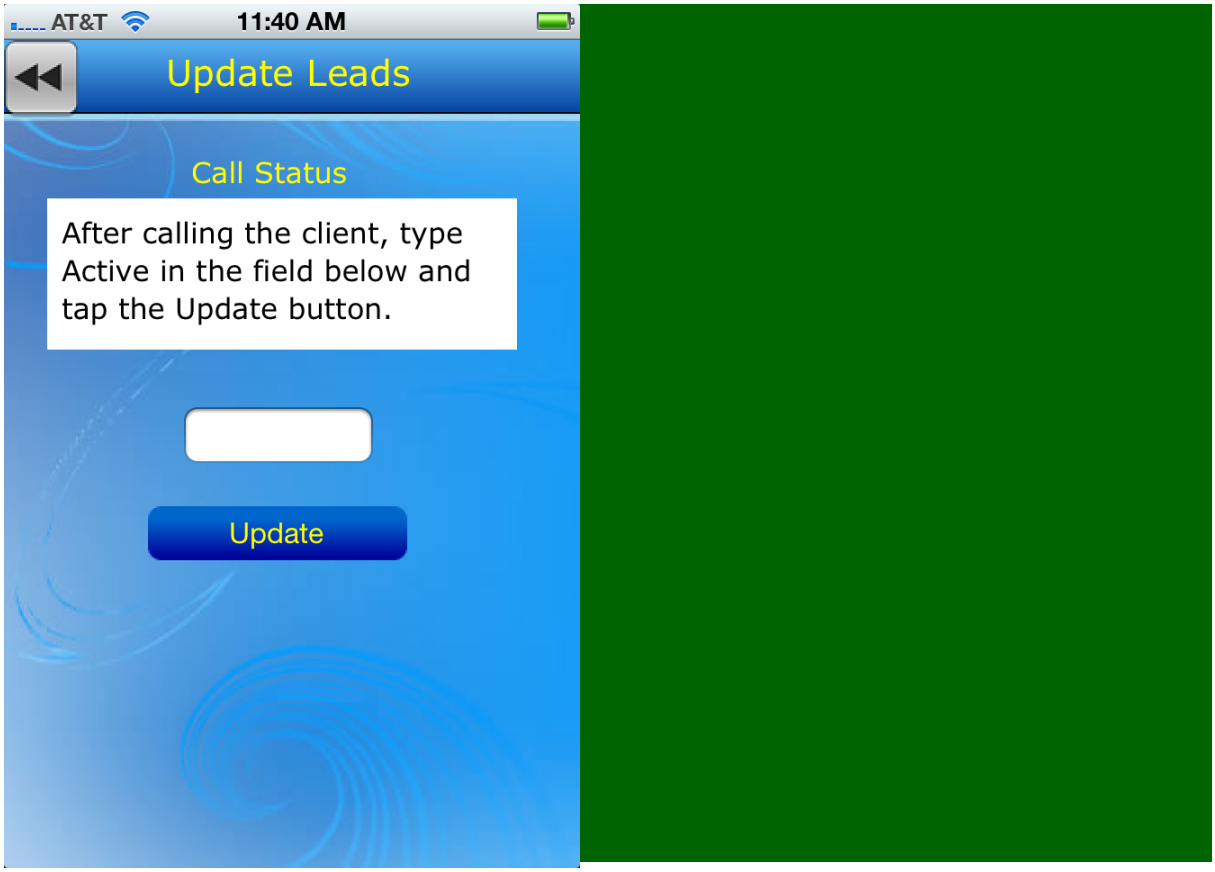
8. There are two more fields on that app screen, so we need to do this twice more. Here's the result.



The app maps all the fields on that app page to the spreadsheet when the user taps the Save Info button on the app page.

Update - Examples

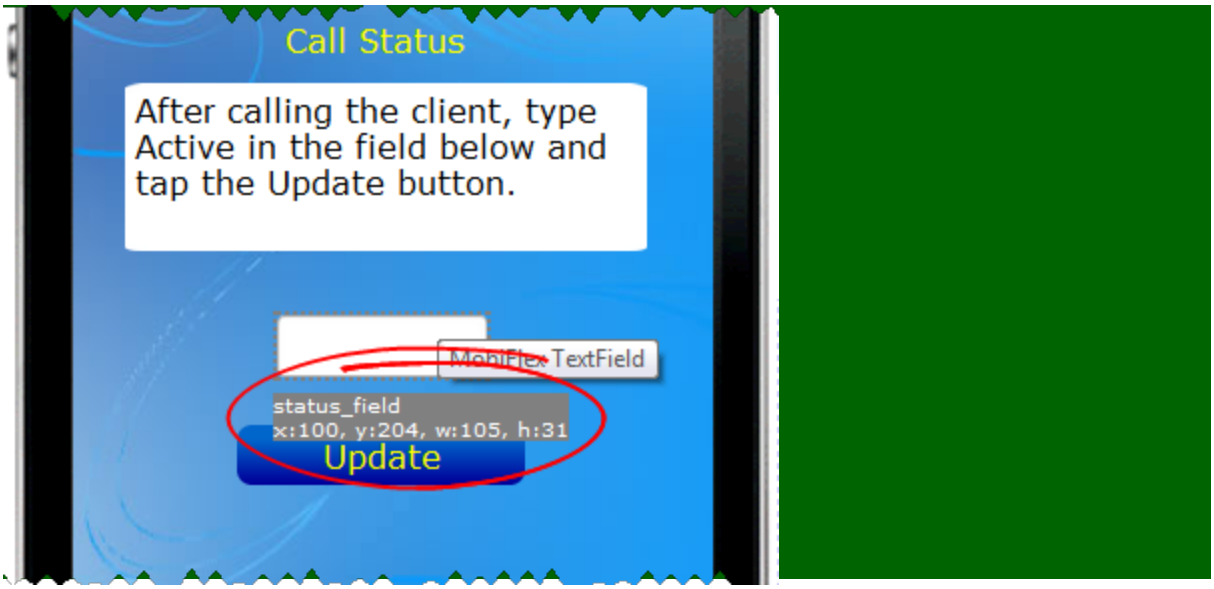
To update a field, a user enters the update value (the value that will replace the old value) in a field on the mobile device, then specifies the spreadsheet field whose value should be replaced by the new value. For example, in the image below, typing the word Active (or any word) in the entry field and tapping the Update button replaces the contents of all specified fields, in this case the *status* field, with the new status Active.



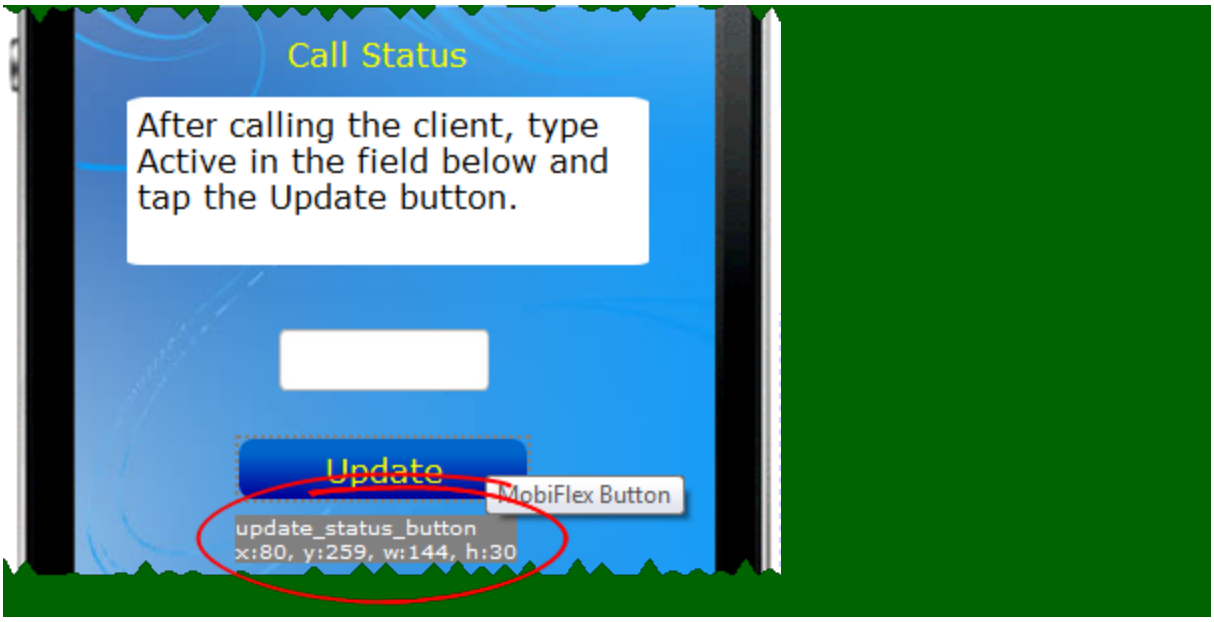
Here's a copy of the spreadsheet.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

We need to know the name of the field in which users can enter the replacement value and the name of the tap event - the button. The name of the entry field is *status_field*, as shown below.

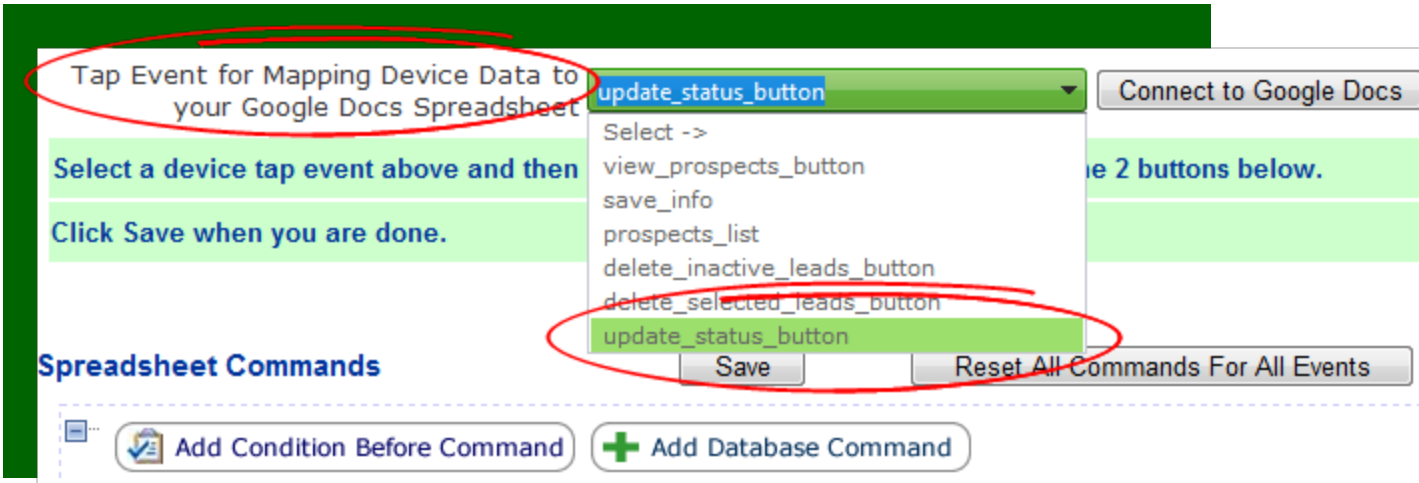


The name of the tap event is the picker name, here *delete_leads_picker*, which we can determine by hovering over the Update button, as shown below.

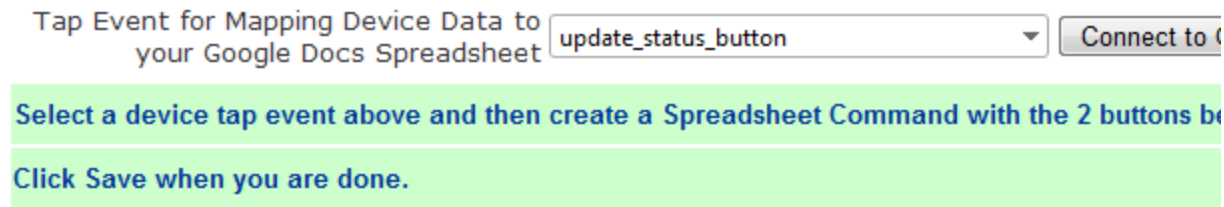


We now have enough to set up the code.

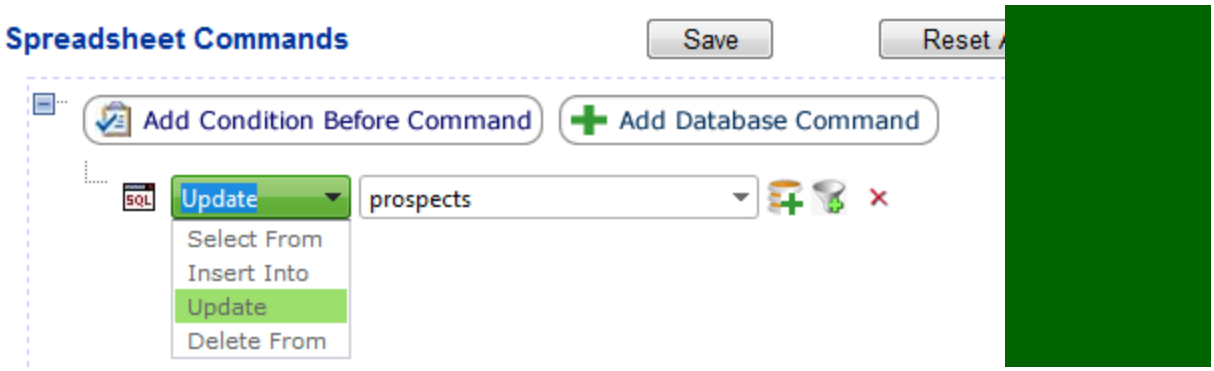
1. Go to the Manage Data page in ViziApps and select *update_status_button* from the Tap Event... field.



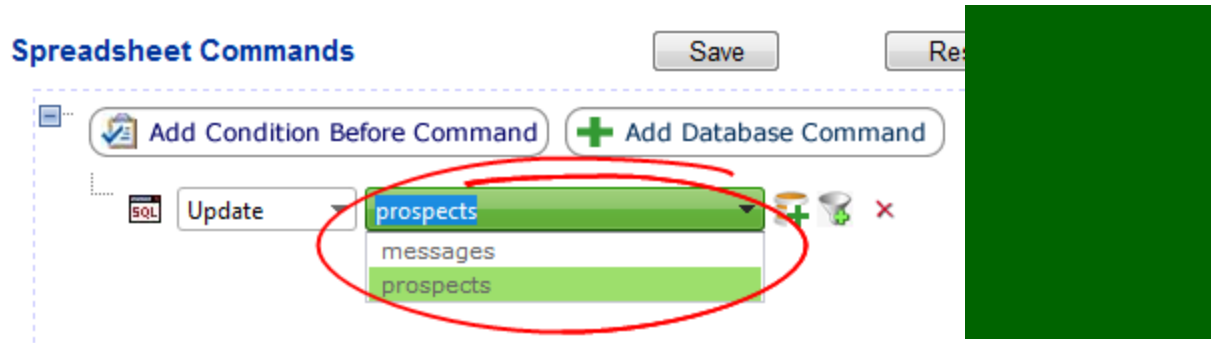
2. Click the Add Database Command button.



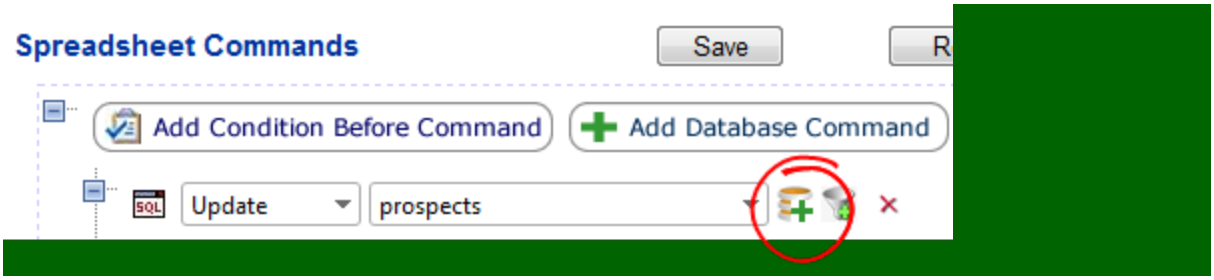
3. Select the Update command from the command pulldown.



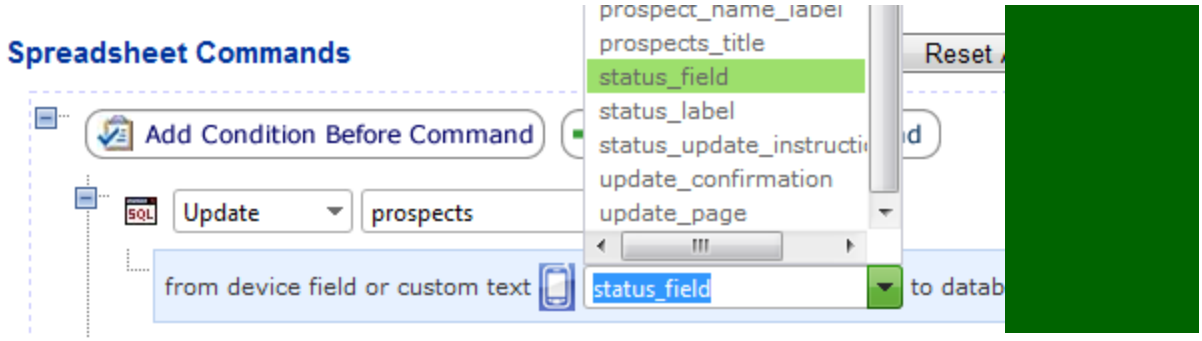
4. Select the worksheet from the worksheet pulldown.



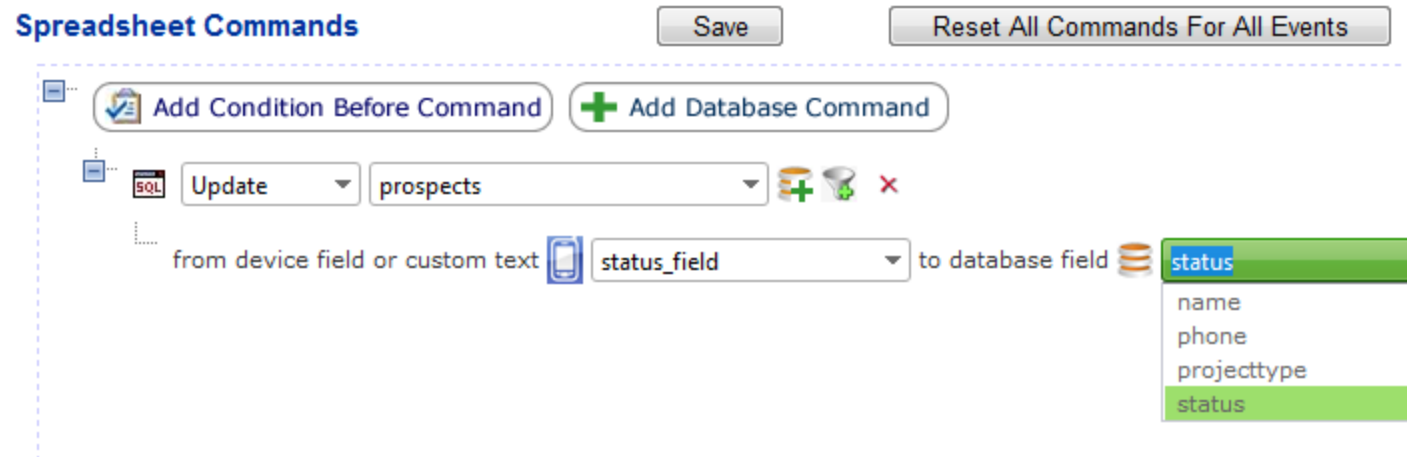
5. Click the Add a Field icon to specify the field containing the replacement value.



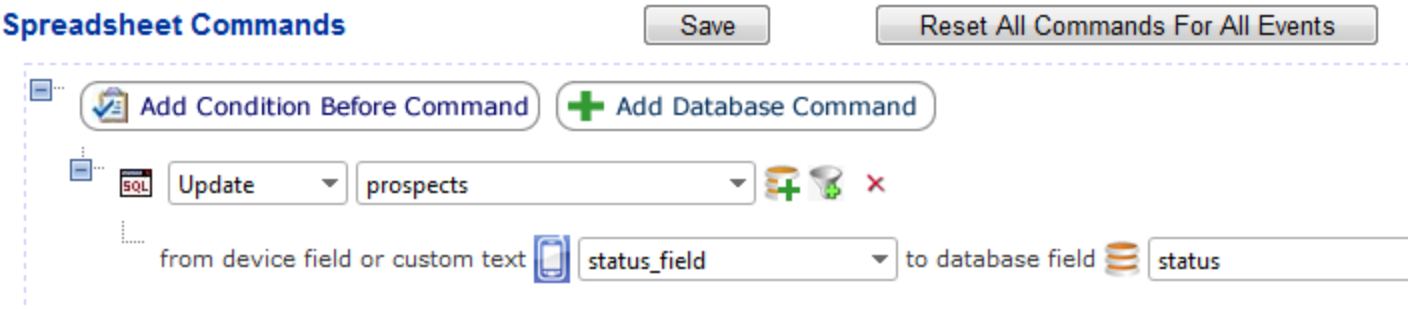
6. Click the pulldown for the device field and select the desired field name, *status_field* in the example below.



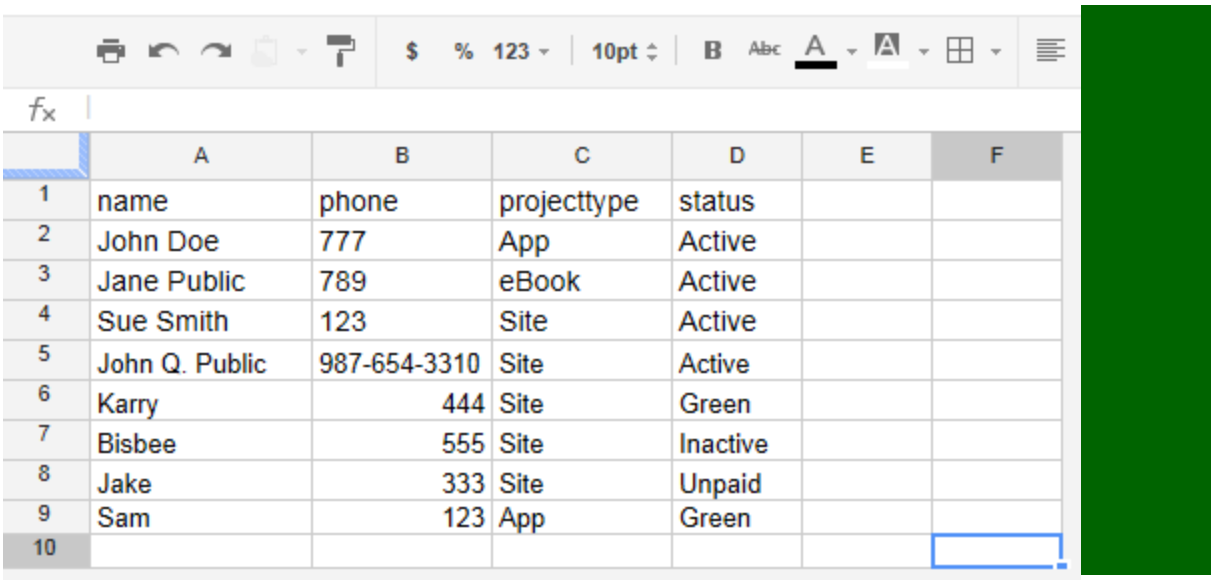
7. Click the pulldown for the database field whose values are to be replaced, in this case the *status* field as shown below.



The image below shows the completed entry.

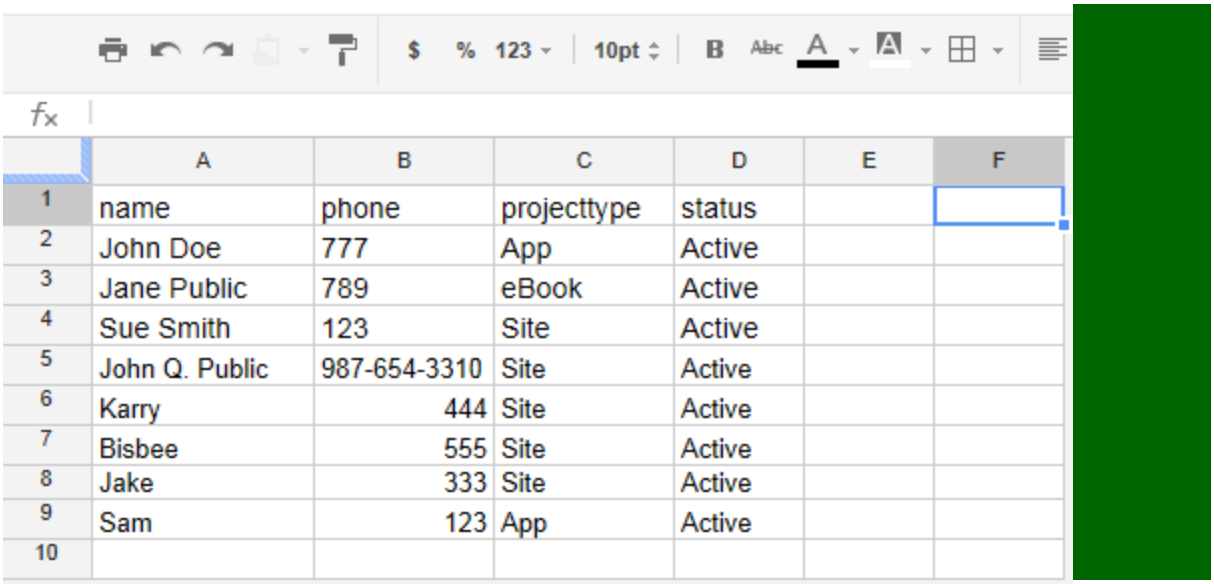


The result is the change in every status field from its current entry to Active, as shown in the image below. First the original spreadsheet.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

Then the changed spreadsheet.



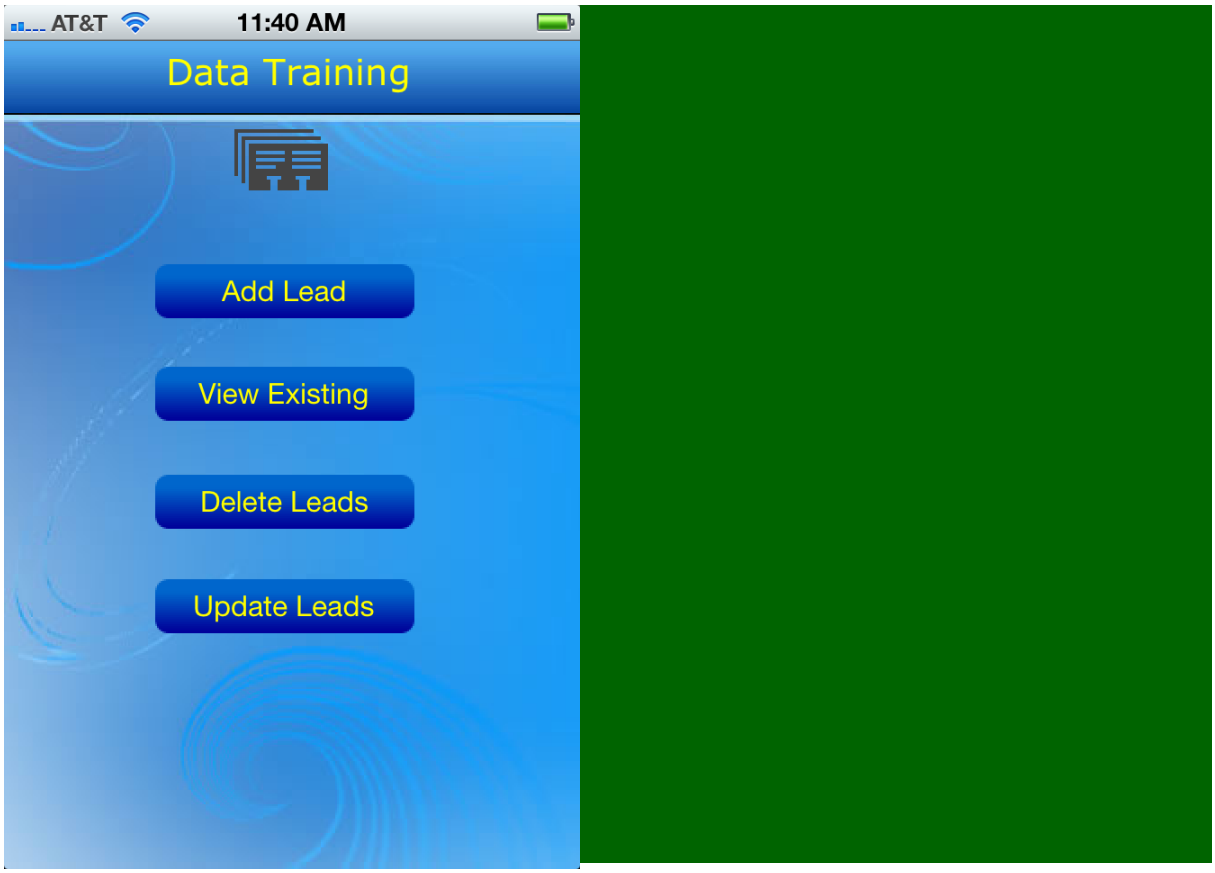
	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Active		
7	Bisbee	555	Site	Active		
8	Jake	333	Site	Active		
9	Sam	123	App	Active		
10						

Note that the Update command updates the contents of all fields with the same name, such as all status fields. But what if you only want to update the status of Unpaid entries, such as Jake in line 8, but not change Karry, Bisbee, or Sam on lines 6, 7, and 9. That requires using conditions, covered in another section of the help. See "Introduction to Conditions in ViziApps" on page 53

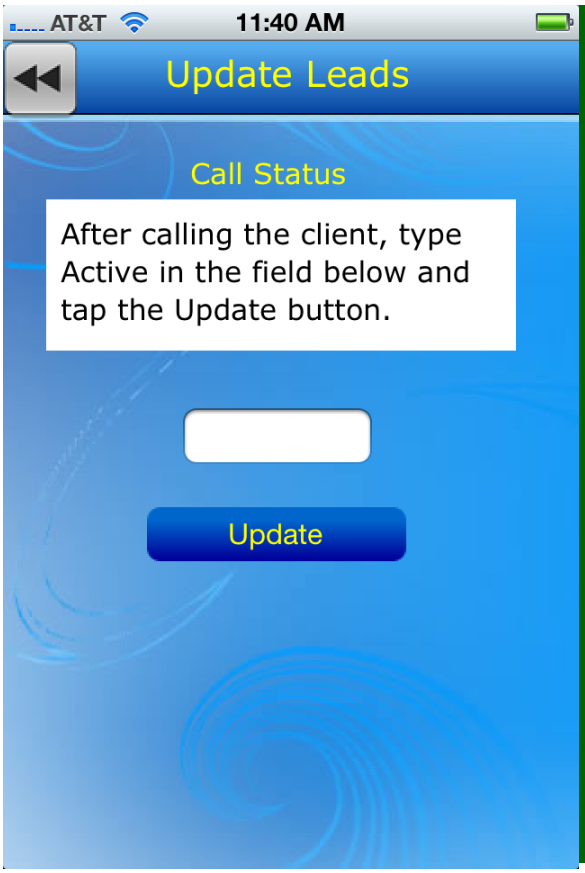
Update - Concepts

The Update command takes an entry made in a field on the mobile device and inserts it into associated fields on the spreadsheet.

For example, tapping the Update Leads button on the page below...



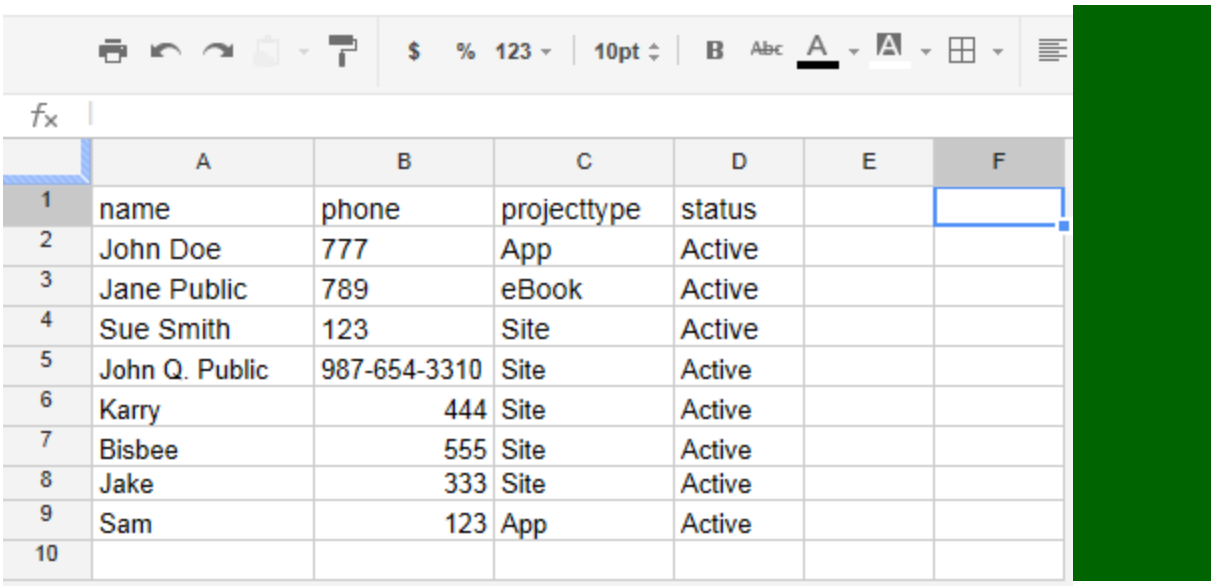
Opens the page below, where a reader can specify the entry to use to update a selected set of fields.



Typing the word Active in the field and tapping the Update button changes the status of all fields to Active. For example, the screen below shows the spreadsheet before the update.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

The screen below shows the same spreadsheet after the update. The status of the last four entries changed from Green, Inactive, or Unpaid, to Active.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Active		
7	Bisbee	555	Site	Active		
8	Jake	333	Site	Active		
9	Sam	123	App	Active		
10						

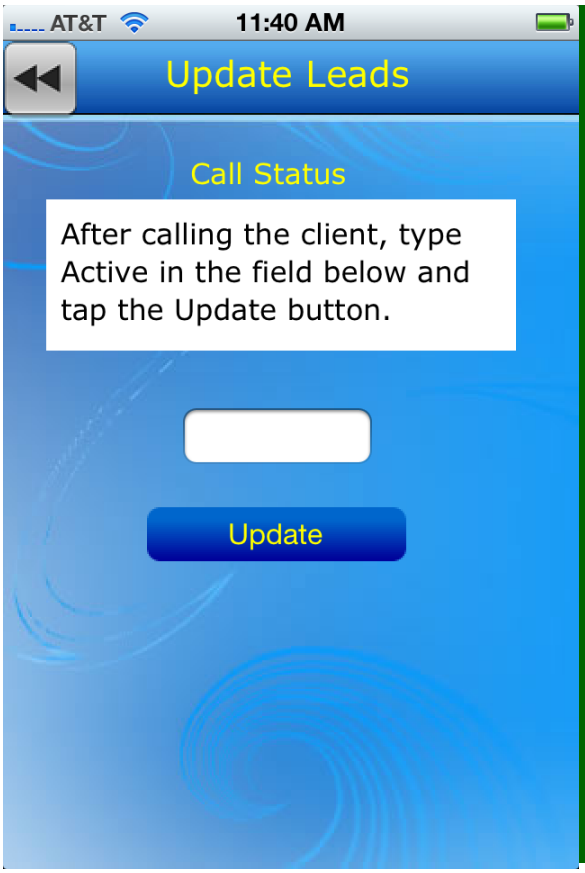
This approach is simple but has two weaknesses.

- Asking users to type the word to use as the update value risks the user misspelling it or typing the wrong word. ViziApps will insert whatever the user enters. A more complex but controlled solution is to give users a preset list of update terms in a picker (iPhone) or a dropdown (Android) and let users pick the term from the list. This is similar to the two Delete command approaches - predefined and user-specified. For information:
  - See "Delete From - Examples - Predefined Value Deletion" on page 42
  - See "Delete From - Examples - User-Specified Value Deletion" on page 47
- This approach changed the values of all the entries to Active. What if we only want to change the values of selected entries, such as only changing status = Green to Active? For that, we need conditions. See "Introduction to Conditions in ViziApps" on page 53

For an example of the mechanics of the update command - See "Update - Examples" on page 32.

Update - Examples

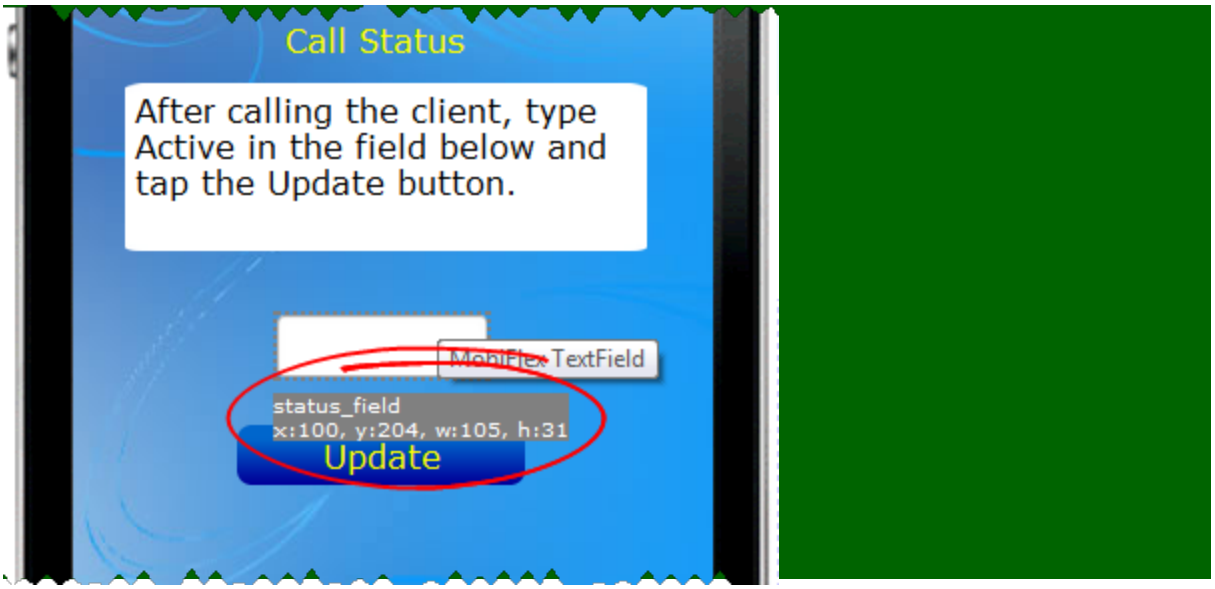
To update a field, a user enters the update value (the value that will replace the old value) in a field on the mobile device, then specifies the spreadsheet field whose value should be replaced by the new value. For example, in the image below, typing the word Active (or any word) in the entry field and tapping the Update button replaces the contents of all specified fields, in this case the *status* field, with the new status Active.



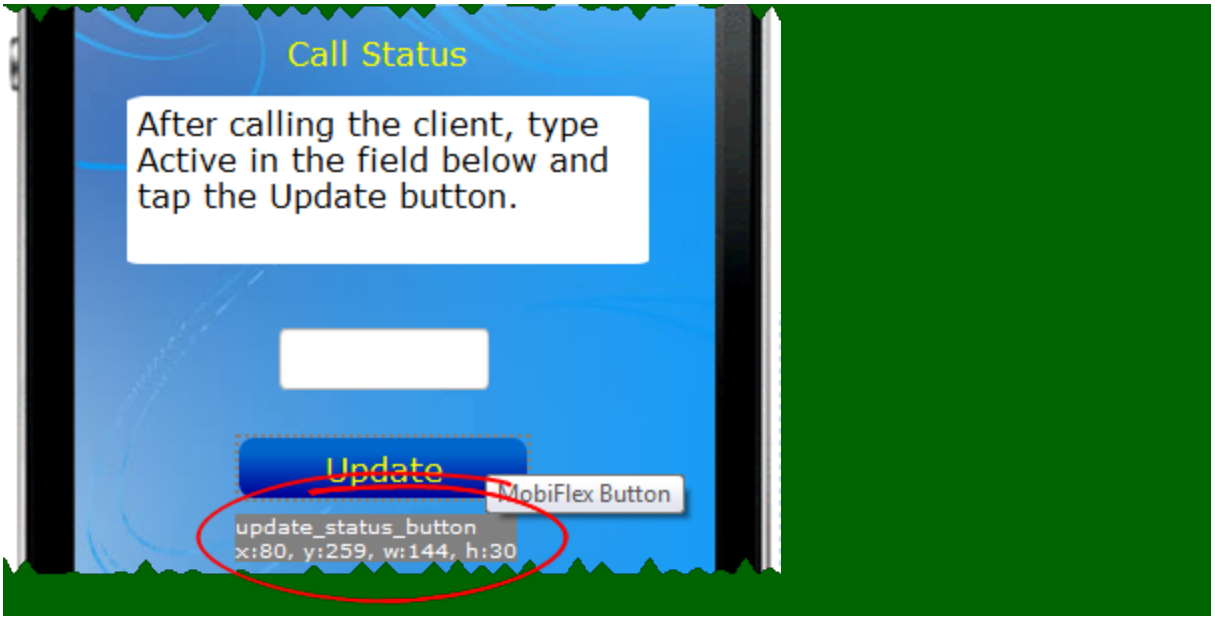
Here's a copy of the spreadsheet.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

We need to know the name of the field in which users can enter the replacement value and the name of the tap event - the button. The name of the entry field is *status_field*, as shown below.

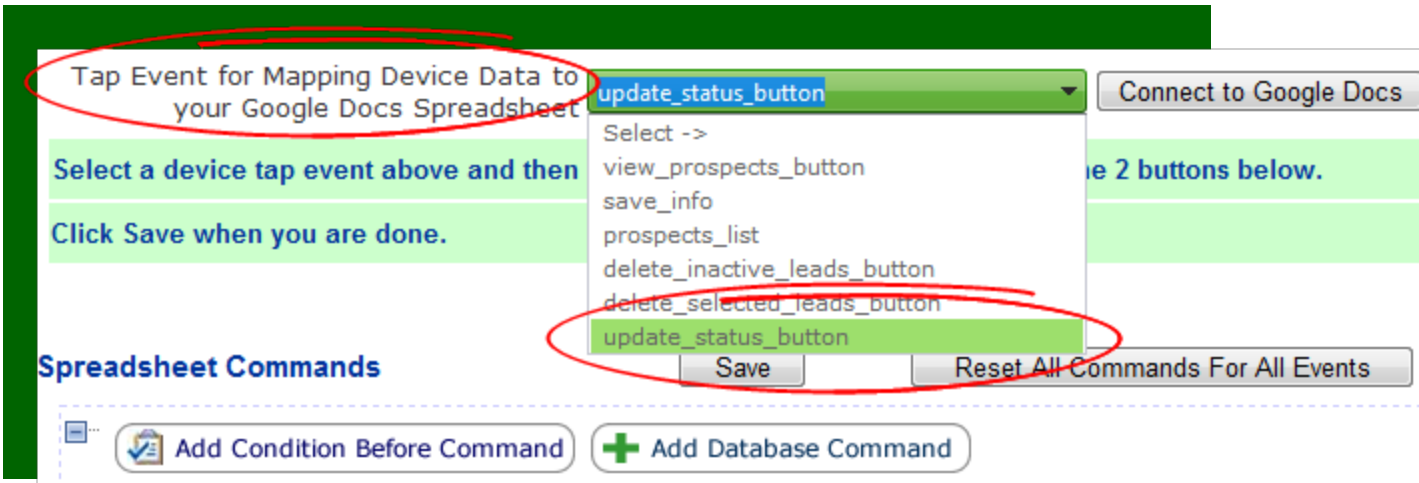


The name of the tap event is the picker name, here *delete_leads_picker*, which we can determine by hovering over the Update button, as shown below.

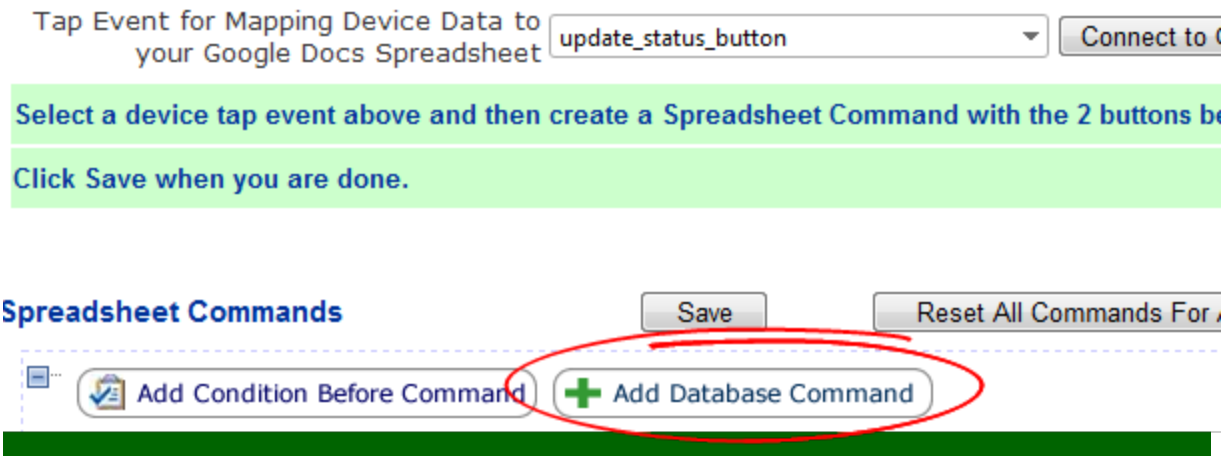


We now have enough to set up the code.

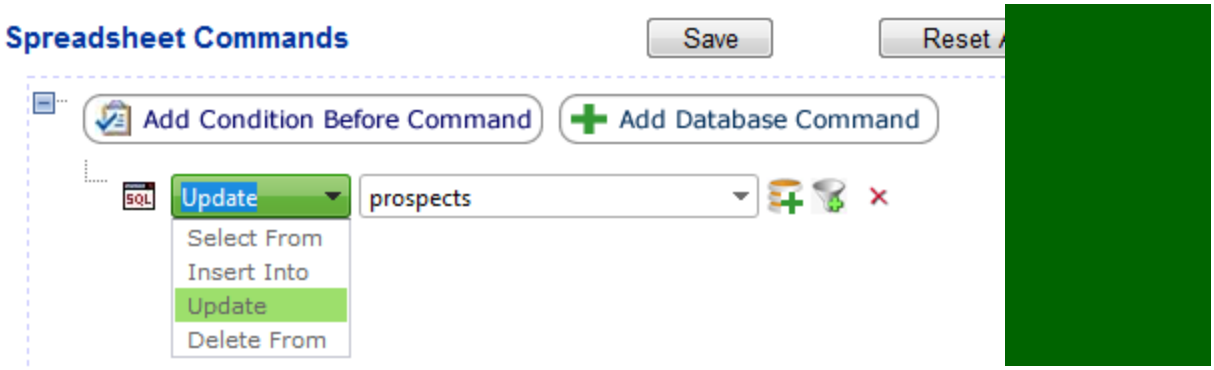
1. Go to the Manage Data page in ViziApps and select *update_status_button* from the Tap Event... field.



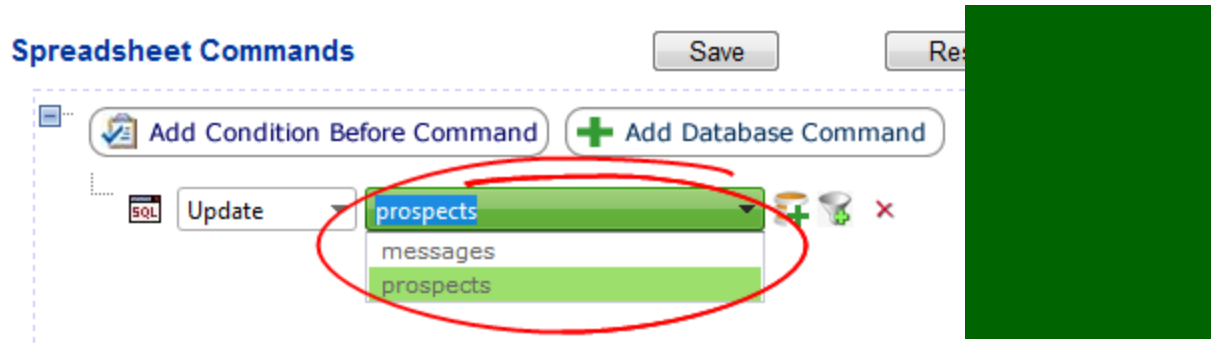
2. Click the Add Database Command button.



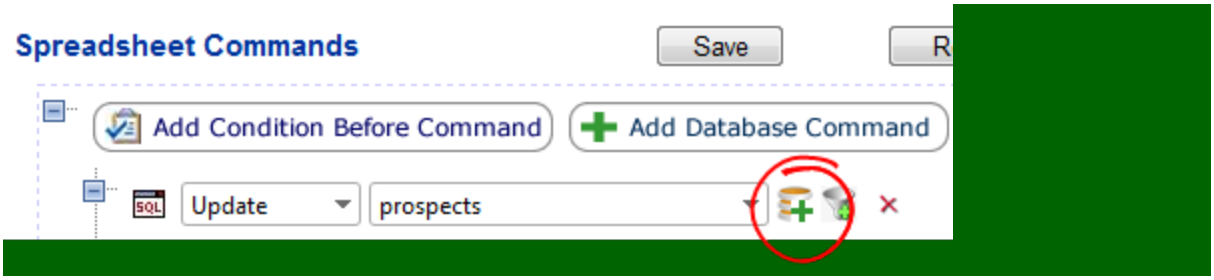
3. Select the Update command from the command pulldown.



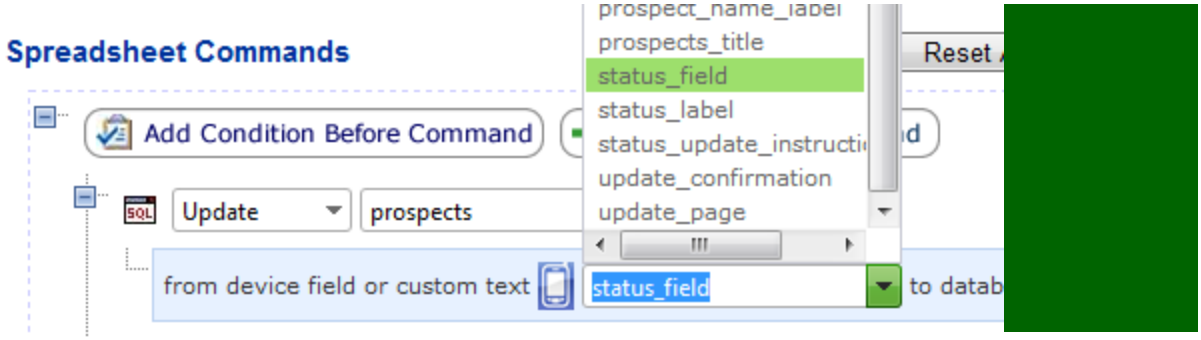
4. Select the worksheet from the worksheet pulldown.



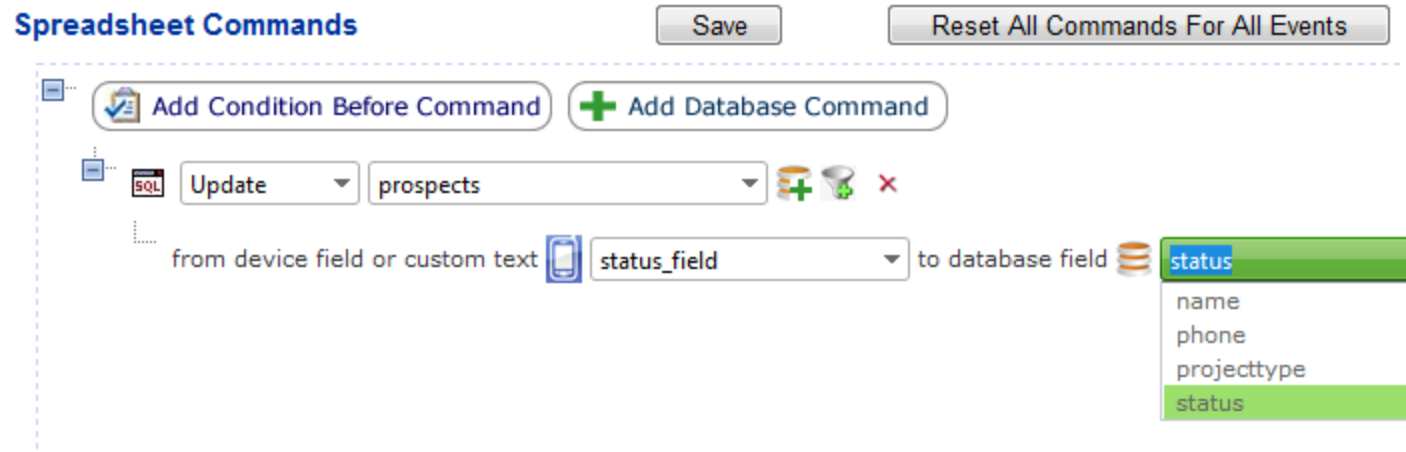
5. Click the Add a Field icon to specify the field containing the replacement value.



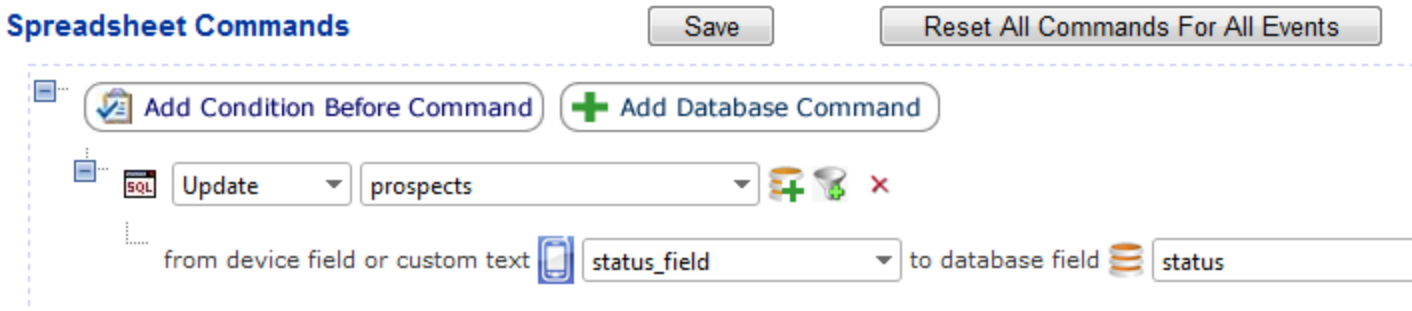
6. Click the pulldown for the device field and select the desired field name, *status_field* in the example below.



7. Click the pulldown for the database field whose values are to be replaced, in this case the *status* field as shown below.



The image below shows the completed entry.



The result is the change in every status field from its current entry to Active, as shown in the image below. First the original spreadsheet.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

Then the changed spreadsheet.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Active		
7	Bisbee	555	Site	Active		
8	Jake	333	Site	Active		
9	Sam	123	App	Active		
10						

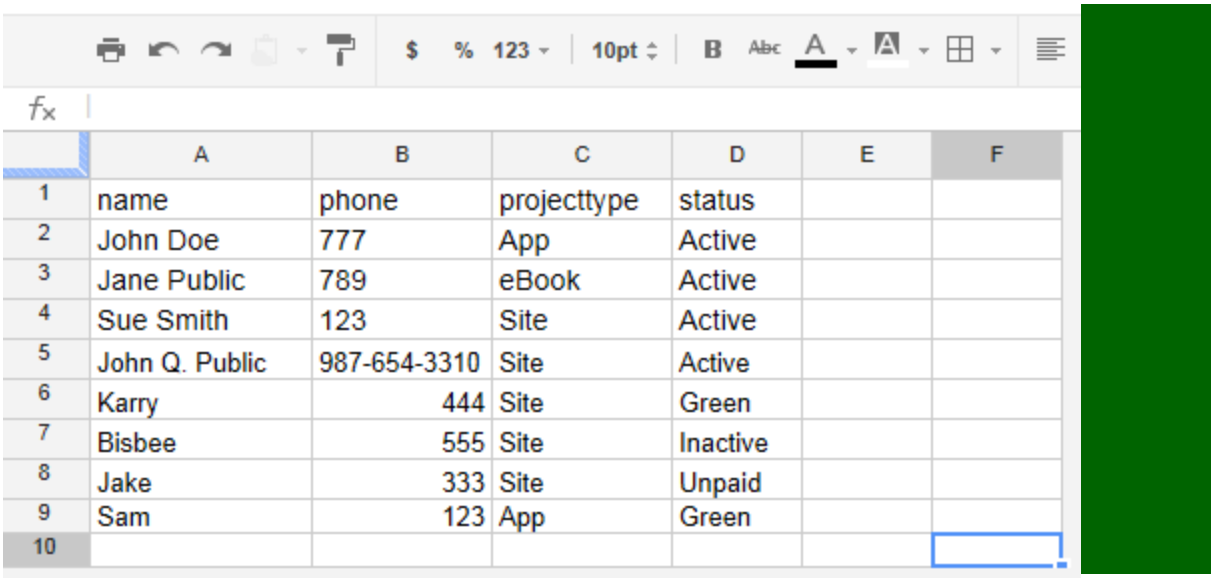
Note that the Update command updates the contents of all fields with the same name, such as all status fields. But what if you only want to update the status of Unpaid entries, such as Jake in line

8, but not change Karry, Bisbee, or Sam on lines 6, 7, and 9. That requires using conditions, covered in another section of the help. See "Introduction to Conditions in ViziApps" on page 53

Delete From - Examples

How did the examples in the Delete From - Concepts topic work? See "Delete From - Concepts" on page 38

The image below shows the spreadsheet for this project.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

Notice the field (column) names - *name*, *phone*, *projecttype*, and *status*. We'll use the *status* field as the key in two deletion examples by specifying that the app should delete any entry - e.g. row - in which a specific field has a specific value. For example, we might tell the app to delete any entry for which the value of the *status* field in Unpaid. This would delete the Jake entry in row 8.

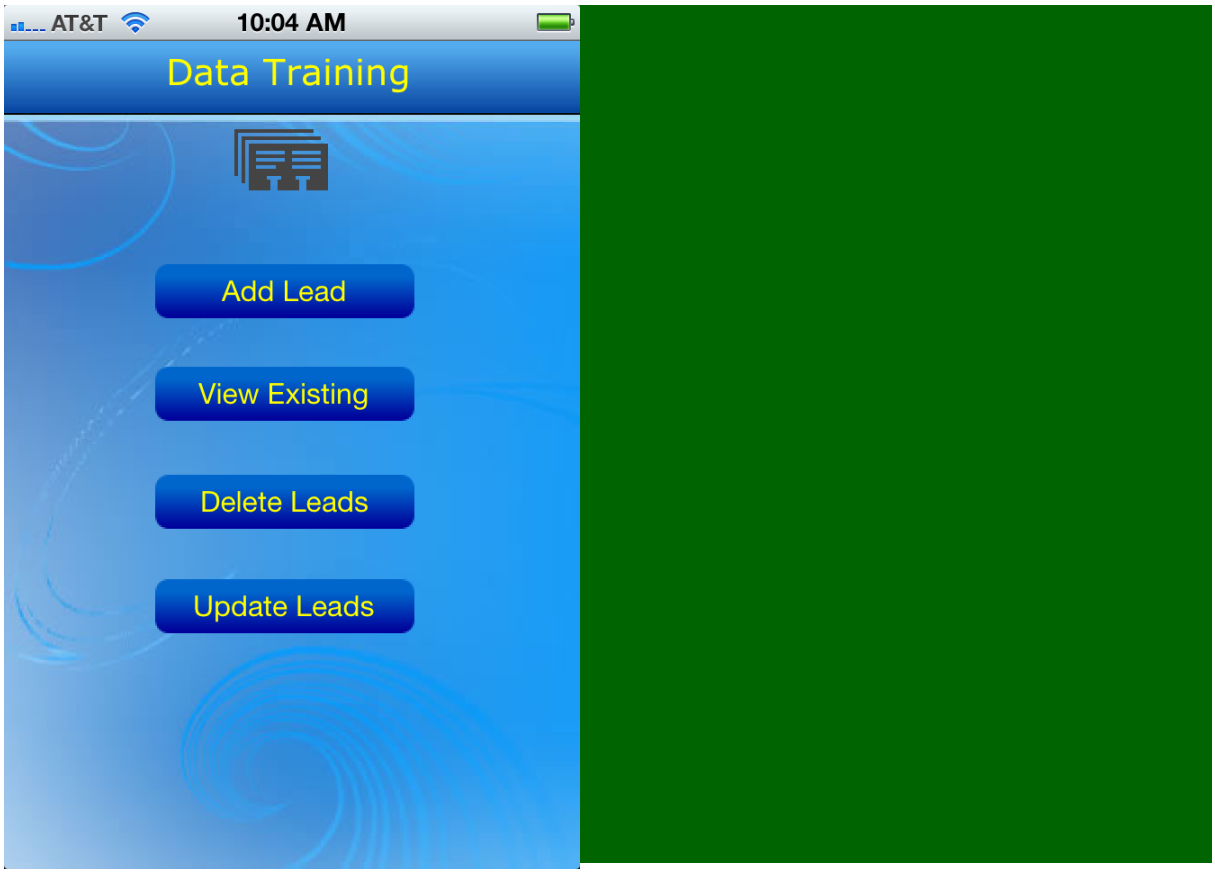
We can do this in two ways. We can tell the app to delete:

- A row in which a field has a predefined value. For example, we can create a button that, when tapped, deletes all entries in which the *status* field has the value Unpaid. See "Delete From - Examples - Predefined Value Deletion" on page 42
- A row in which a field has a value selected by the reader. For example, we can create a control such that, if the reader selects "Green" as the value for the *status* field, the app will delete all entries with that value, such as the Karry and Sam entries in rows 6 and 9 above. See "Delete From - Examples - User-Specified Value Deletion" on page 47

Delete From - Concepts

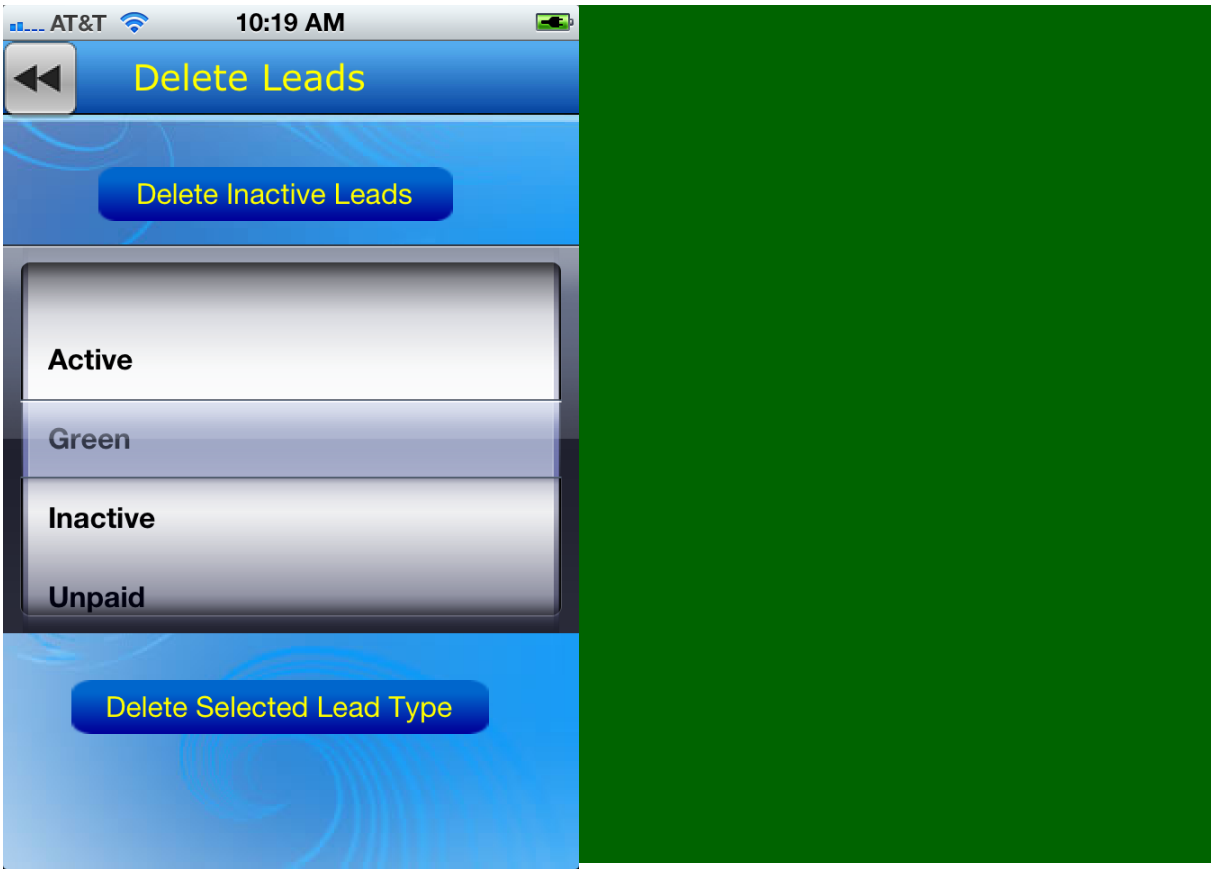
The Delete From command deletes data from fields on the spreadsheet.

For example, tapping the Delete Leads button on the page below.

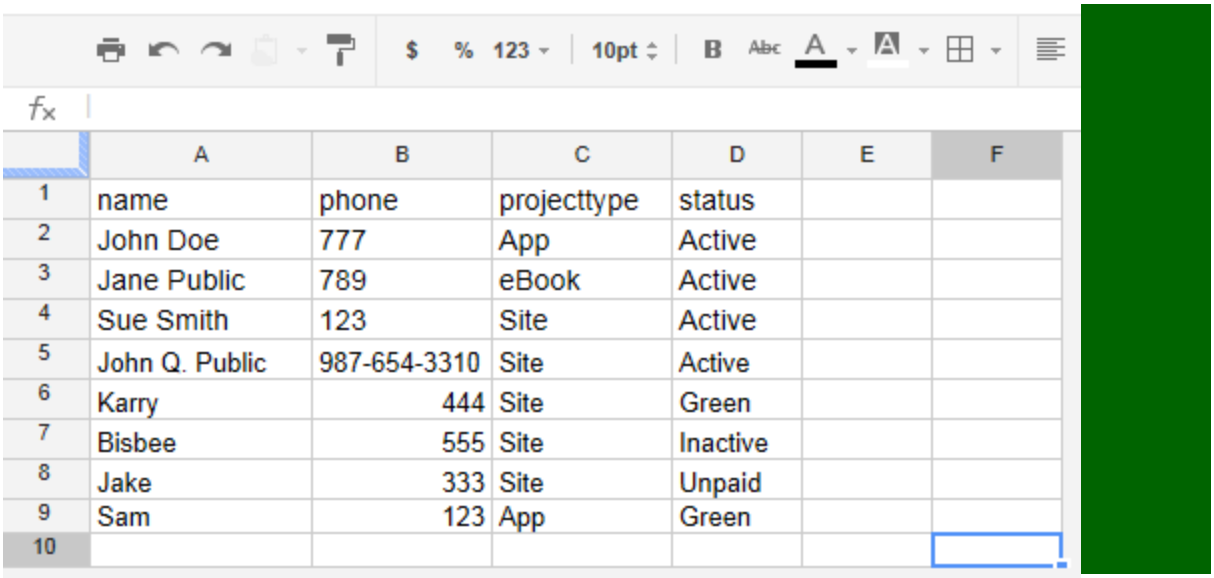


Opens the page below that offers two ways to delete entries:

- Deleting entries with a predefined value (here, entries whose status in the spreadsheet is set as "Inactive").
- Letting readers select the value of the entries to delete (here, by selecting from Active, Green, Inactive, or Unpaid status).

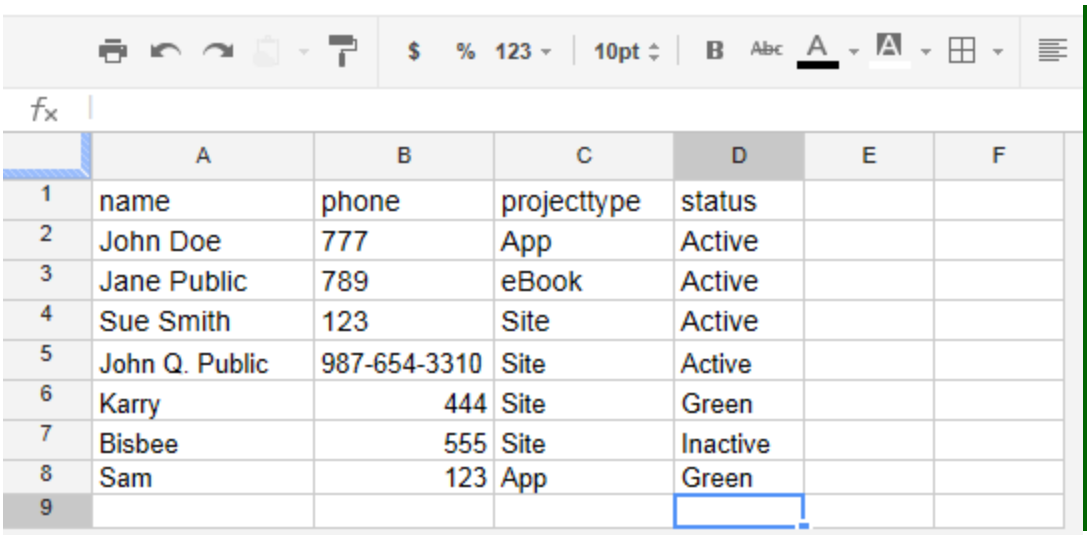


The screen below shows the spreadsheet and the data before a deletion.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

And the screen below shows the spreadsheet and the data after deleting all entries whose status is "Unpaid".



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Sam	123	App	Green		
9						

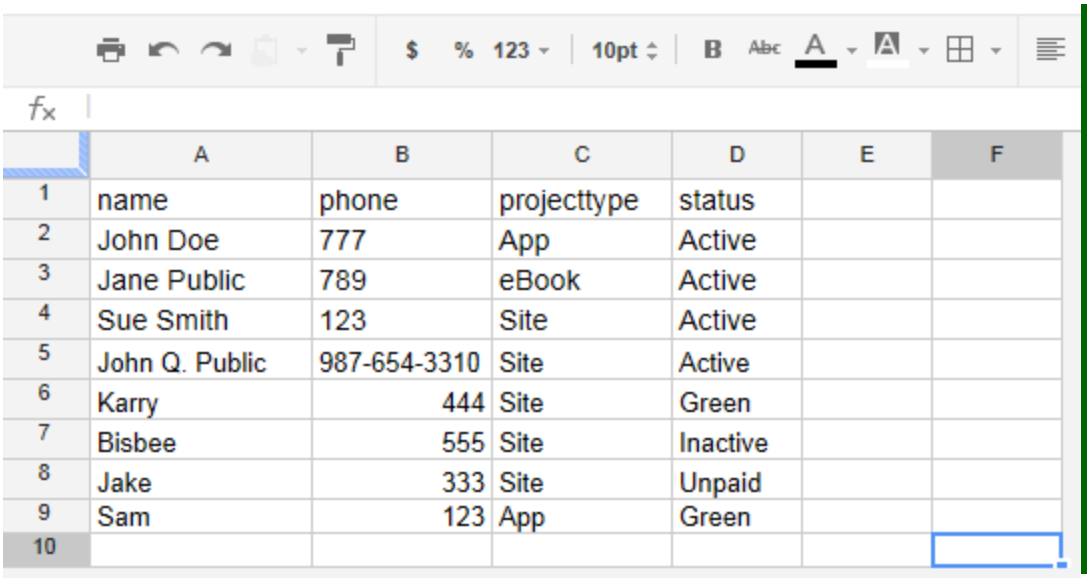
ViziApps deleted all entries, Jake in this case, whose status is Unpaid.

For an example of the mechanics of these two Delete From commands - See "Delete From - Examples" on page 41.

Delete From - Examples

How did the examples in the Delete From - Concepts topic work? See "Delete From - Concepts" on page 38

The image below shows the spreadsheet for this project.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

Notice the field (column) names - *name*, *phone*, *projecttype*, and *status*. We'll use the *status* field as the key in two deletion examples by specifying that the app should delete any entry - e.g. row - in which a specific field has a specific value. For example, we might tell the app to delete any entry for which the value of the *status* field in Unpaid. This would delete the Jake entry in row 8.

We can do this in two ways. We can tell the app to delete:

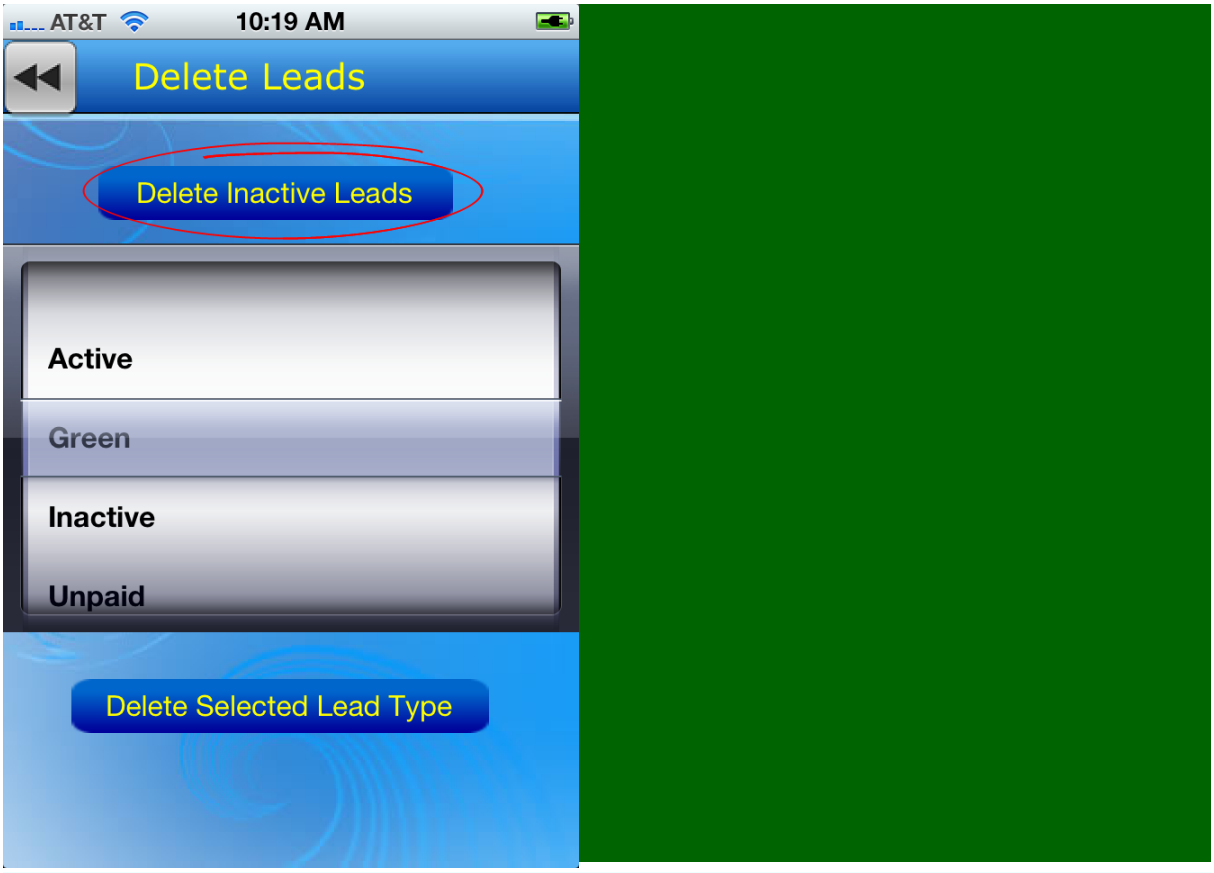
- A row in which a field has a predefined value. For example, we can create a button that, when tapped, deletes all entries in which the *status* field has the value Unpaid. See "Delete

From - Examples - Predefined Value Deletion" on page 42

- A row in which a field has a value selected by the reader. For example, we can create a control such that, if the reader selects "Green" as the value for the *status* field, the app will delete all entries with that value, such as the Karry and Sam entries in rows 6 and 9 above. See "Delete From - Examples - User-Specified Value Deletion" on page 47

Delete From - Examples - Predefined Value Deletion

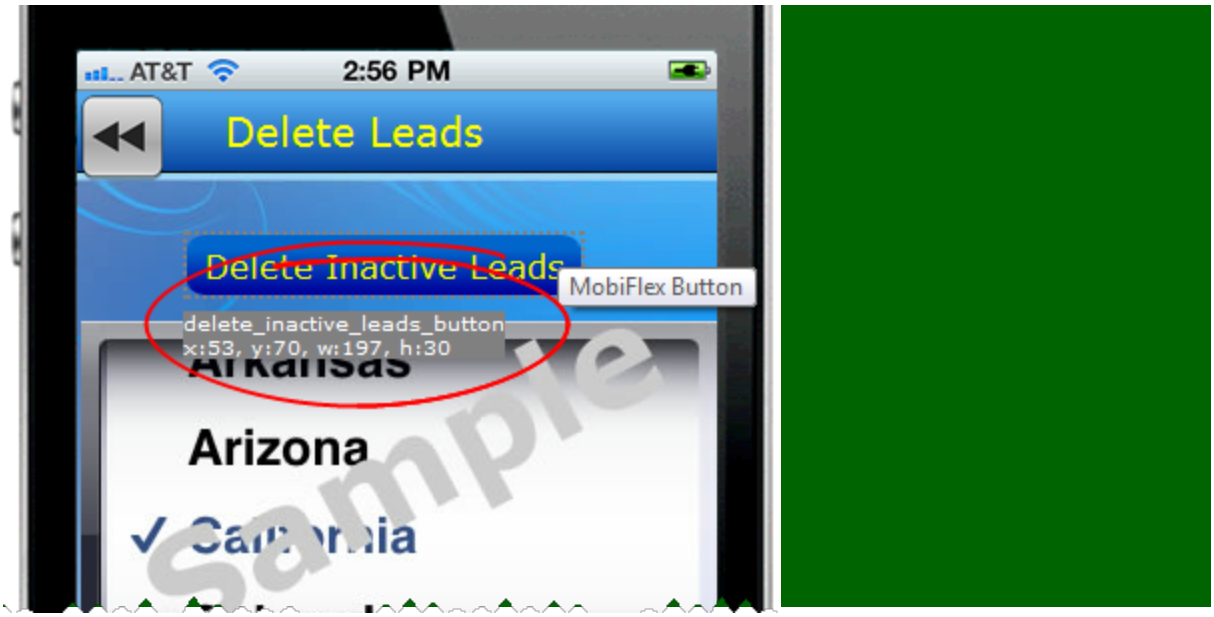
In this approach, the value to be deleted is hard-coded into the deletion button. For example, in the image below, tapping the Delete Inactive Leads is hard-coded to delete all entries whose status is Inactive.



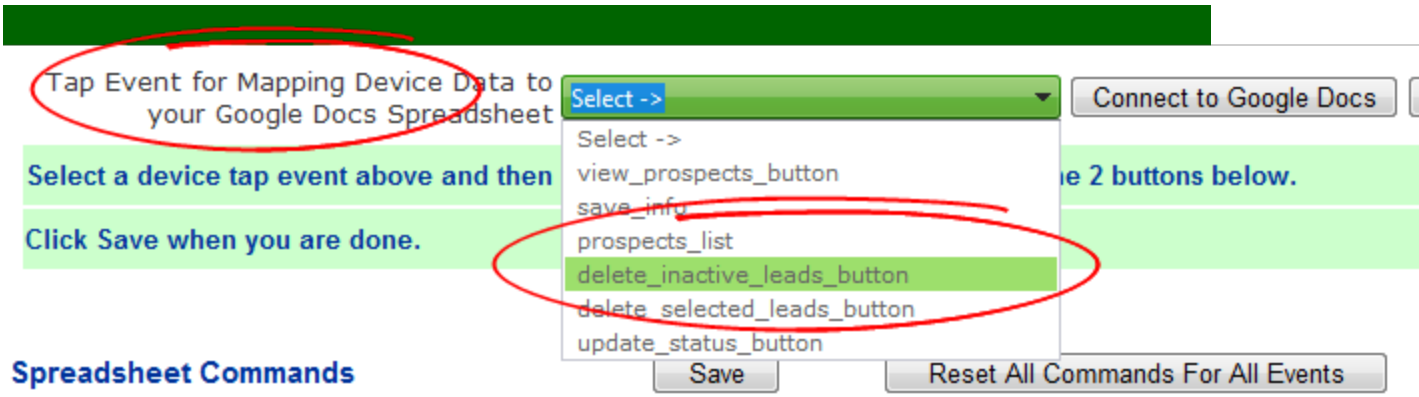
Here's a copy of the spreadsheet.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

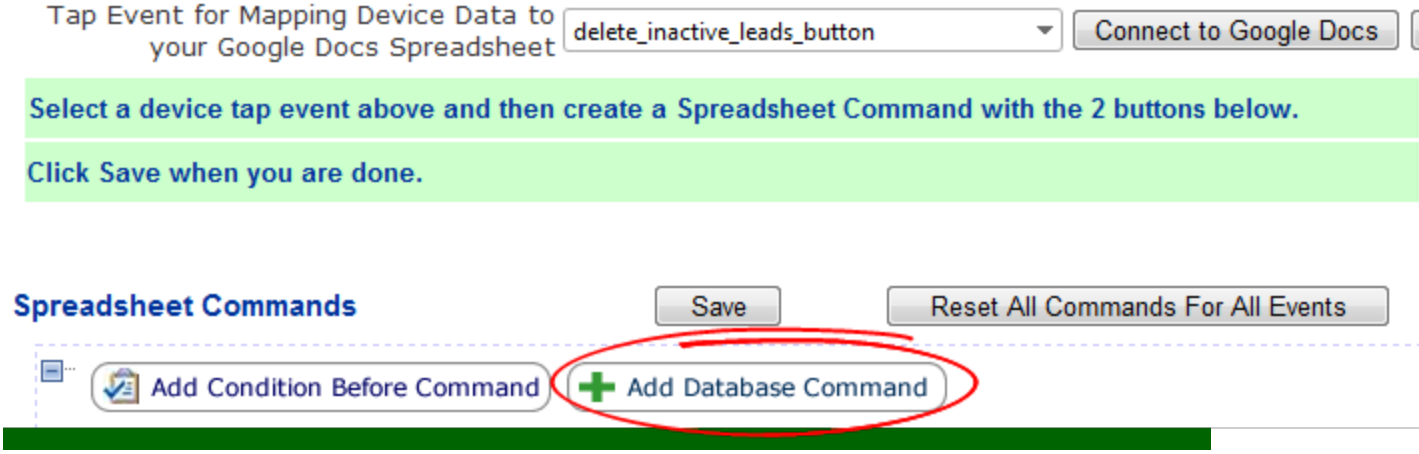
The name of the tap event for this example is *delete_inactive_leads_button*, which we can determine by hovering over the Delete Inactive Leads button, as shown below.



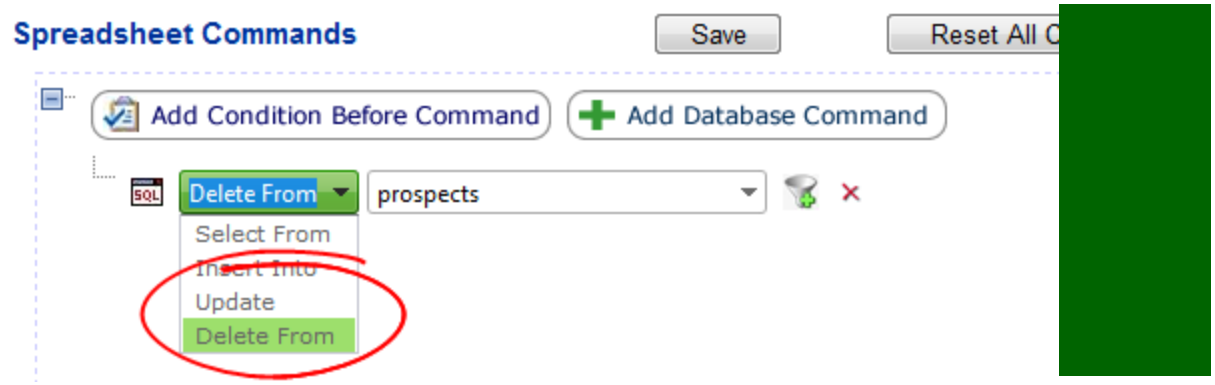
1. Go to the Manage Data page in ViziApps and select *delete_inactive_leads_button* from the Tap Event... field.



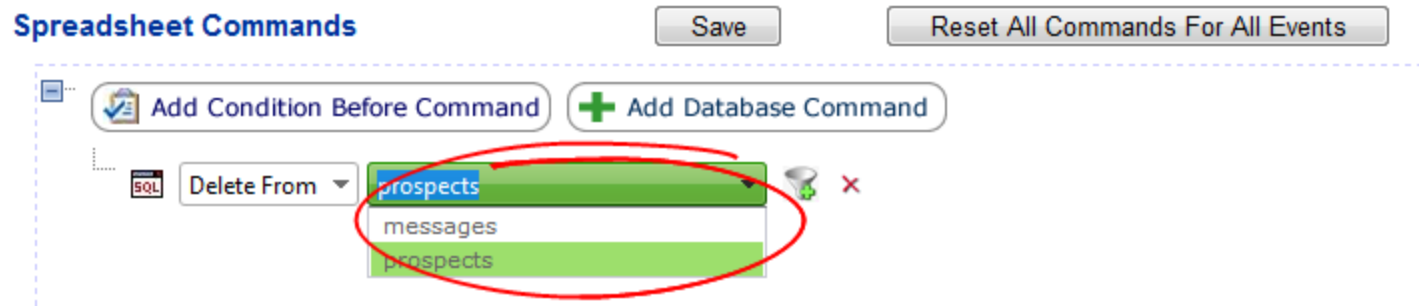
2. Click the Add Database Command button.



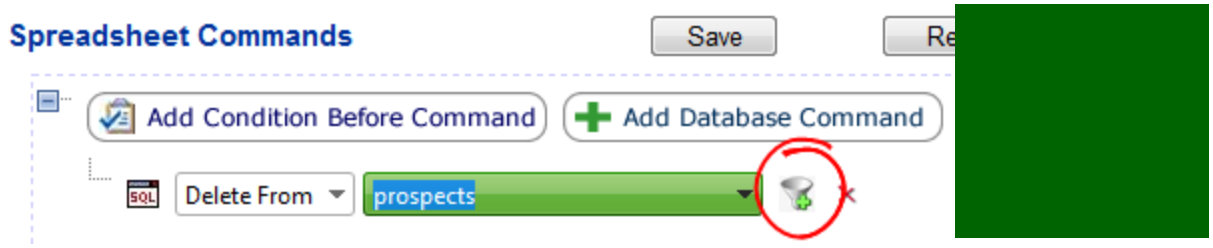
3. Select the Delete From command from the command pulldown.



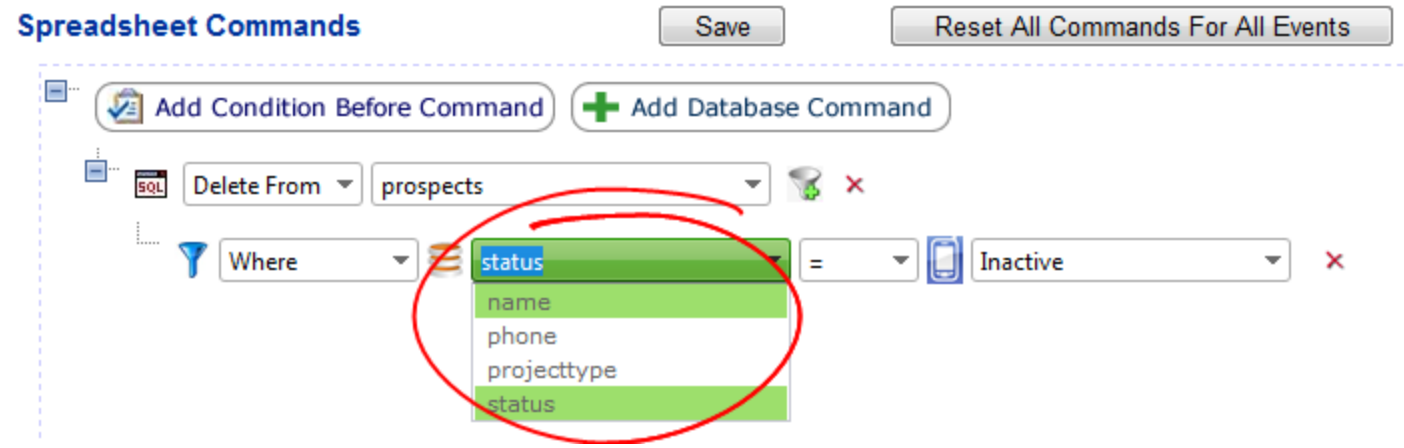
4. Select the worksheet from the worksheet pulldown.



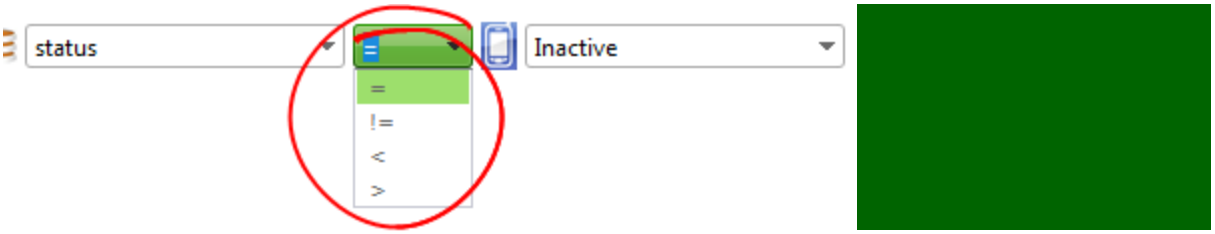
5. Click the Add a Where Condition icon to specify the field value to delete.



6. Click the pulldown for the spreadsheet field and select the desired field name, *status* in the example below.

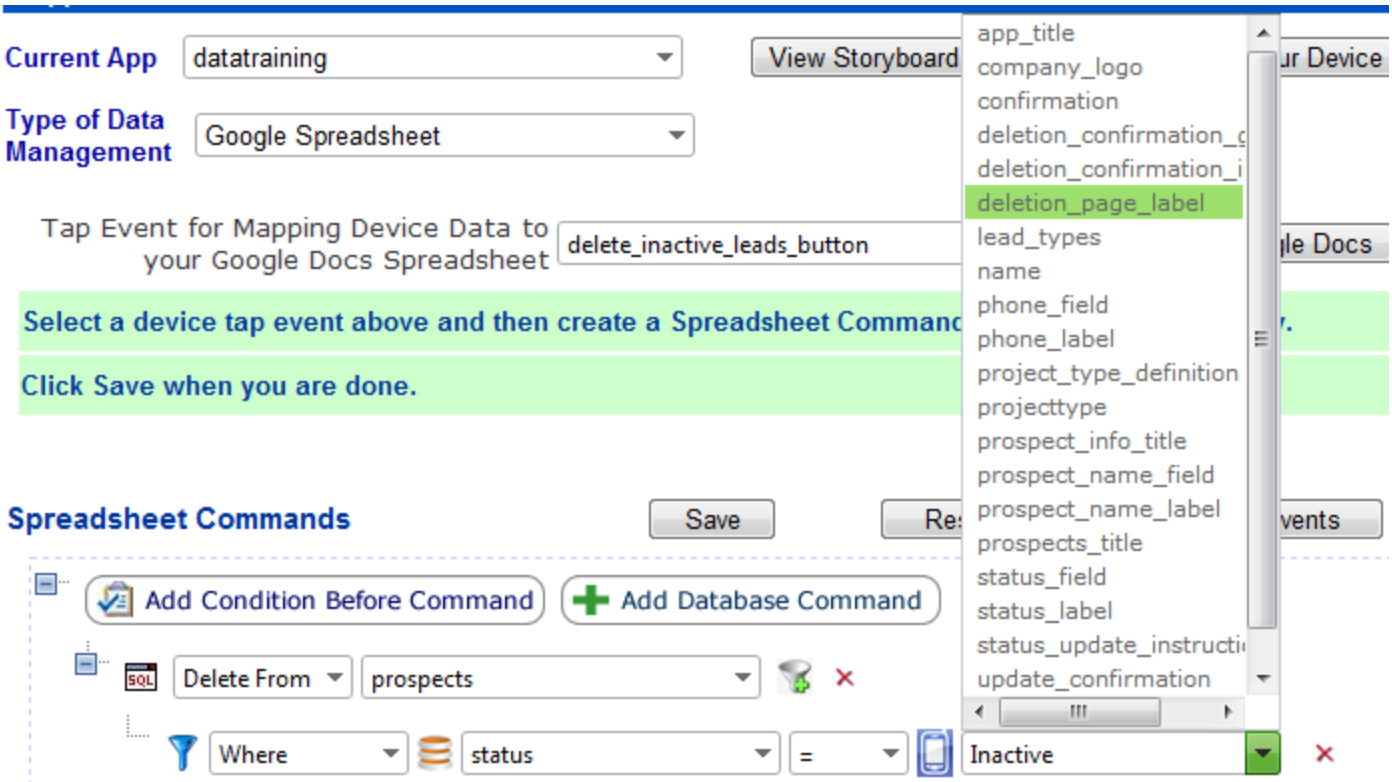


7. Click the pulldown for the arithmetic operator - equal or not-equal - to specify how the deletion will work.

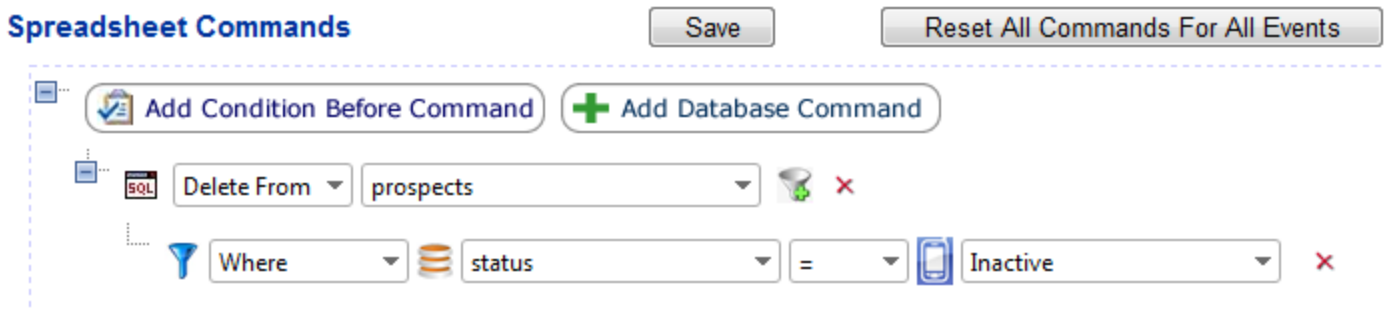


Selecting status=Inactive deletes all entries with a status of Inactive, but selecting status!=Inactive deletes all entries *except* those with a status of Inactive. Selecting < or > lets you specify whether the factor for deletion is to be greater than or less than some value.

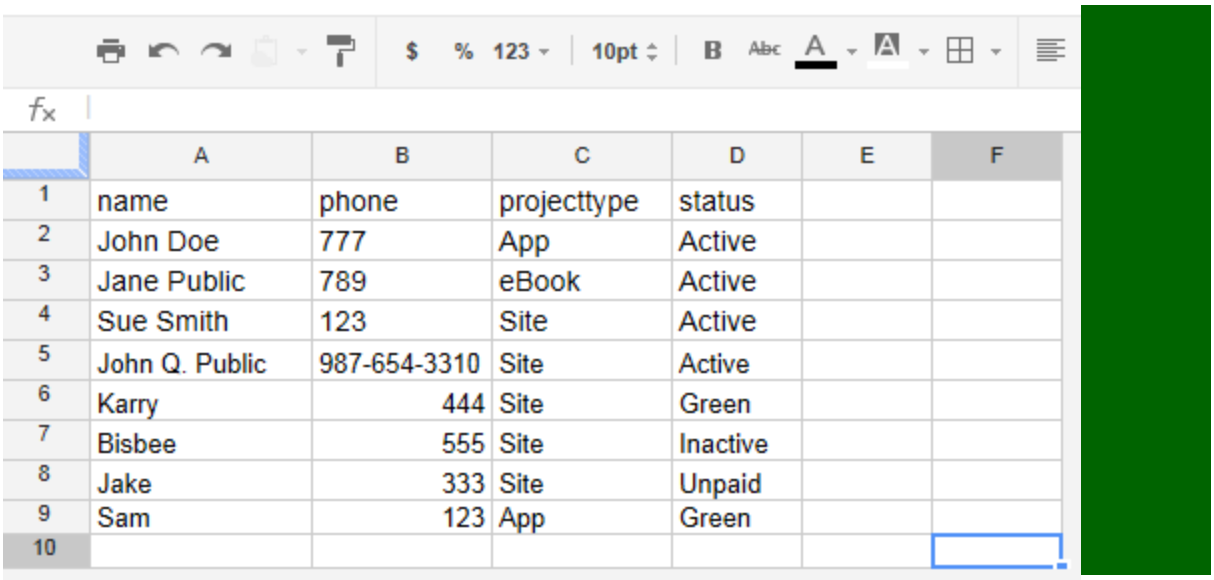
8. Click the pulldown for the device field and select or type the desired value. In the example below, the desired value is *Inactive*, which is not in the list of available fields. Instead, it's a value that a user typed when making the overall entry.



The image below shows the completed entry.



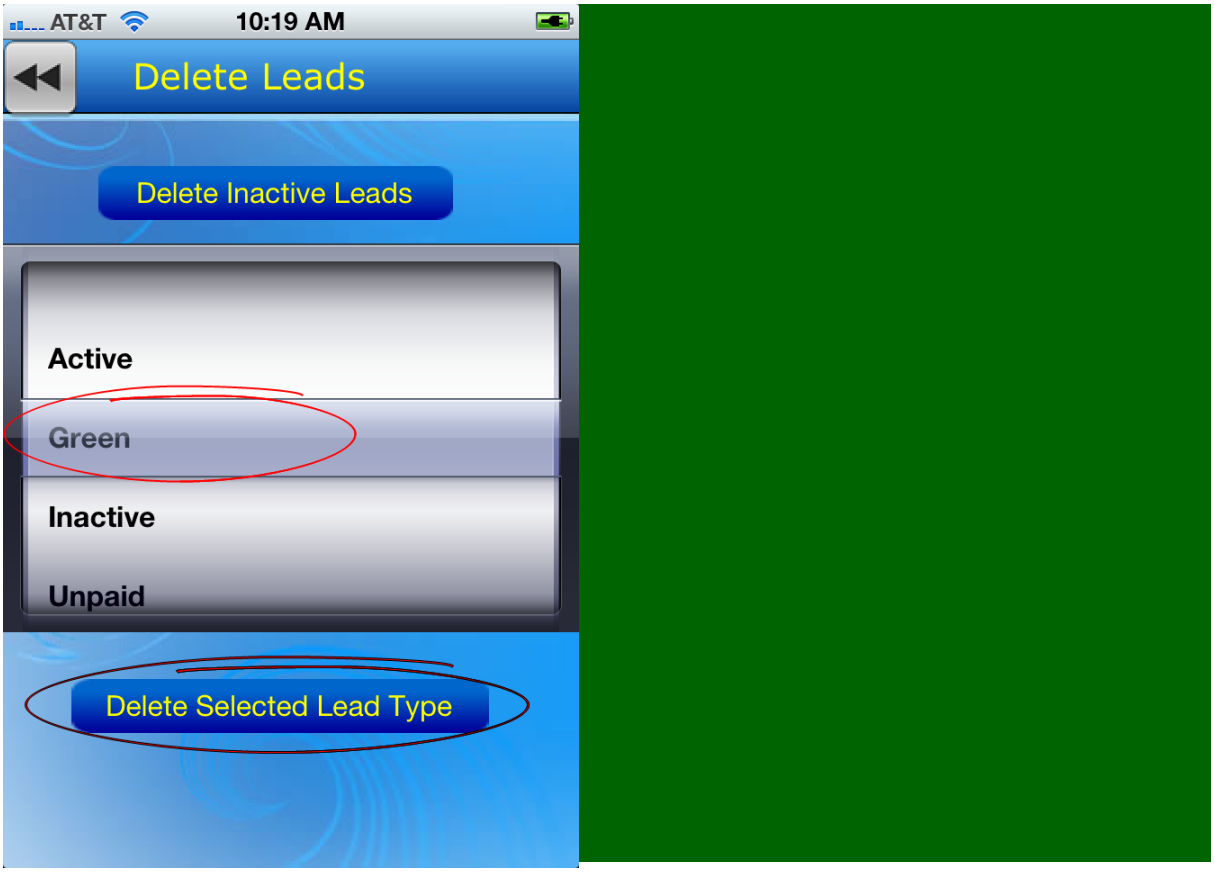
The result is the deletion of any entry whose status has the value Inactive, the Bisbee entry on line 7 in the image below.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

Delete From - Examples - User-Specified Value Deletion

In this approach, the reader selects the value to be deleted by clicking the desired option on the picker (iPhone) or dropdown (Android). For example, in the image below, tapping the Green option on the picker, then tapping the Delete Selected Lead Type button at the bottom deletes all entries whose status is Green, in this case the Karry and Sam entries.

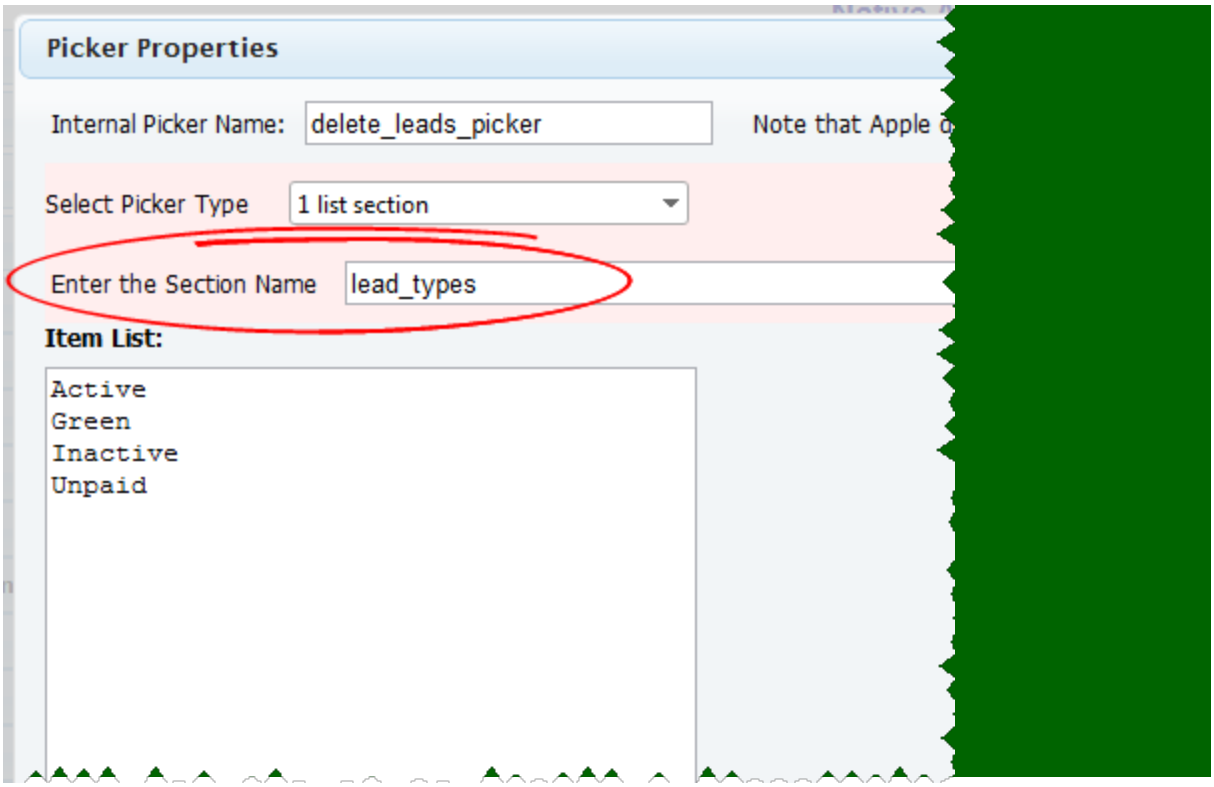


Here's a copy of the spreadsheet.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

The name of the field in which readers can select the deletion value is the picker name, here *delete_leads_picker*, which we can determine by hovering over the picker, as shown below.

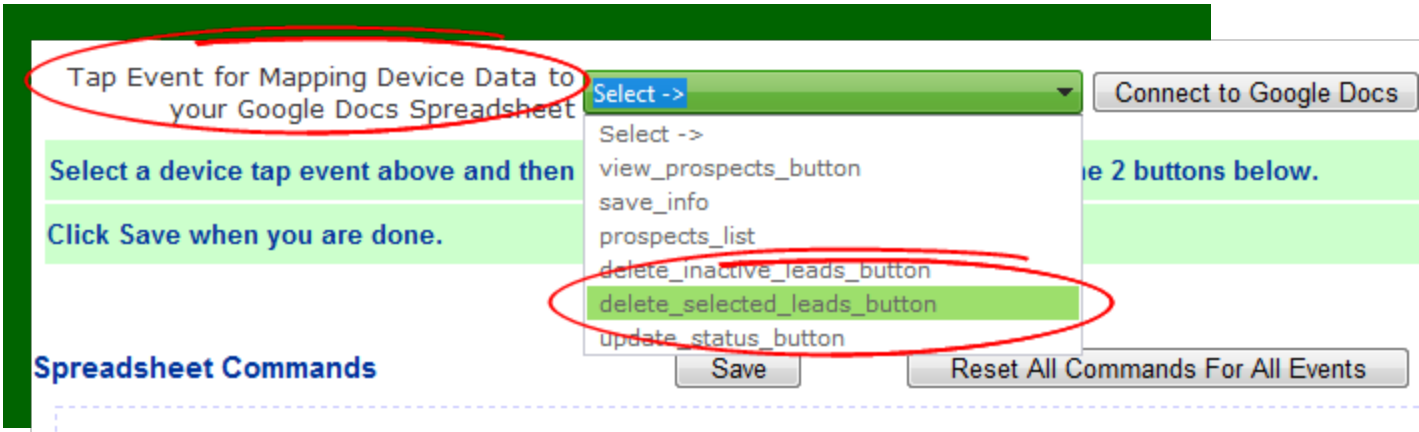
However, a picker can have one or more fields, each with a programmatic section name that we'll use to refer to data in that field. To find the section name, double-click on the picker to open its properties window, shown below.



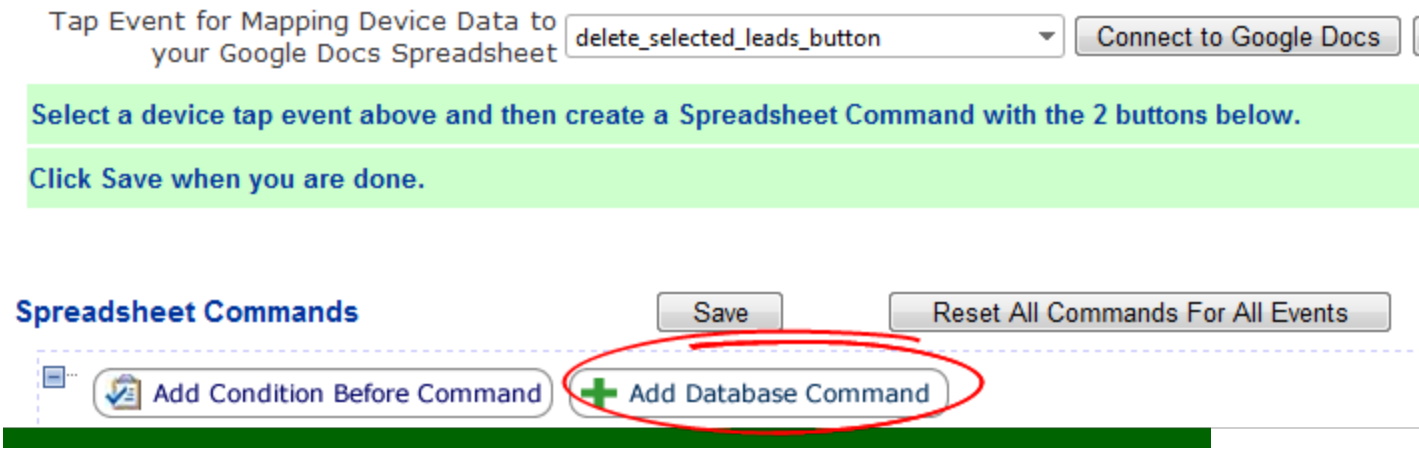
The name of the tap event for this example is *delete_selected_leads_button*, which we can determine by hovering over the Delete Selected Lead Type button, as shown below.

1. Go to the Manage Data page in ViziApps and select *delete_selected_leads_button* from

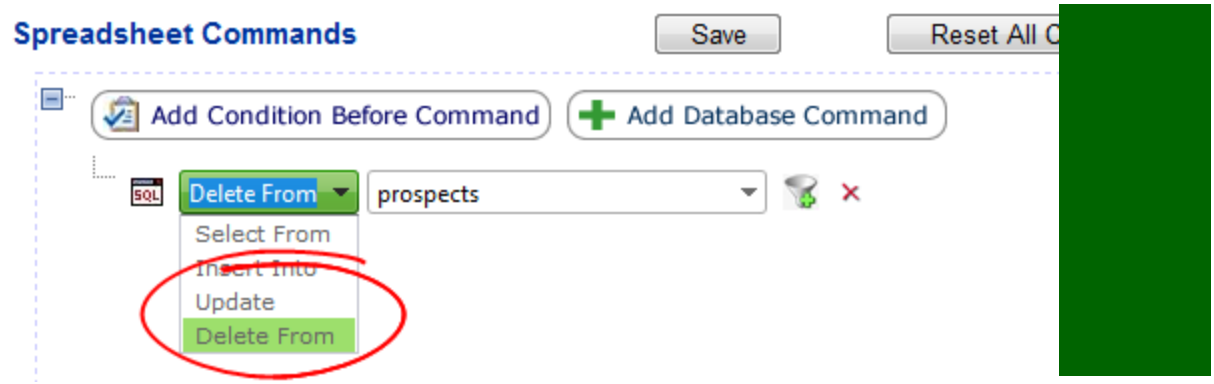
the Tap Event... field.



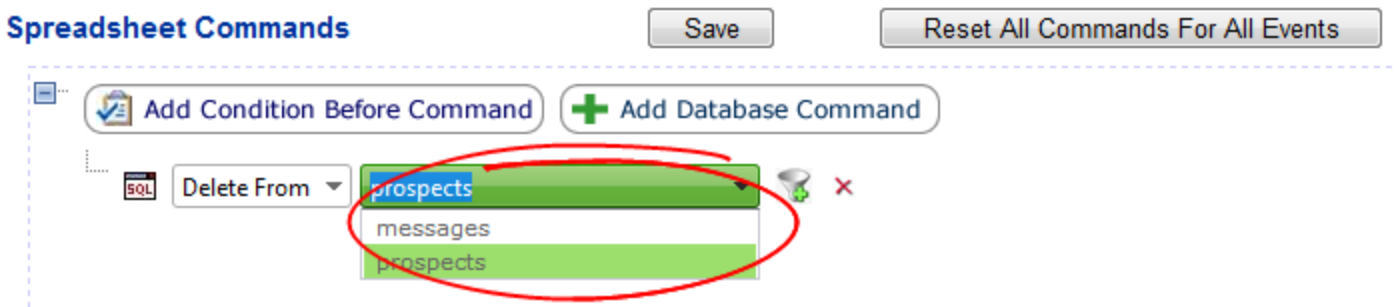
2. Click the Add Database Command button.



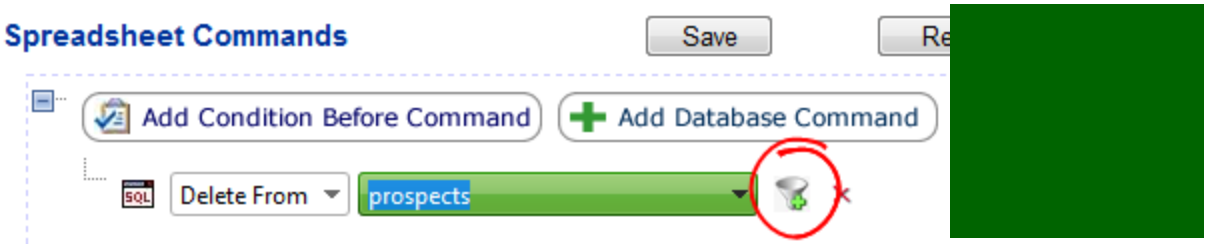
3. Select the Delete From command from the command pulldown.



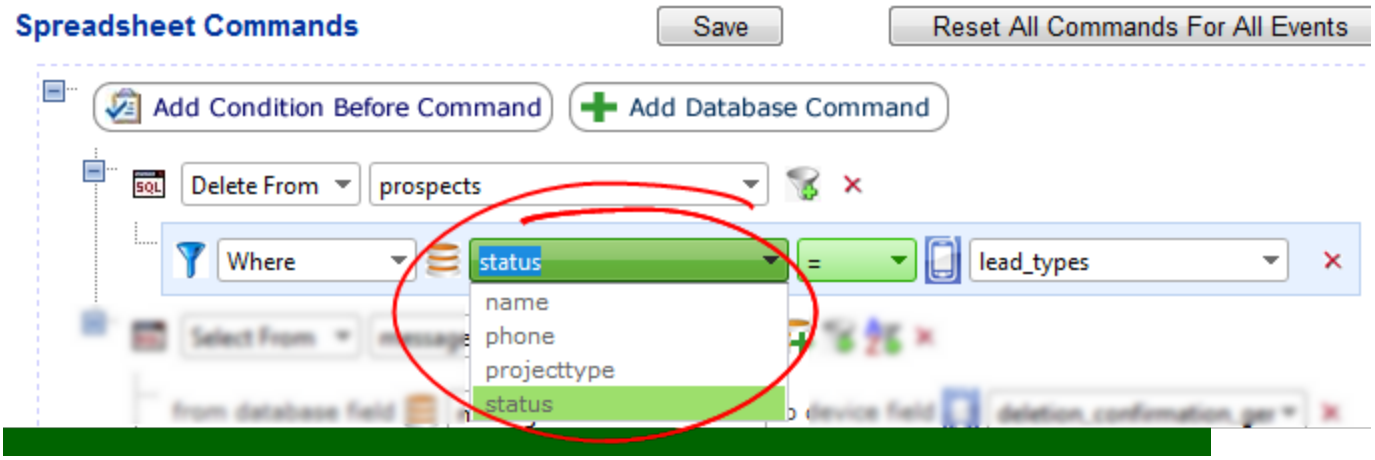
4. Select the worksheet from the worksheet pulldown.



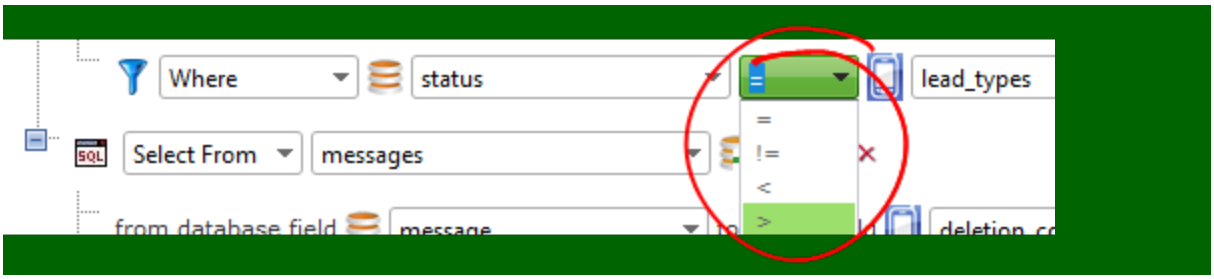
5. Click the Add a Where Condition icon to specify the field value to delete.



6. Click the pulldown for the spreadsheet field and select the desired field name, *status* in the example below.

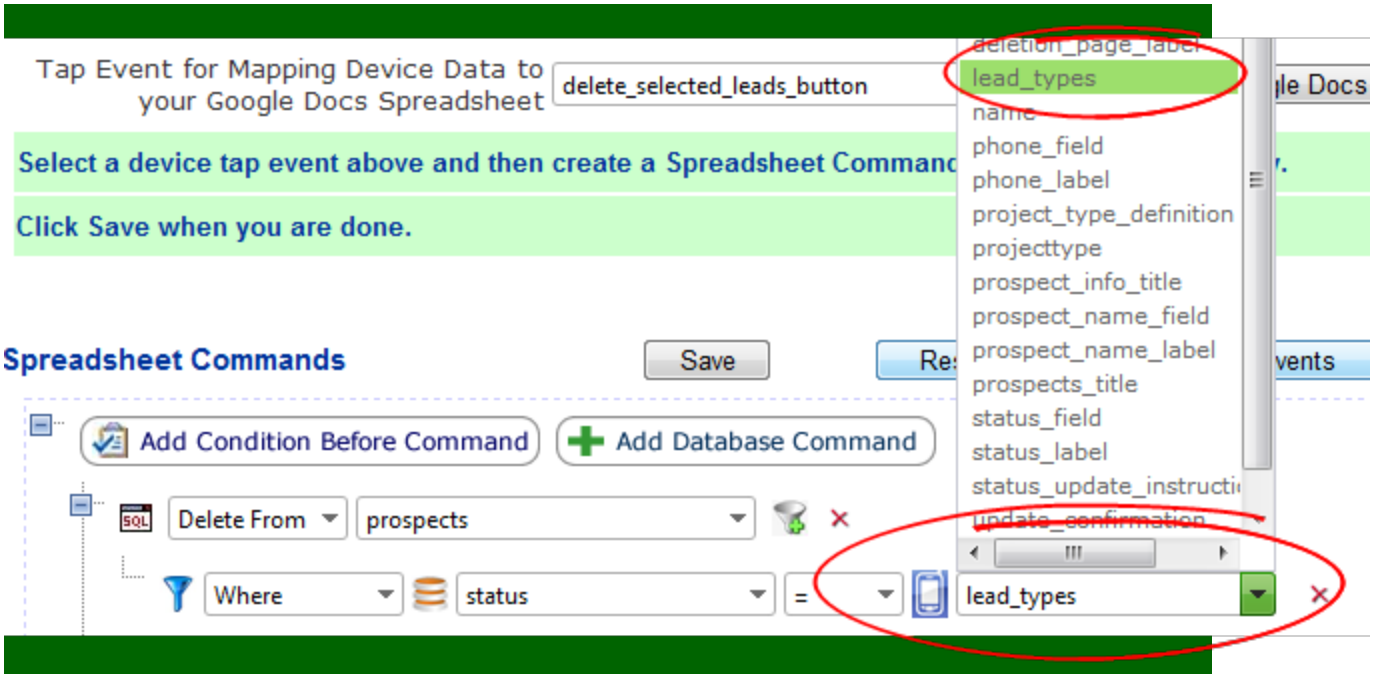


7. Click the pulldown for the arithmetic operator - equal or not-equal - to specify how the deletion will work.



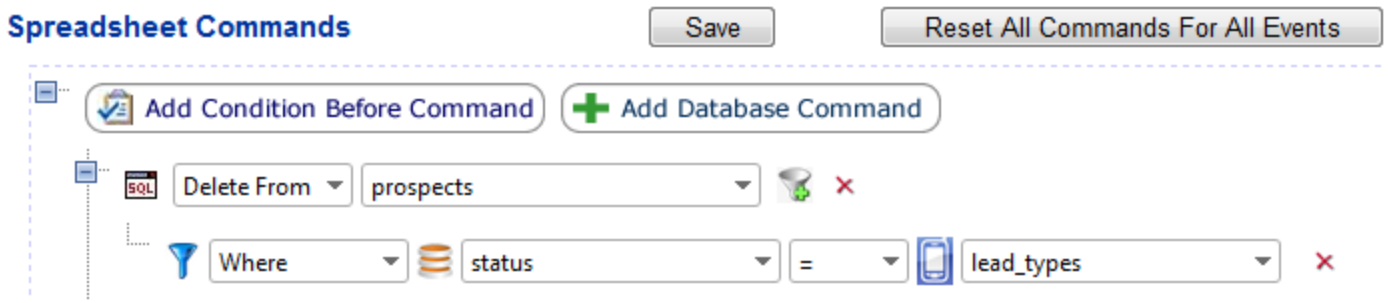
Selecting status=Inactive deletes all entries with a status of Inactive, but selecting status!=Inactive deletes all entries *except* those with a status of Inactive. Selecting < or > lets you specify whether the factor for deletion is to be greater than or less than some value.

8. Click the pulldown for the device field and select or type the desired value. Here, we want to tell the app to delete the value specified when the reader taps the desired value in the picker. That means that we have to tell the app to delete the value stored in the picker's *lead_types* field.

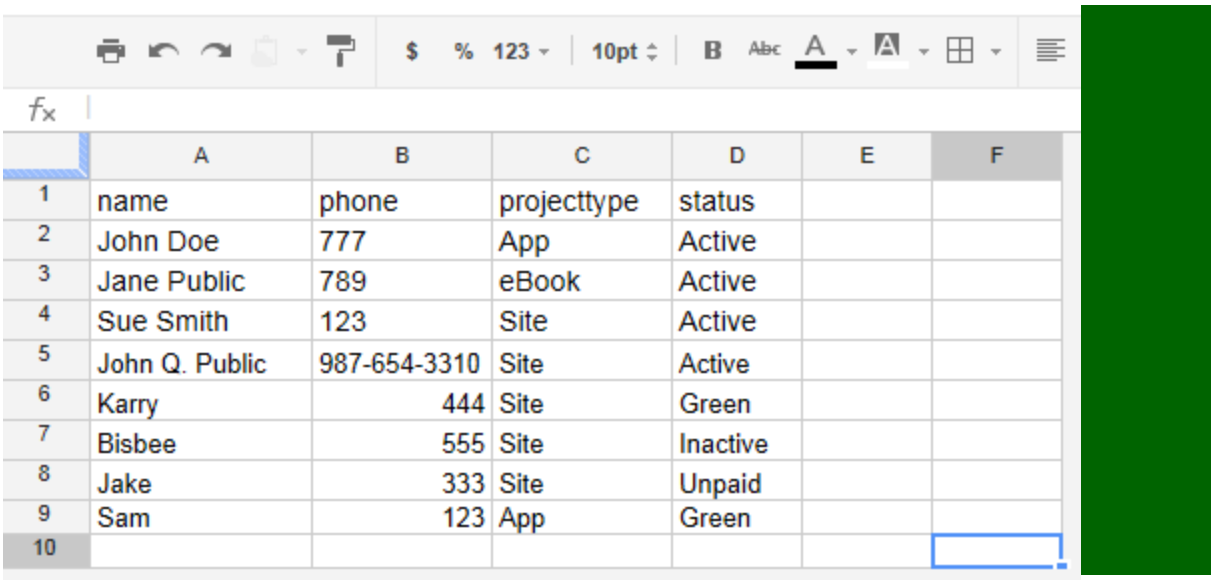


Unlike the hard-coded example, where the value of Inactive was one that the user typed and was not in the list of available fields, *lead_types* is in the list of available fields.

The image below shows the completed entry.



The result is the deletion of any entry whose status value equals Green, which the user selected from the picker (dropdown on an Android) and was thus stored as the value of the *lead_types* field. Specifically, the deletion of the Karry and Sam entries on lines 6 and 9 in the image below.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

Introduction to Conditions in ViziApps

ViziApps' database commands let us pass data between a mobile device and a spreadsheet, update data on a spreadsheet, or delete data from a spreadsheet. But we may want to refine those commands to perform them subject to certain conditions. For example:

- Use the Update command to change an entry's status to Active only if that entry's status is Inactive, rather than changing all entries' status to Active.
- Use the Update command to change a specific entry's status to Active only if its status is Inactive. For example, several clients may have a status of Inactive, but you want to be able to change a specific client's status to Active after calling that client.

In such cases, we want the app to perform a task only on data that meets a condition. To do so, we specify the condition after the command. See "Conditions On Commands" on page 54

- Users must login before the app will perform a tasks, such as requiring users to login in order to post a message. We also want the app to respond differently if the login is right or wrong.

In this case, we want to test for a condition before performing the task. This means we want to specify the condition before the command. This also lets us specify if/then options - what to do if the condition is met or not met, right or wrong, such as logging in successfully vs. unsuccessfully. See "Conditions With Options" on page 53

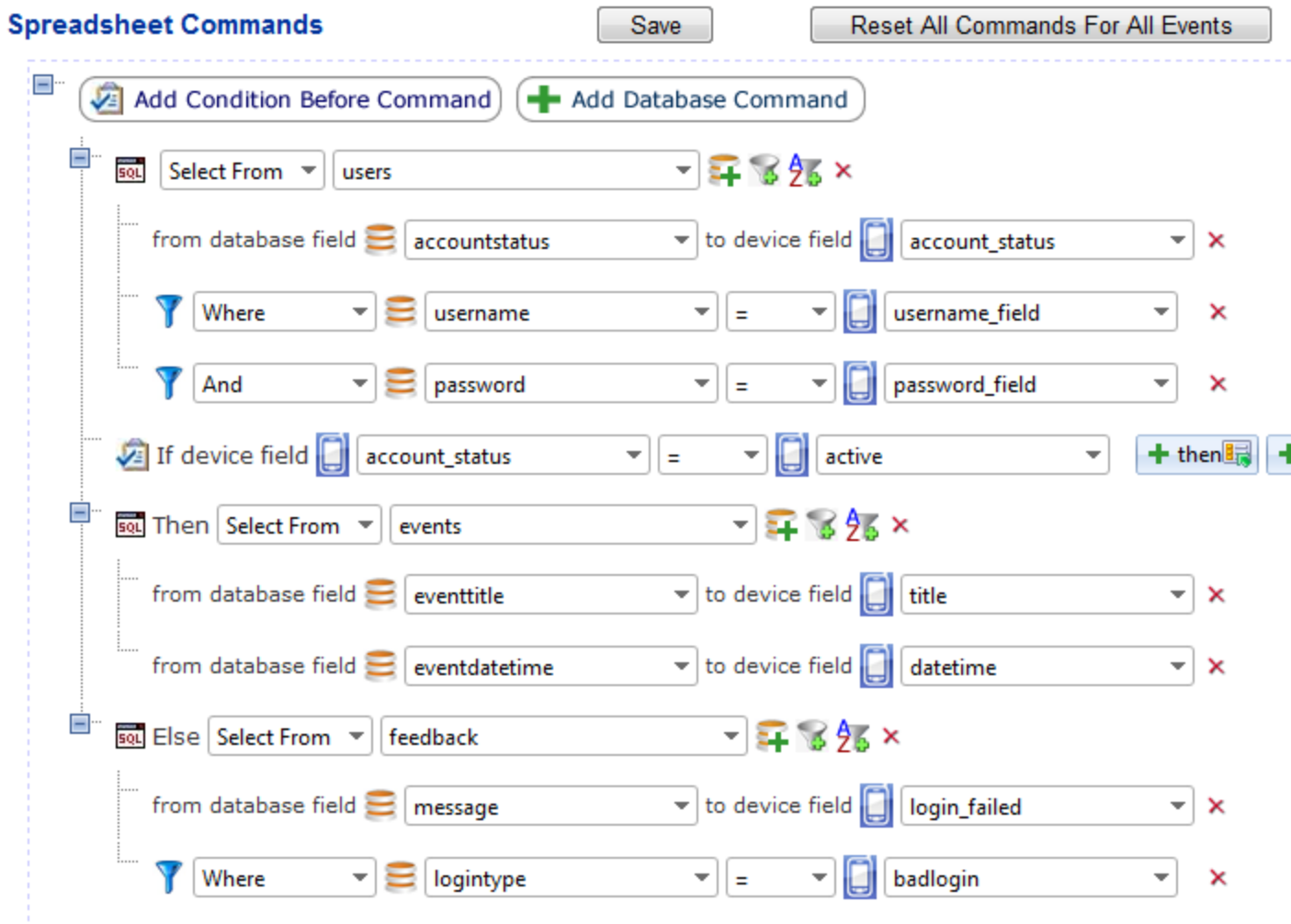
Conditions With Options

We may want a condition to be met before an app can perform a task *and* make the app do one thing if the condition is met and another if it isn't. For example, we want users to login, then provide one set of options if the login is valid and different options if it isn't. We do this using an If/Then/Else statement in the condition. *If* the condition is met, then do X, or *else*, if the condition is not met, do Y.

We'll illustrate this case with the example of a user login. Imagine that an app offers a listing of events but requires that users login in order to read it. For a valid login, the list of events displays. For an invalid login, an error message displays instead.

The command set, shown below, looks more complex than any other examples so far. However, if you're gone through the other command descriptions, you'll see that there's only one new element here - adding the condition before a command and specifying the associated If/Then/Else actions.

Here's the completed data management commands.



In this case, we're passing an *account_status* value from a spreadsheet field to a field on the device. The device field is a "hidden" field because we don't want the users to see it. We just want a field on the mobile device that can hold a value that we'll then test for.

The app then compares the username and password that the user enters on the mobile device to those values in the corresponding database fields.

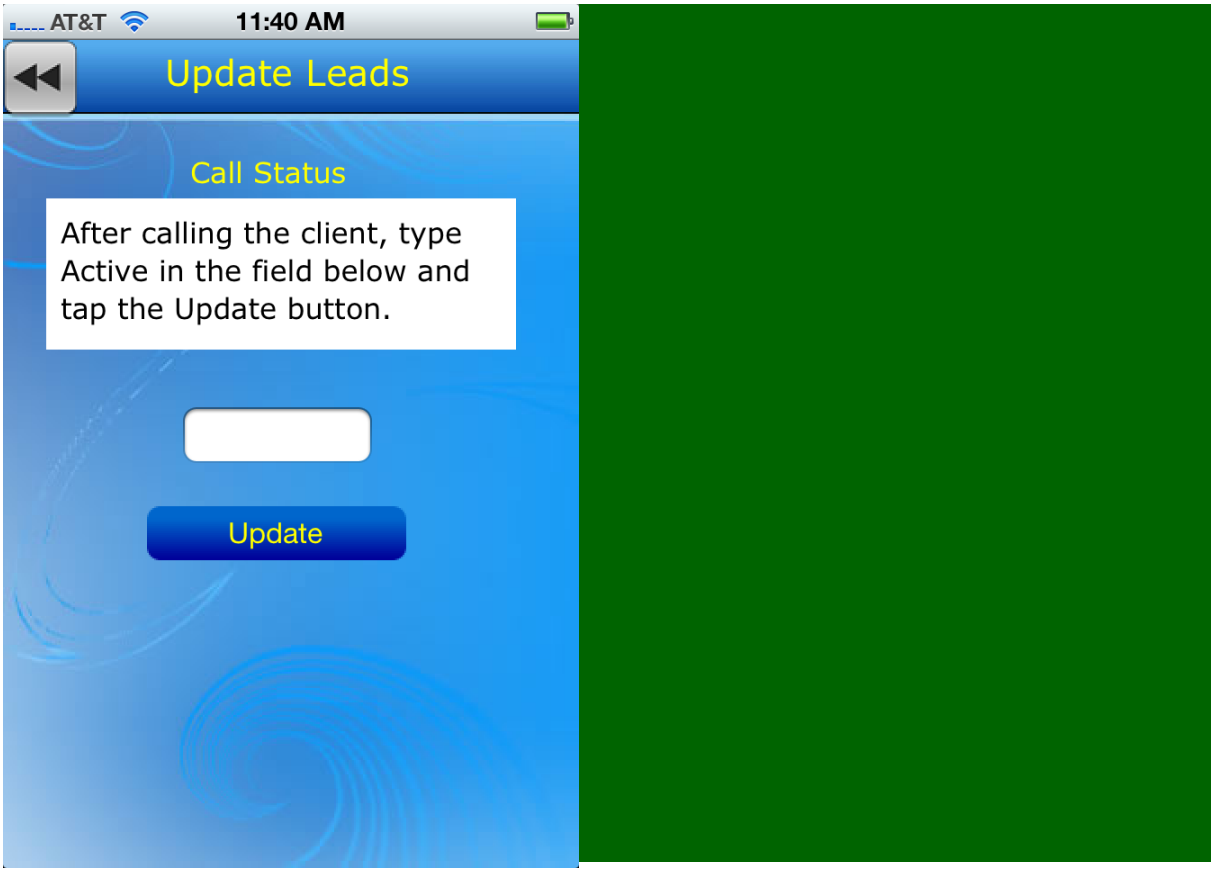
The app then checks to see if the account status for that username and pasword is *active*.

- If it status is active, the app then performs the Then statement and displays the list of events.
- If the status is not active, then app then performs the Else statement and displays the login_ failed error message.

Conditions On Commands

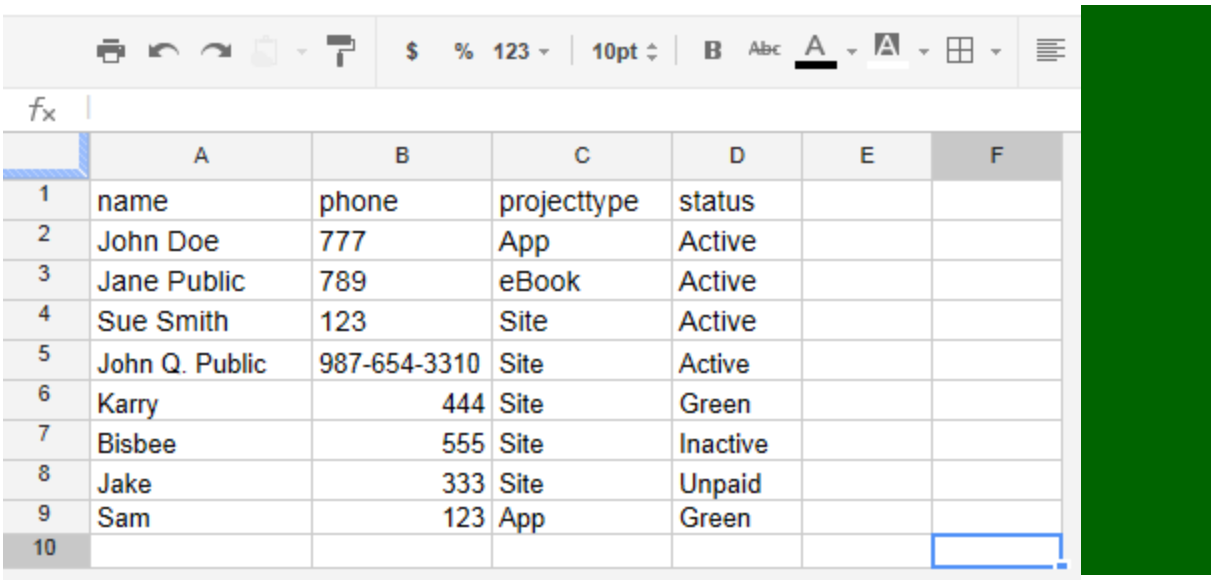
Inserting conditions after commands lets us specify the data on which to apply the commands. We'll base this discussion on the illustration in the Update Example topic and take it one step further. See "Update - Examples" on page 32

To recap, the update feature is controlled by the page shown below.



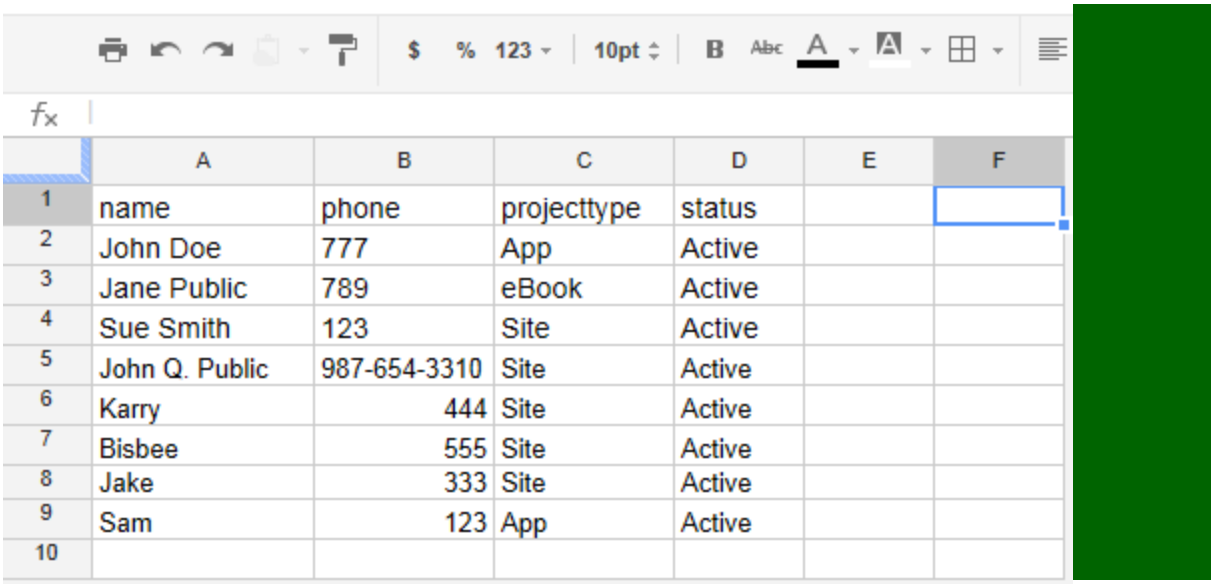
To update a spreadsheet field, the user types the new value in the field on the page above and taps the Update button. This replaces the contents of each specified field with the new value. In our example, it replaces the old value in a status field with the new value of Active.

Here's the spreadsheet before running the Update command.



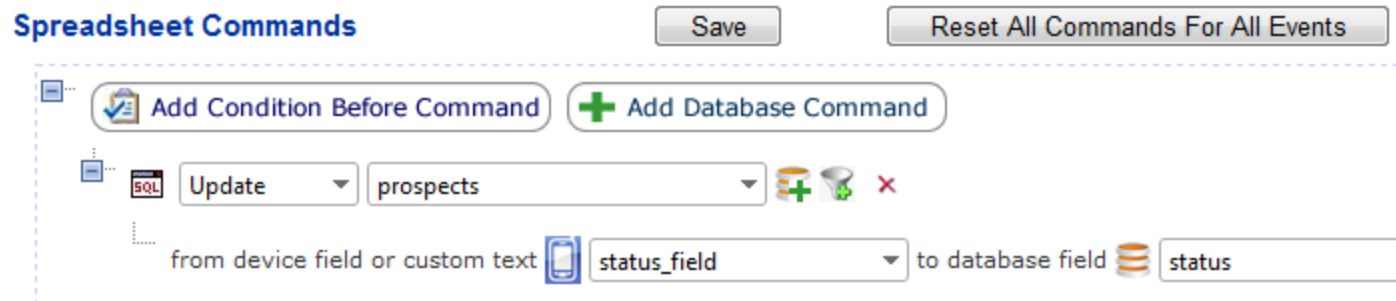
	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

And after running the command.



	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Active		
7	Bisbee	555	Site	Active		
8	Jake	333	Site	Active		
9	Sam	123	App	Active		
10						

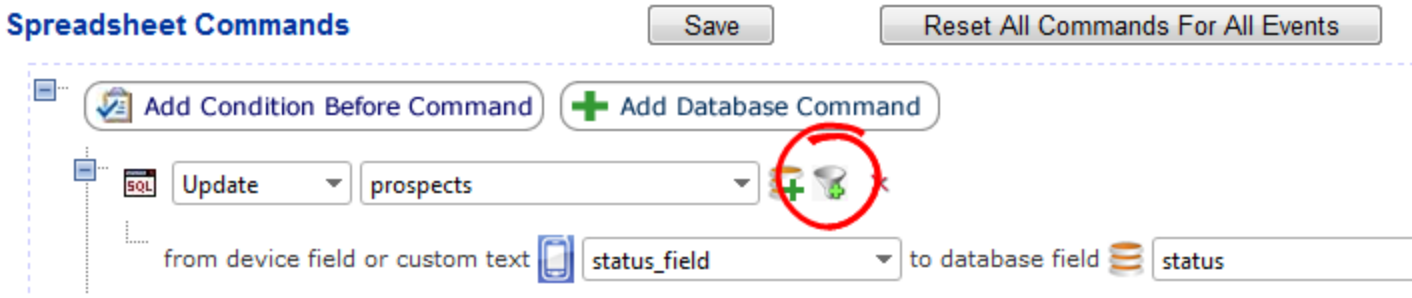
It worked, but the Update changed the value of all status fields to active. We want to change only specified fields. Let's say we want to change only values of Green to Active. We need a condition. (Here's the finished data management entry from the simple Update command. for reference.)



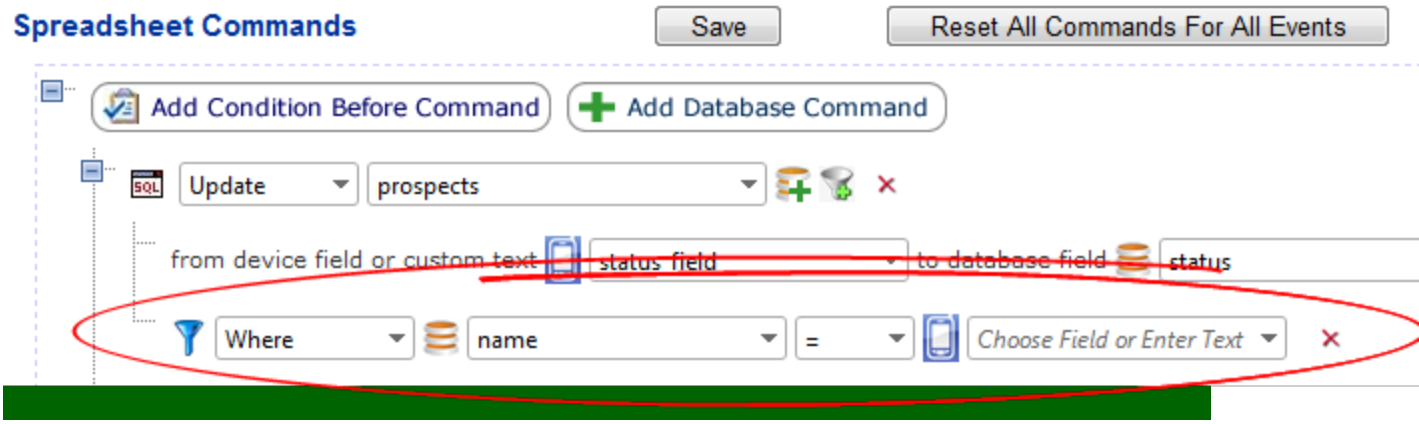
We need to specify that the command should only apply to cases where the status field value is

Green. Here's how to add the condition to the existing commands.

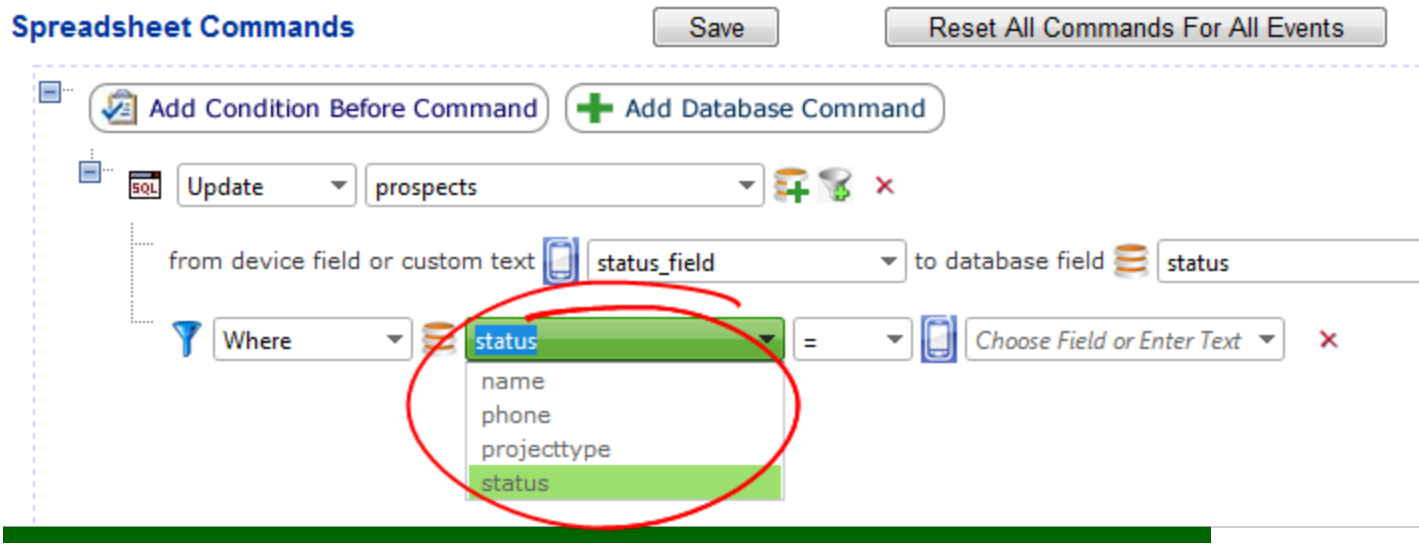
1. Click the Add a Where Condition button.



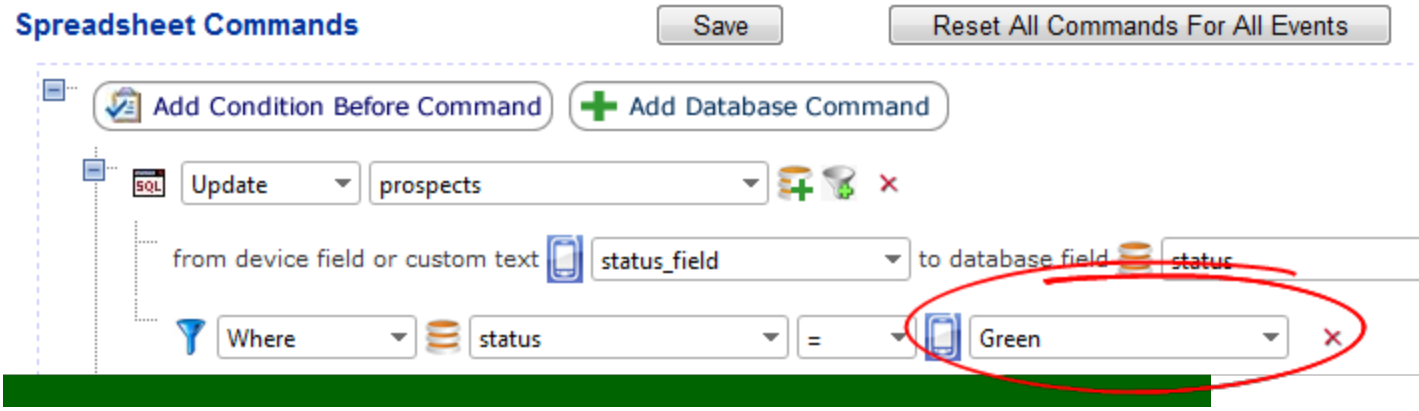
This adds the condition statement below the command, as shown below.



2. Leave the command option set to Where, click the database field pulldown, and select the field whose value will be the basis for the condition, in this case the status field, shown below.



3. Leave the arithmetic operator set to =. (You could select !=, <, or > depending on the condition you want to define.)
4. Type the value to check for in the device field, here the value of Green, shown below.



The result is the change of every entry whose status is Green to a status of Active. First the original spreadsheet.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Green		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

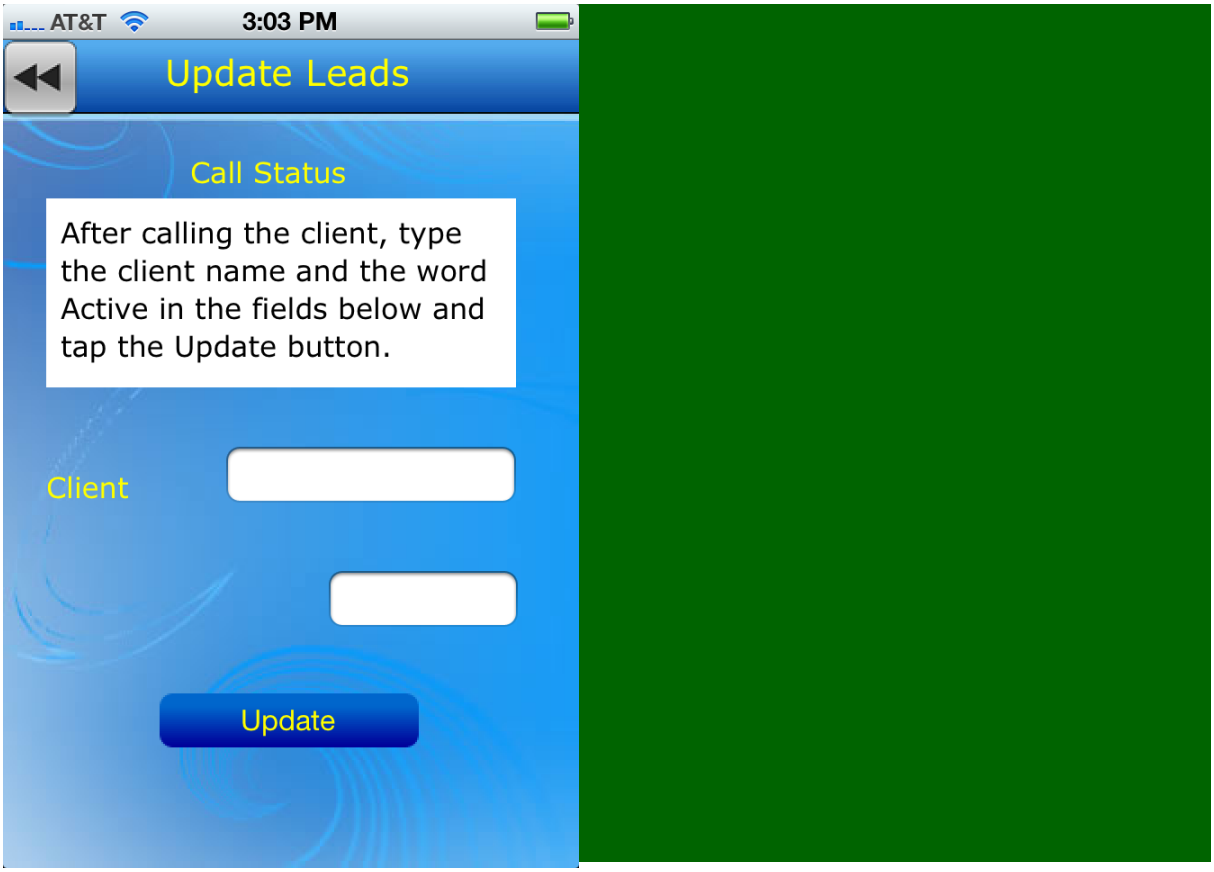
Then the changed spreadsheet.

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Active		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Active		
10						

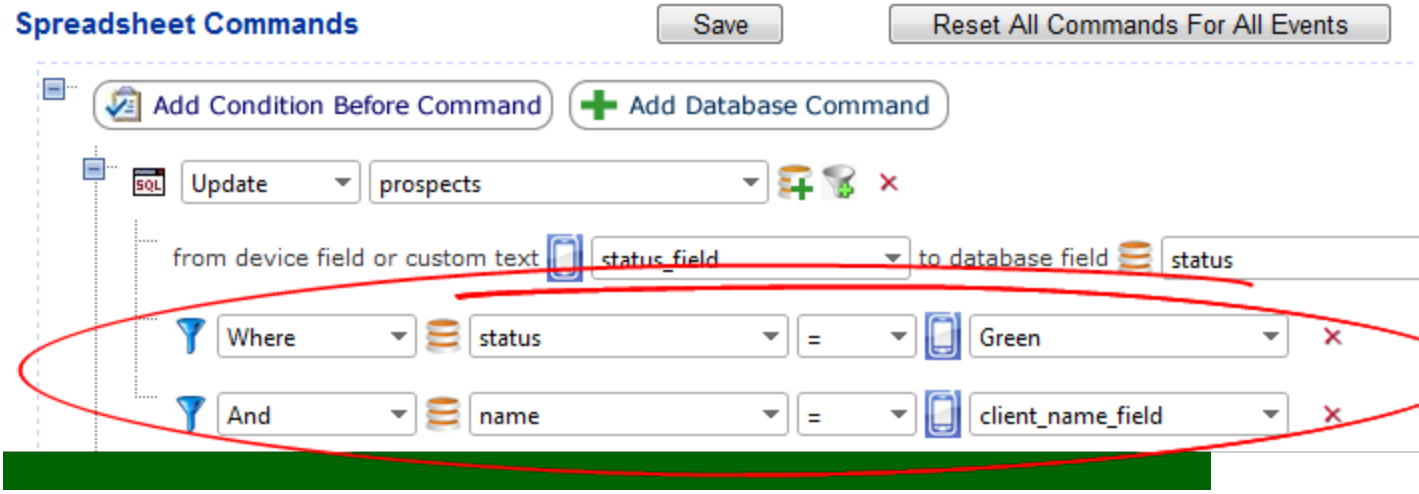
Karry and Sam have had their status changed from Green to Active.

What if you only want to change one entry at a time, such as being able to select and change customer Karry's status from Green to Active.

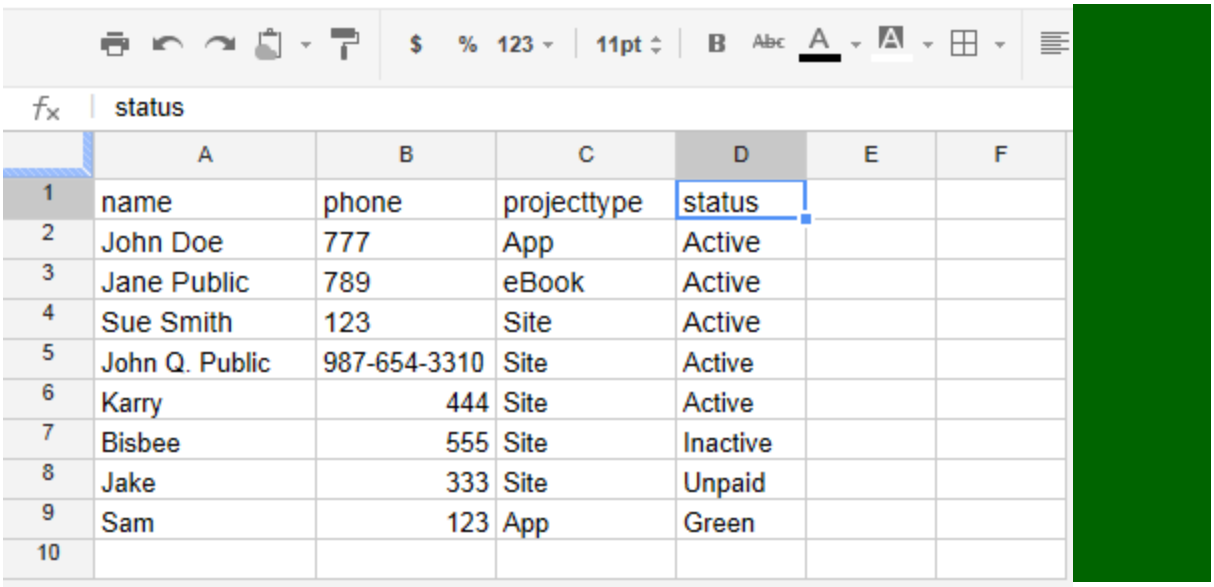
The process is similar to the one above, except that you need a second condition. You also need to add a field on the Update page to contain the value that the new condition will test for. So you'd change the Update page to something like this.



The code name for the Client field is client_name_field. We can now add a condition in order to be able to specify which entry's status to change based on the client name, like this.



The second condition, And, tells the app to change the value of the status field from Green to Active for the client whose name is specified in the client_name_field on the Update page. Specifying Karry in that field changes Karry's status value from Green to Active but leaves Sam's set to Green.



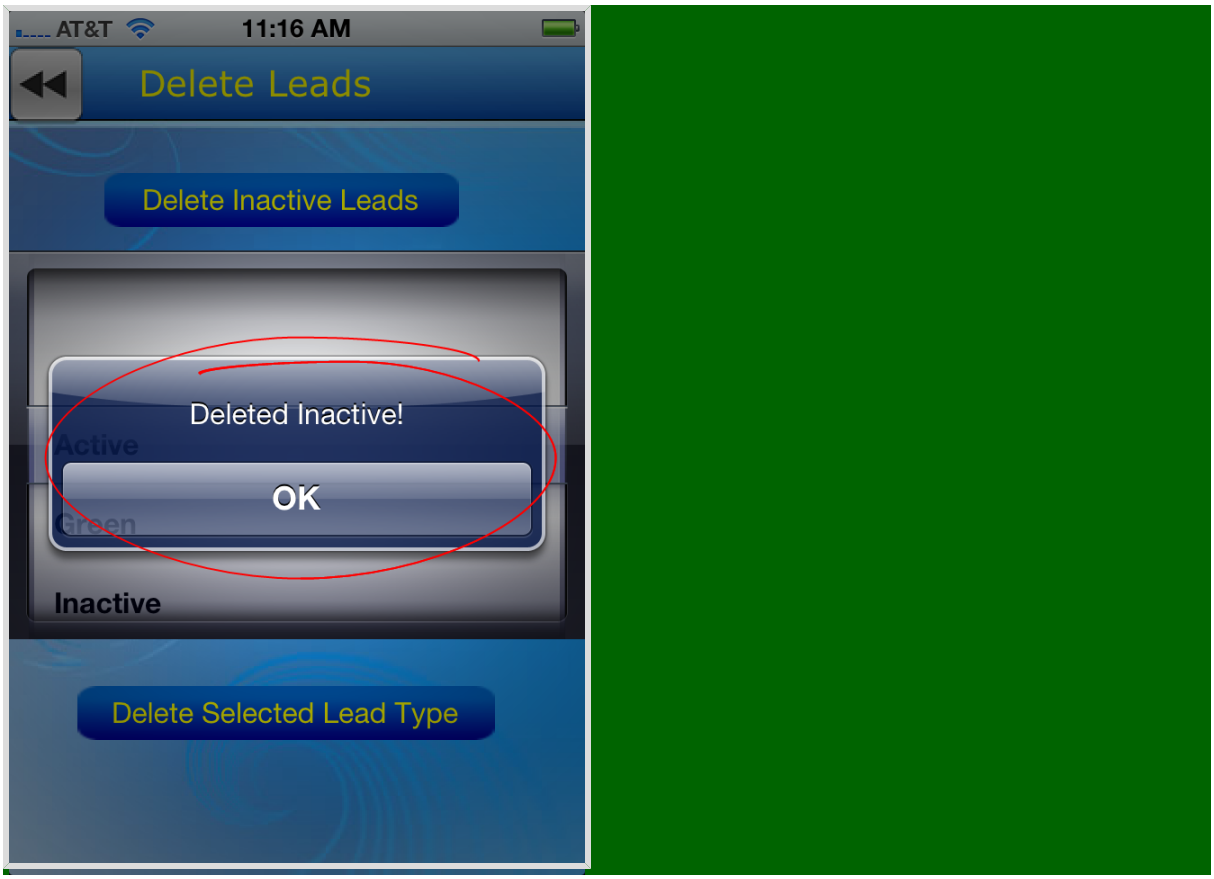
fx | status

	A	B	C	D	E	F
1	name	phone	projecttype	status		
2	John Doe	777	App	Active		
3	Jane Public	789	eBook	Active		
4	Sue Smith	123	Site	Active		
5	John Q. Public	987-654-3310	Site	Active		
6	Karry	444	Site	Active		
7	Bisbee	555	Site	Inactive		
8	Jake	333	Site	Unpaid		
9	Sam	123	App	Green		
10						

Feedback for Data Operations

If users take some action that affects data in the spreadsheet, they need feedback to tell them that the action worked.

Here's a simple feedback message that appears after a user deletes the spreadsheet entries marked as Inactive by tapping the Delete Inactive Leads button.

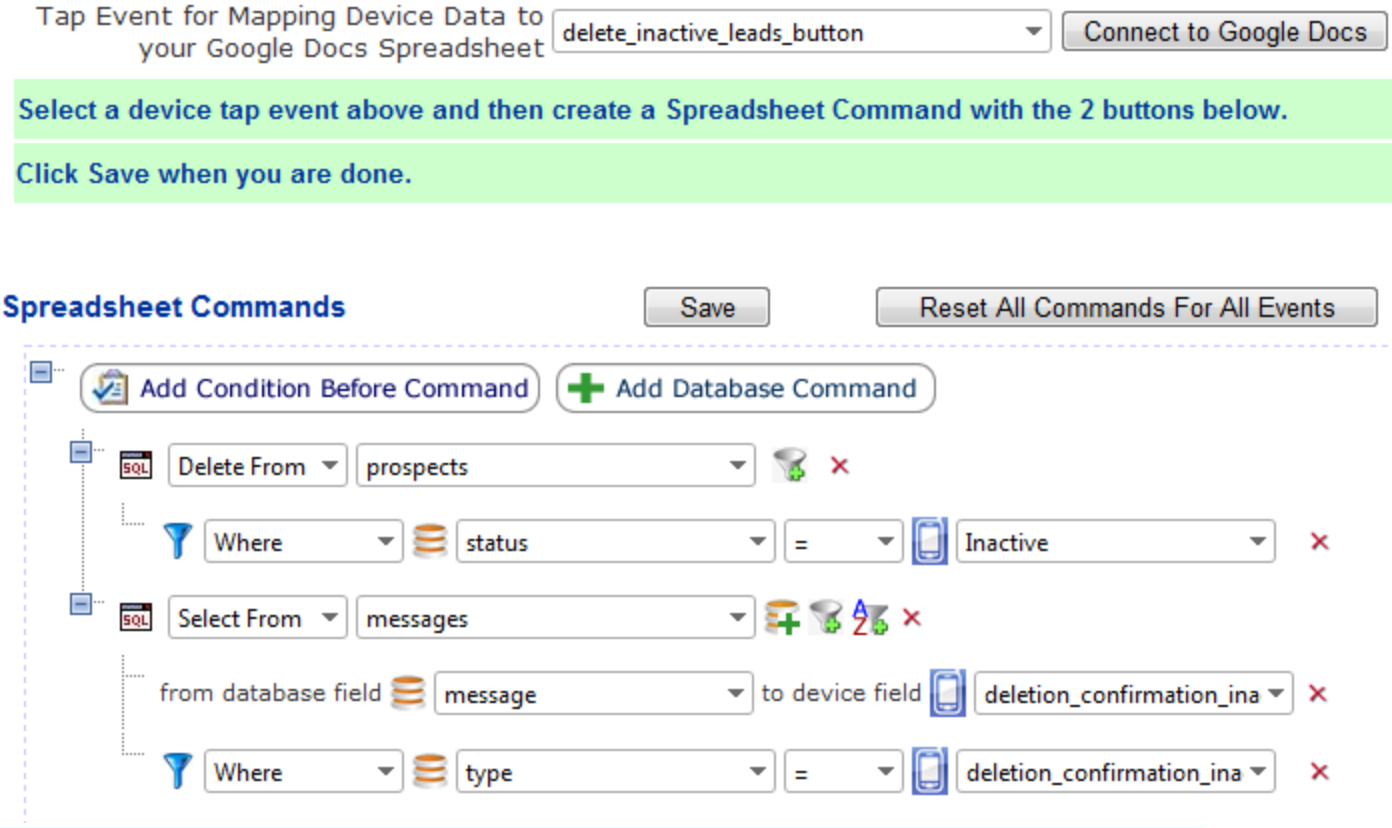


The feedback feature lets us specify the message, and automatically adds a tap option that closes the message. Here's how to add a message, specifically the one shown above.

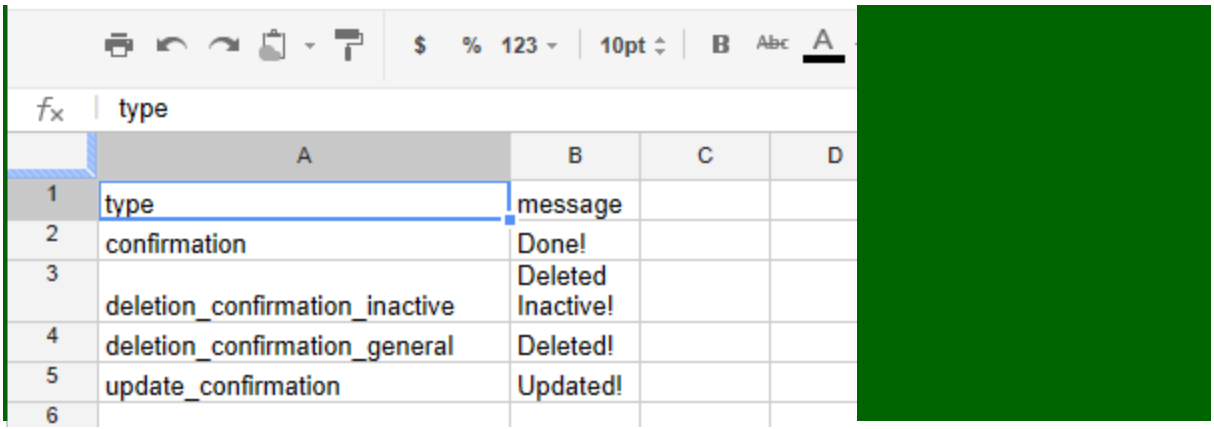
First, the app page has to include a ViziApps alert. This screen object is on the output page but hidden. Here's what the page looks like in ViziApps' development mode.

The two orange triangles with white exclamation points are alerts. The code name for the one on the left *is deletion_confirmation_inactive*, which displays when we hover over the alert.

Here's the complete set of data management commands for the Delete Inactive Leads button on the page shown above.



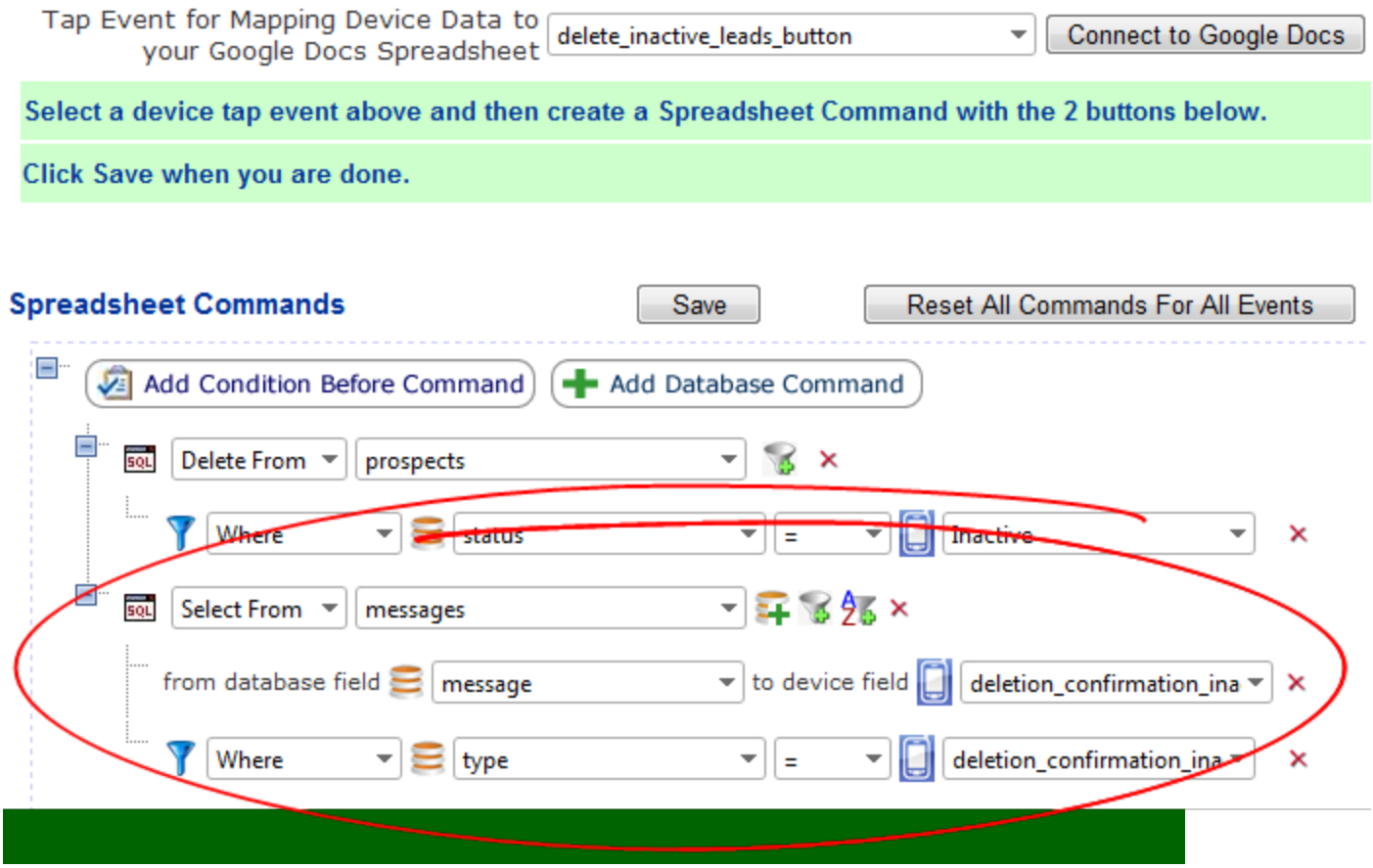
After completing the first command, the Delete From and its Where condition, we click the Add Database Command button again. We're going to use a different worksheet from the same spreadsheet, messages, and shown below.



fx type

	A	B	C	D
1	type	message		
2	confirmation	Done!		
3	deletion_confirmation_inactive	Deleted Inactive!		
4	deletion_confirmation_general	Deleted!		
5	update_confirmation	Updated!		
6				

This worksheet contains two columns - type and message - with one for each data activity. How does this work?

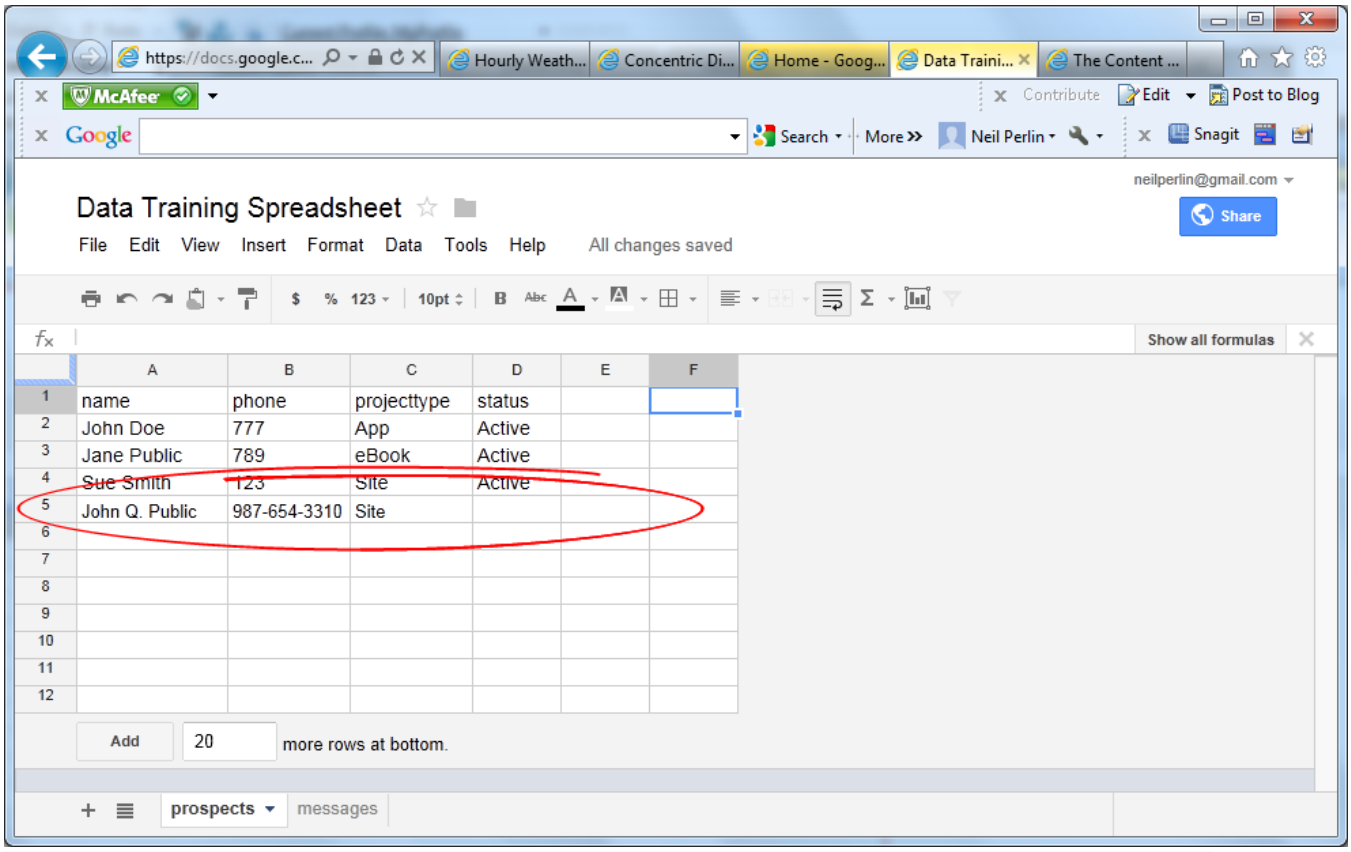


1. Click the Add Database Command button to insert another command.
2. We want to select a message for display on the mobile device, so we select the Select From command and the messages worksheet.
3. We now need to specify the database field that contains the message and the field on the mobile device in which to insert the message. On the worksheet example above , the messages are stored in a field called message. We want to insert this message in a field on the mobile device that's doesn't display until we put content, the alert, in it.
4. There are two alerts on the page, one to show the confirmation for deleting inactive entries and one to show the confirmation for deleting user-selected entries. We select the first one, *deletion_confirmation inactive*.
5. Finally, we need to specify which type of message to display. The type field on the spreadsheet contains one called *deletion_confirmation_inactive*, which is the one we want.

The result is to tell the app that after deleting the Inactive entries, the app should switch to a different worksheet, select the message whose type is *deletion_confirmation_inactive*, and display it in the hidden field, the alert, of the same name.

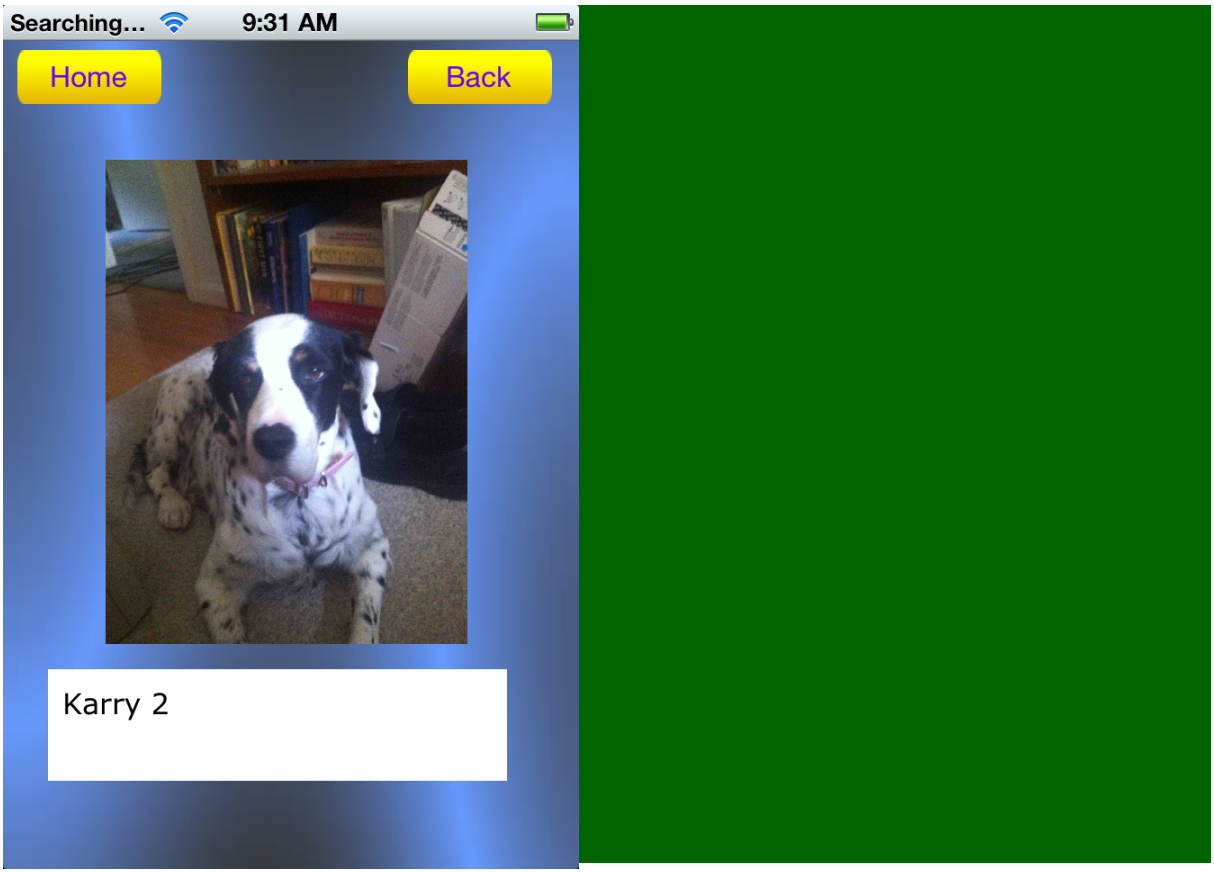
How Images Are Stored

Text or numeric entries in your app fields are stored as such in the spreadsheet. For example, John Q. Public entered in a *name* field and 978-654-3310 entered in a *phone* field display in the spreadsheet like the example below.

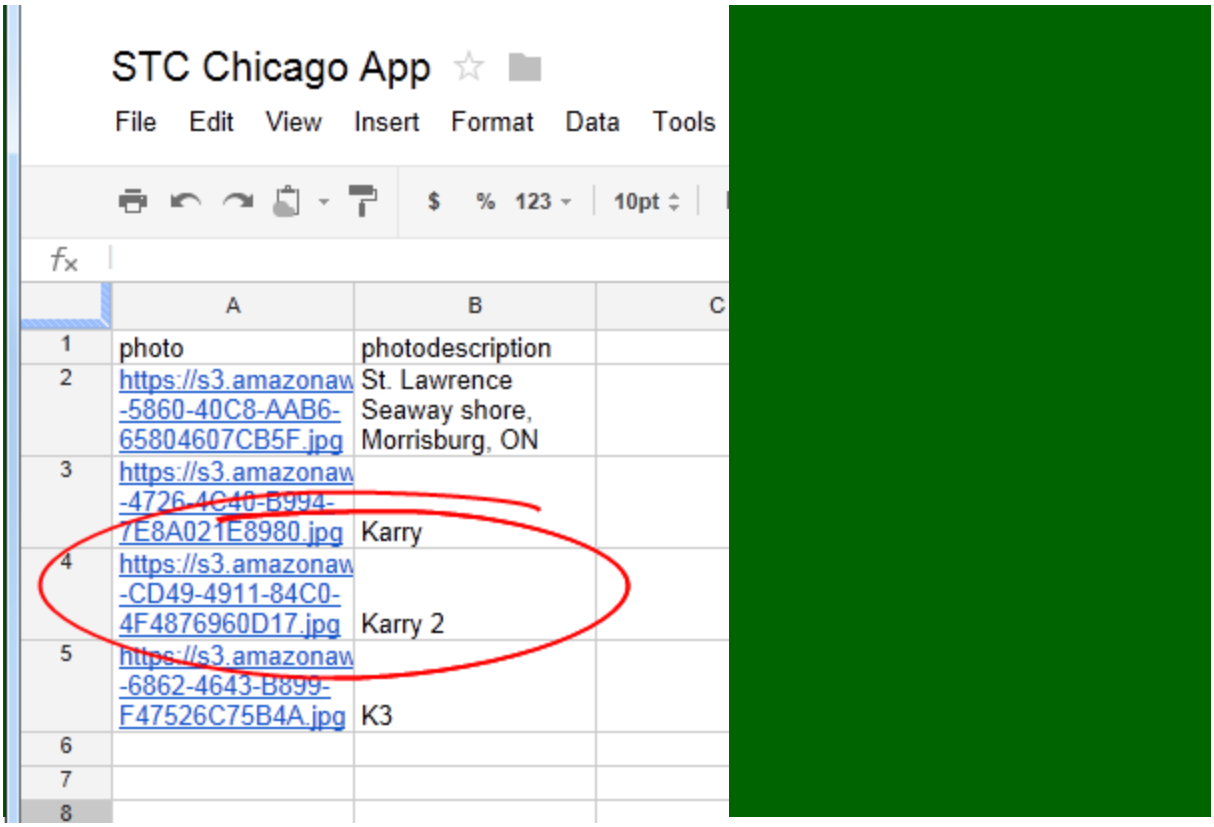


There are two exceptions to this, images and maps. This topic explains how images are stored. See "How Maps Are Stored" on page 67 for information about maps.

When you take a photo using the mobile device camera, the mobile device stores the image in the cloud and automatically enters the image's URL for storage in the spreadsheet. The image below shows an example, a photo and some descriptive text.



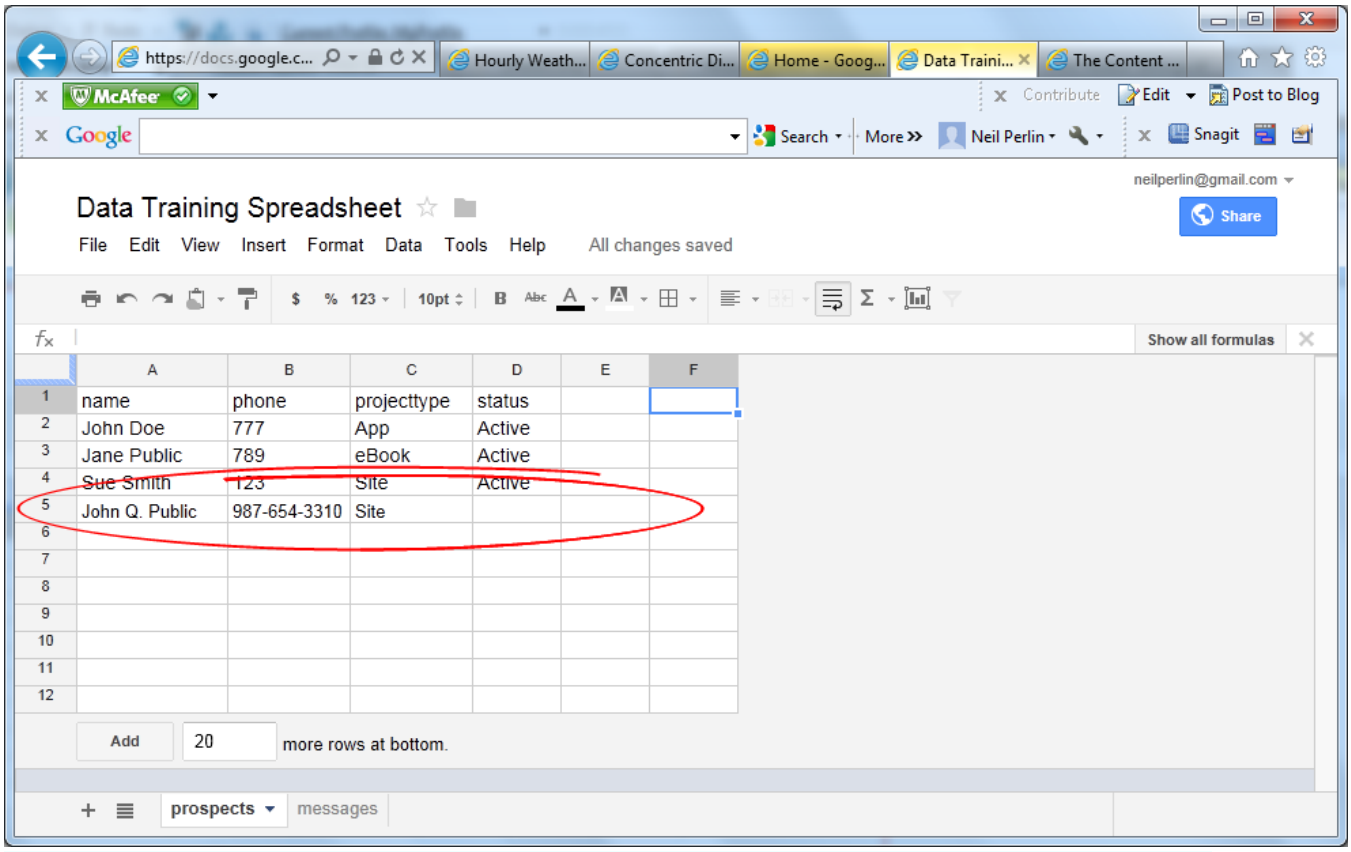
The spreadsheet entry looks like line 4 below. The field names are *photo* and *photodescription*. The field entry for the *photodescription* field is Karry 2, but for the *photo* field it's the image URL, not the image itself.



So Google Docs handles the programmatic aspects for you.

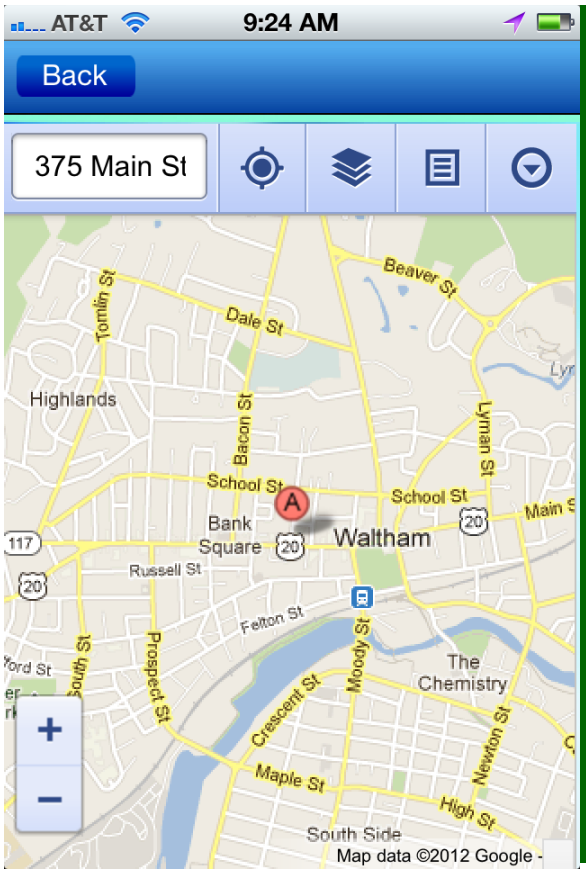
How Maps Are Stored

Text or numeric entries in mobile device fields are stored as such in the spreadsheet. For example, John Q. Public entered in a *name* field and 978-654-3310 entered in a *phone* field display in the spreadsheet like the example below.



There are two exceptions to this, images and maps. This topic explains how maps are stored. See "How Images Are Stored" on page 64 for information about images.

Maps are stored as URLs that contain address data and a Google Maps command to be processed by Google Maps. For example, the map shown below:

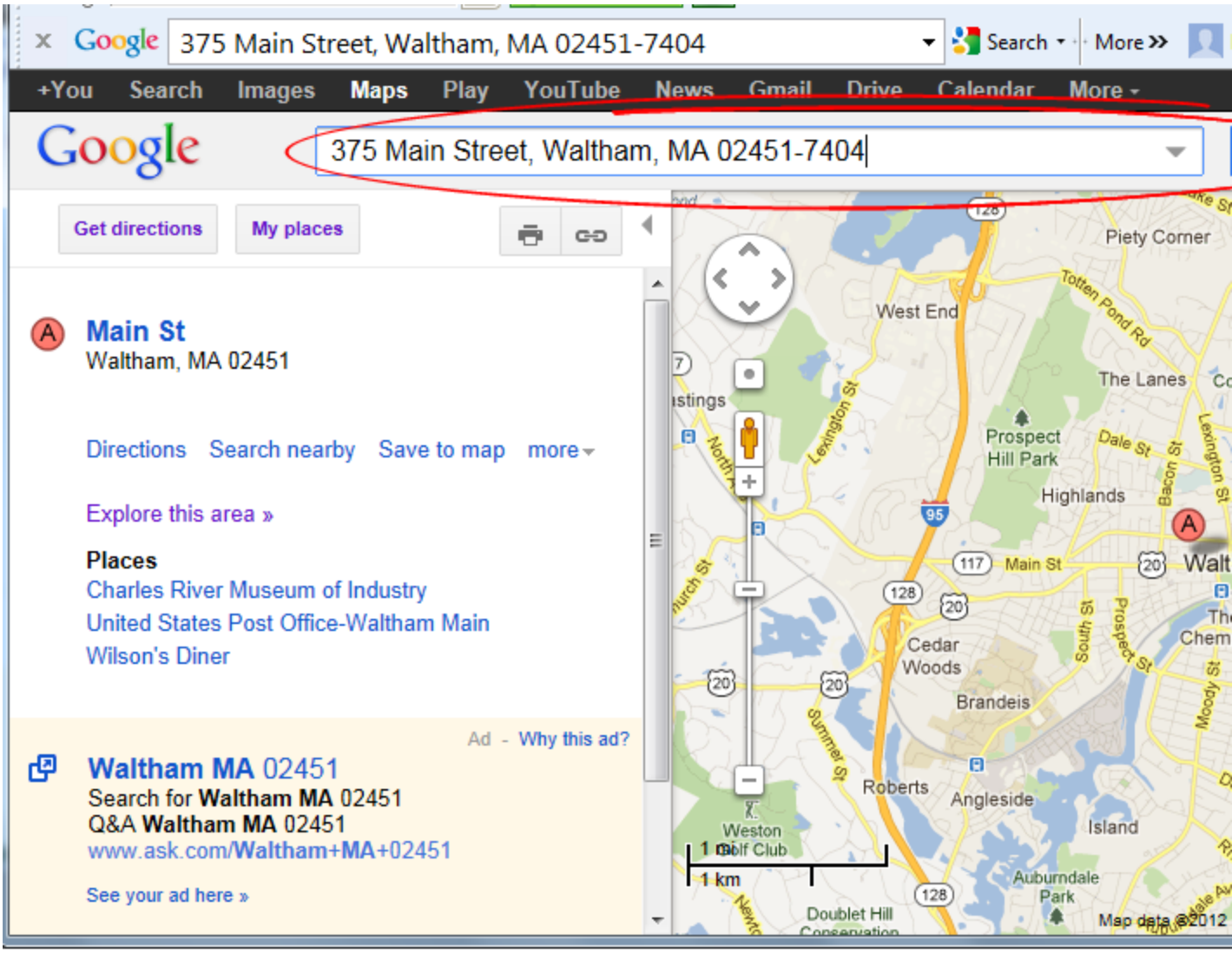


Is controlled by this entry in the spreadsheet - http://maps.google.com/maps?hl=en&q=375+Main+Street,+Waltham,+MA+02451-7404

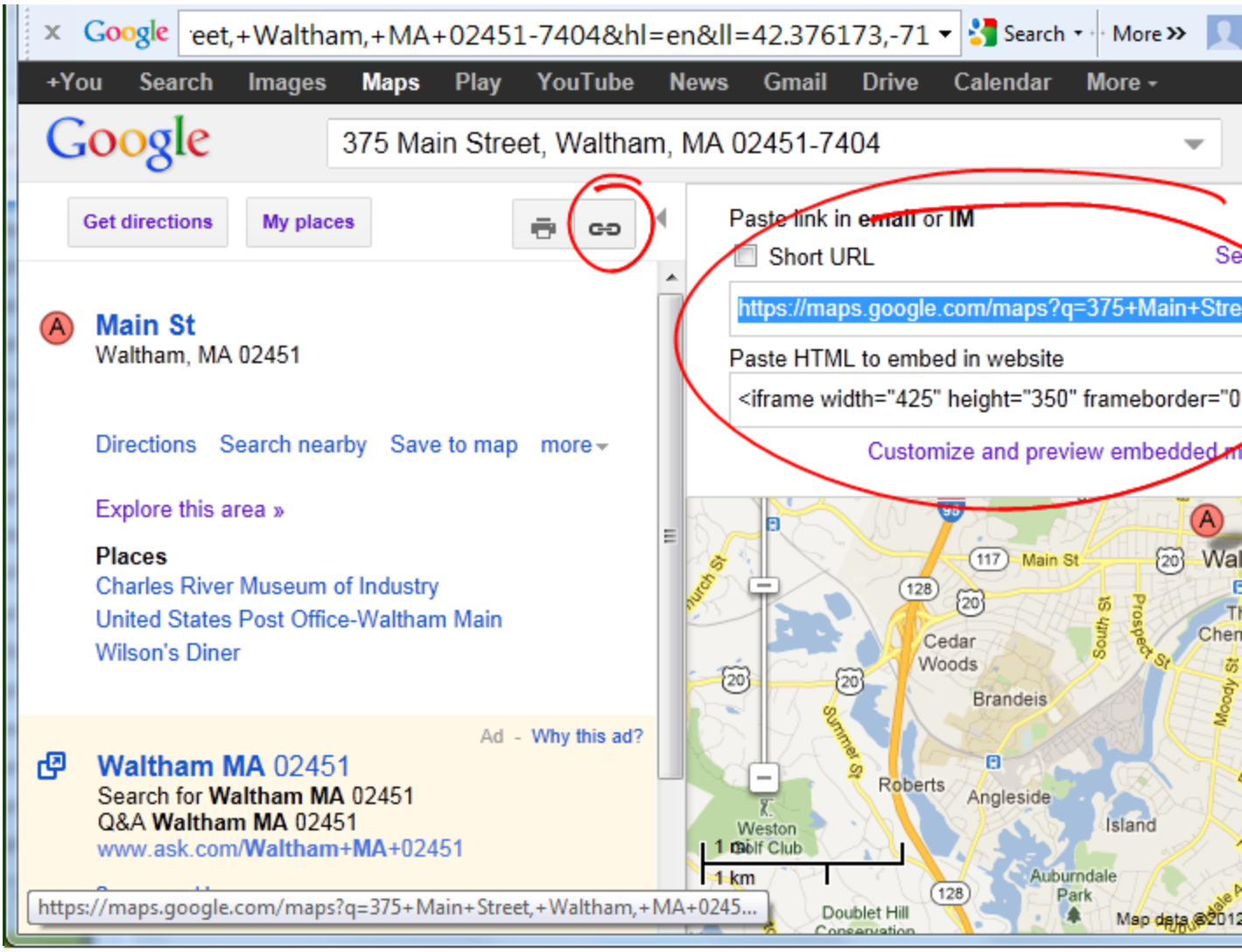
How do we create these entries? There are two ways, depending on how many you have to create and how familiar you are with spreadsheet commands.

Manually Creating the URL

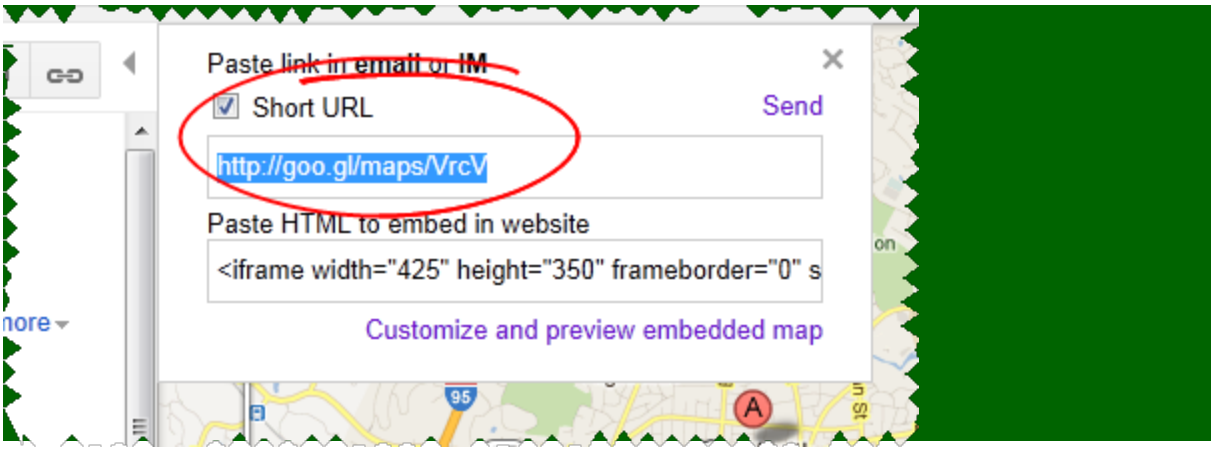
You can insert each address in the address box of a Google Maps screen, capture the resulting link URL, and insert it in the spreadsheet. This is slow but simple. For example, to get the URL for the map above, open Google Maps, type the address in the address field, and click the blue Search icon, as shown below.



Then click the Link icon, which opens a window displaying the URL for the location. The image below shows the Link icon and the URL window.



Copy the URL, highlighted and circled on the right in the image above, into your spreadsheet. You can generate a short URL instead by clicking the Short URL option, which gives the result shown in the image below.



Automatically Creating the URL

If you already have the address data in your spreadsheet, you can semi-automatically create URLs by using the spreadsheet's concatenate command to combine the different pieces into a URL. Here's the complete URL:

http://maps.google.com/maps?hl=en&q=375+Main+Street,+Waltham,+MA+02451-7404

and here are the sections:

- http://maps.google.com/maps? - The Google Maps command. Include it as is.
- hl+en - Specifies the host language. en = English.
- & - Separates two parameters. Use as shown.
- q - The address, as if it were entered in the http://maps.google.com page.
- + - The symbol for a space

So the first section of the URL - http://maps.google.com/maps?hl=en&q= - can simply be a constant value. This is more complex than the manual method but much faster.

Best Practices for Using Spreadsheets

As you create apps, you'll probably derive some of your own best practices. Here are some generic ones that apply to almost any app.

Planning and Design

- Don't put sensitive data like credit card numbers in a Google Docs spreadsheet. If you need to work with sensitive data, use Google Docs *within* Google Apps for Business (See "Google Apps for Business" on page 73) or use ViziApps' SQL database or web service options.
- Consider the client as you set up your spreadsheet and worksheets in order to:
  - Limit the overall maintenance and updating work that the client has to do.
  - Limit the maintenance and updating work to as few worksheets as possible.

  The goal is to simplify the work to be done by clients for whom app data maintenance is a small part of their work and something with which they're not very familiar or comfortable.

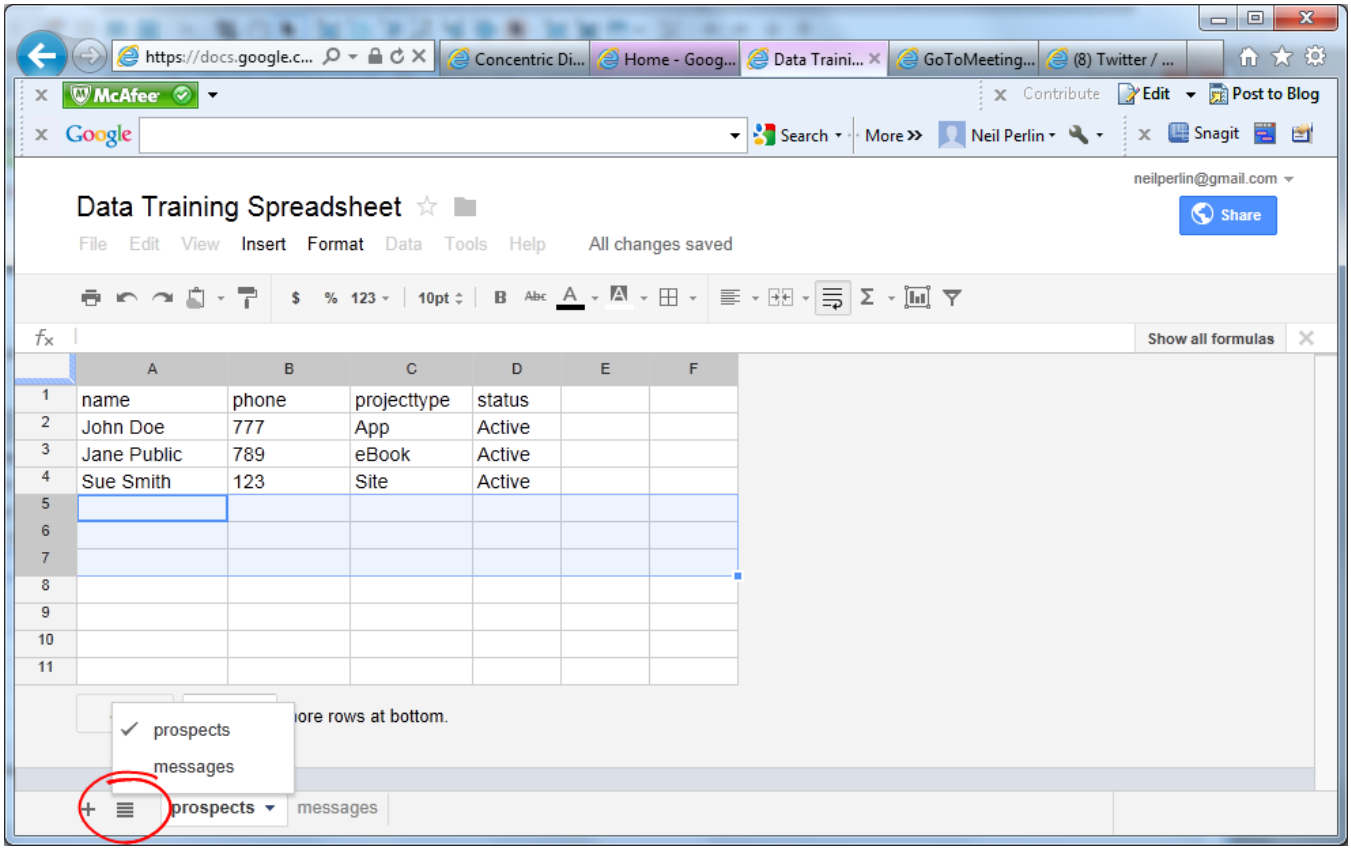
Development

- Give worksheets descriptive names like "prospects". Don't use vague names like “sheet1”.
- Make sure spreadsheet field names can be easily associated with device field names. For example, if users will enter a prospect's name in a field on the device in order to pass that entry to a field in the spreadsheet, name the device field something like *prospect_name_field* and the spreadsheet field something like *prospect_name*.

  Why *prospect_name_field* for the device field name rather than *prospect_name*? Every object on an app page needs an internal code name for reference. A field on the device, such as prospect name field, may actually consist of two objects, the field label and the entry field. You can't have duplicate names so you'd have to name the objects something like *prospect_name_label* and *prospect_name_field*.

  If you find it confusing or cumbersome to refer to the *prospect_name_field* field, you could simply refer to the two objects as *prospect_name_label* and *prospect_name*, with the understanding that the latter refers to the actual entry field. It's your preference.

- If you import an Excel spreadsheet into Google Docs, Google Docs may add an extra, "phantom" worksheet that can block ViziApps' access to the data. If you get an error message about a worksheet that you didn't create and don't see when you connect to the Google Docs spreadsheet, click the All Sheets button at the bottom left of the spreadsheet to see a list of all the worksheets.



The phantom worksheet may be listed there. This is an old and rare problem but it may recur. If you plan to import your data by importing an Excel spreadsheet into Google Docs, connect the app to the spreadsheet first, using the Connect to Google Docs button on ViziApps' Data Management page. If the error occurs, you can work around it by copying and pasting the data from the Excel spreadsheet to the Google Docs spreadsheet rather than importing the Excel spreadsheet itself.

Google Apps for Business

ViziApps recommends that you do not use the standard Google Drive spreadsheet when developing your app because Google Drive does not support the high request volumes that your app might require. Instead, ViziApps recommends that you use Google Apps for Business which does allow high request volumes, as well as provides high security for access.

Note that you *can* use the standard Google Drive spreadsheet for experimentation in developing apps, but there is a bug in Google Drive as of Q2 2012. Users who log into Google Drive may occasionally get a "captcha required" message with no way to enter the captcha. ("captcha" is the acronym for Completely Automated Public Turing test to tell Computers and Humans Apart. They're the tests that ask users to read and type a visually distorted word before getting access to a web page. The theory is that a computer can't read a captcha, thus blocking internet bots.)

Google Apps for Business is a service from Google that provides a cloud-based collection of business software. It includes Gmail, Google calendar, and various other applications and tools - among them, for the purpose of ViziApps and this discussion, is Google Docs. Data - calendar entries, documents, spreadsheets, presentations, etc. - is no longer tied to individual PCs but is instead in the cloud. The benefit of using Google Apps for Business is that it offers the same simplicity and ease of use as Google Docs but with more features, power, and security.

In more detail, what does Google Apps for Business offer?

Availability

All your data is saved in the cloud automatically. This has several major benefits.

- The data is always available to any user, anywhere, on any device with web access that supports the cloud. No need to call the office to have someone find and send you a file.
- The data is on the cloud and backed up, reducing the chances of lost data due to a technical problem on a PC.
- The cloud model makes collaboration among employees easy. No more need to constantly send the latest copy of a document to all participants as an email attachment. The latest version of the document is in the cloud and accessible to everyone.

"Subcontracted IT"

Adopting Google Apps for Business is equivalent to subcontracting out your IT to a third-party. No need to buy servers, update software, do backups, etc.

Security

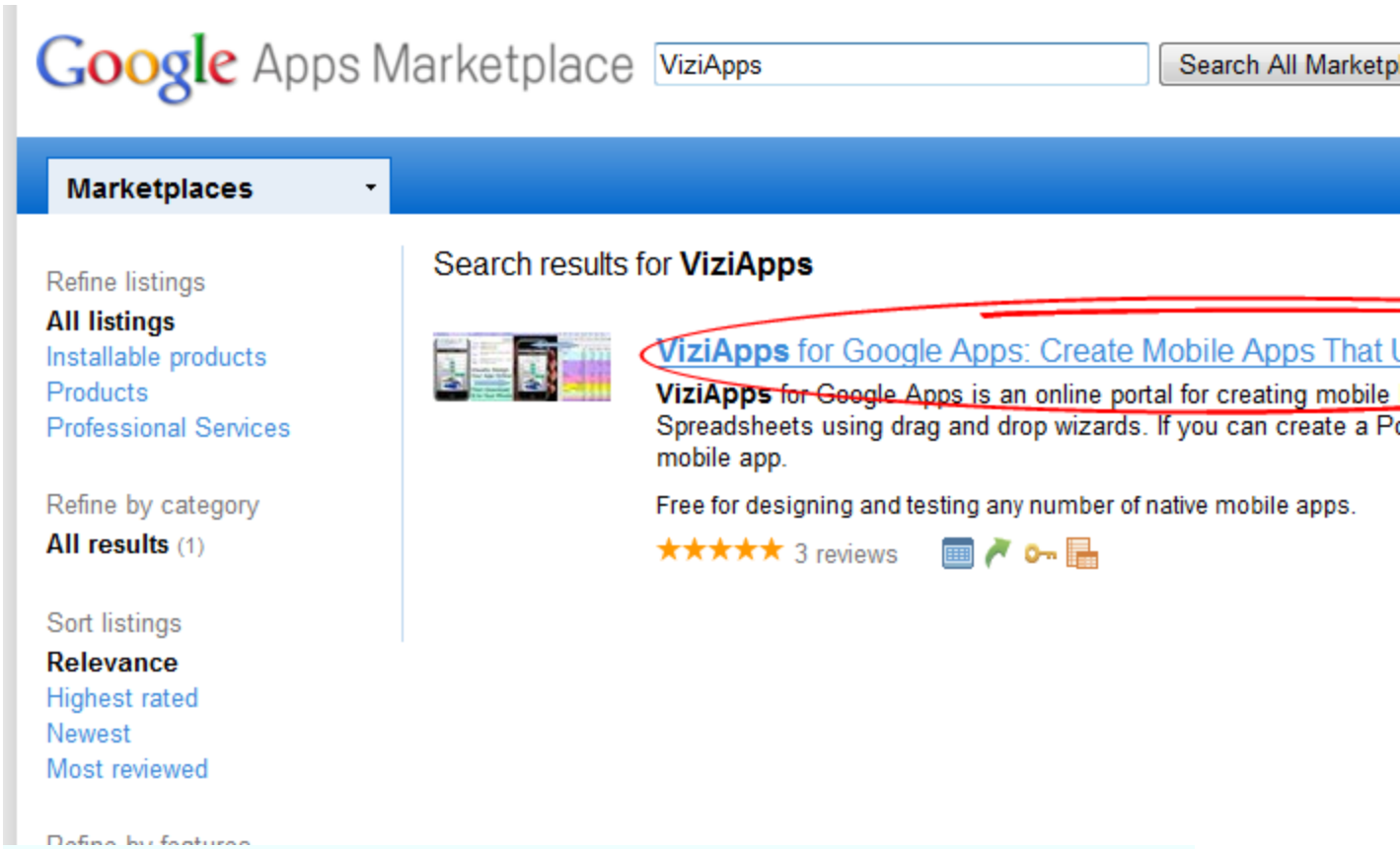
Google places great emphasis on the security that it uses to safeguard data in the cloud including, as Google says on the Google Apps for Business page at http://www.google.com/enterprise/apps/business/benefits.html:

- You own and control your data, not Google.
- Your data is constantly backed up.
- High security and reliability, with a guaranteed 99.9% uptime.
- Encryption and authentication of data.
- Single sign-in to Google Apps simplifies doing business in the cloud . You only need to remember one username and password rather than many.

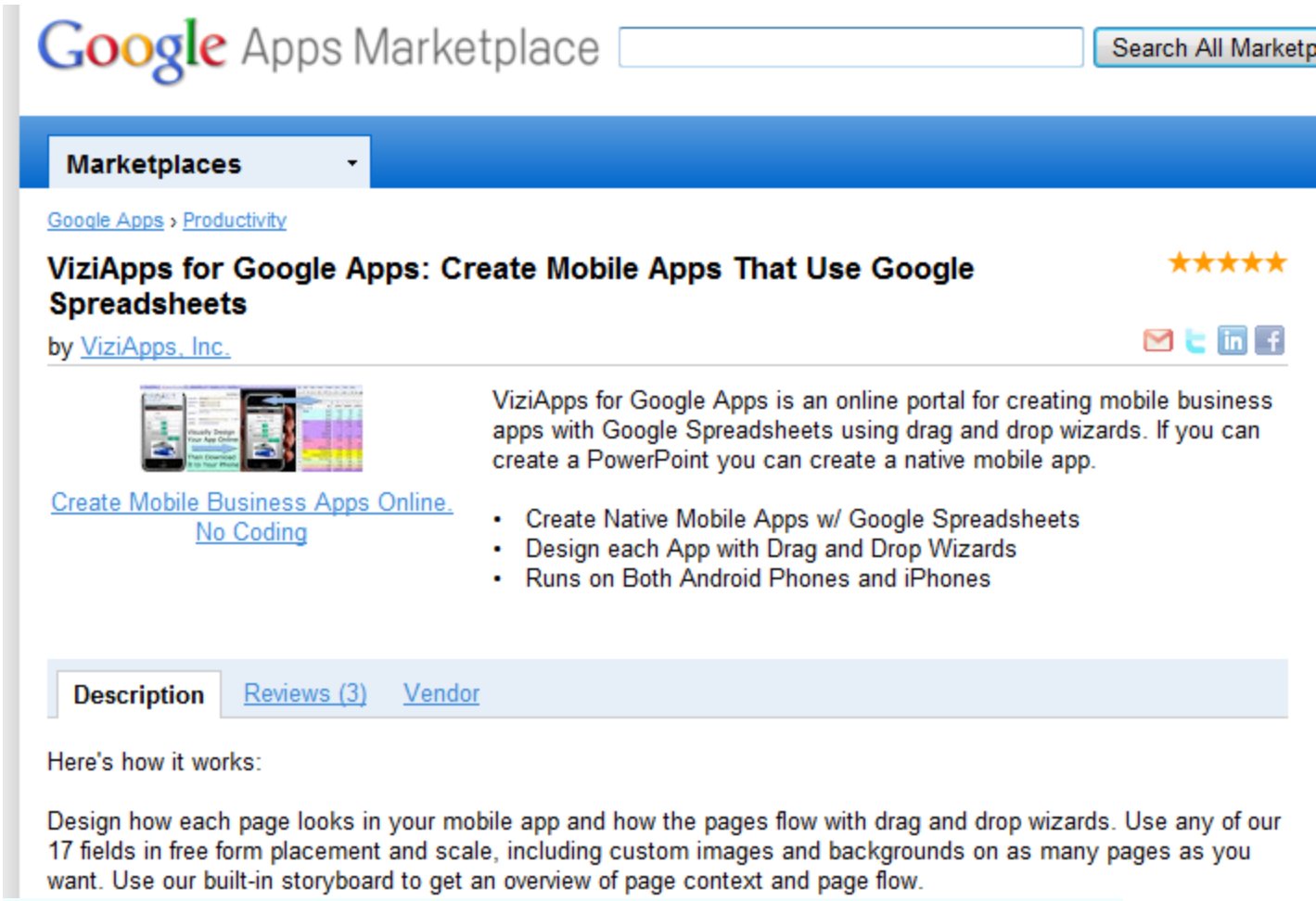
Accessing ViziApps Through Google Apps for Business

Follow these steps:

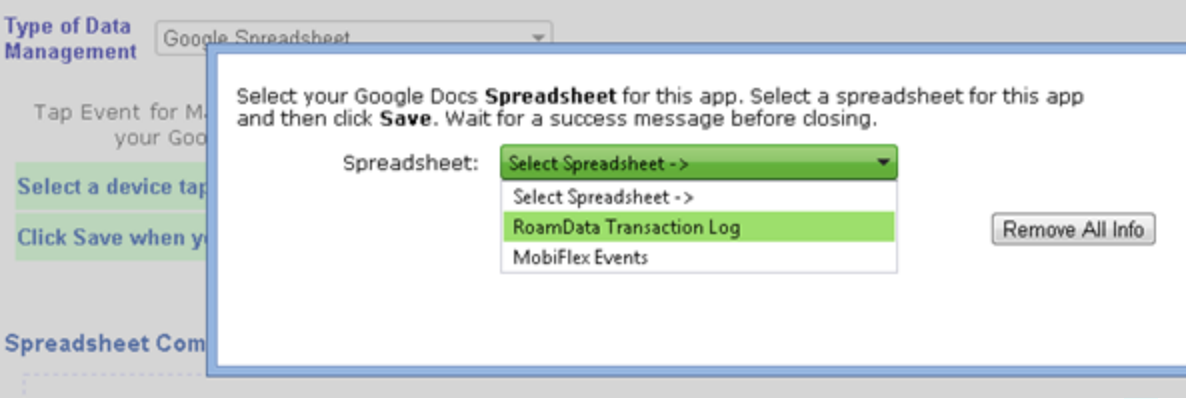
1. Login into your Google Apps Business account with your domain name and go to the Docs. (If you need to create a Google Apps for Business account, go to http://www.google.com/enterprise/apps/business/index.html.)
2. Go to Google Apps Marketplace at https://www.google.com/enterprise/marketplace/
3. Search for ViziApps in the search field at the top of the page. You should get one hit, shown below.



4. Select the link. You'll see a new page that lets you add ViziApps, shown below.



5. Review the description, then click the Add It Now button in the upper right corner of the screen to give ViziApps access to your documents.
6. Click More in the Google Docs menu and select ViziApps for Google Apps. ViziApps opens.
7. Login. After you do, the steps for using ViziApps for the same as described elsewhere in the help topics except that you don't have to log in on the ViziApps Data Management screen to select a database. Instead, when you click the Connect to Google Docs button, you'll see this connection dialog box.

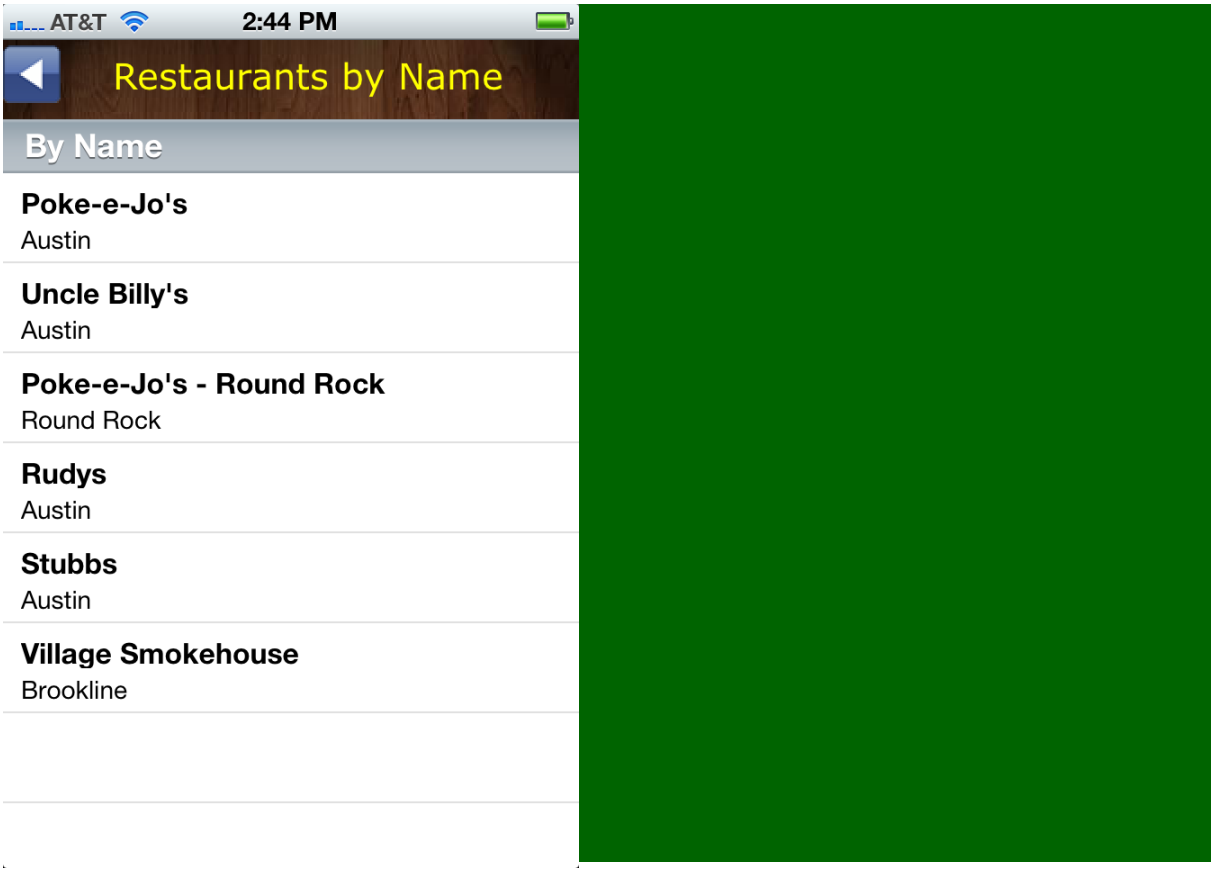


This dialog box doesn't ask you to login, since you already have.

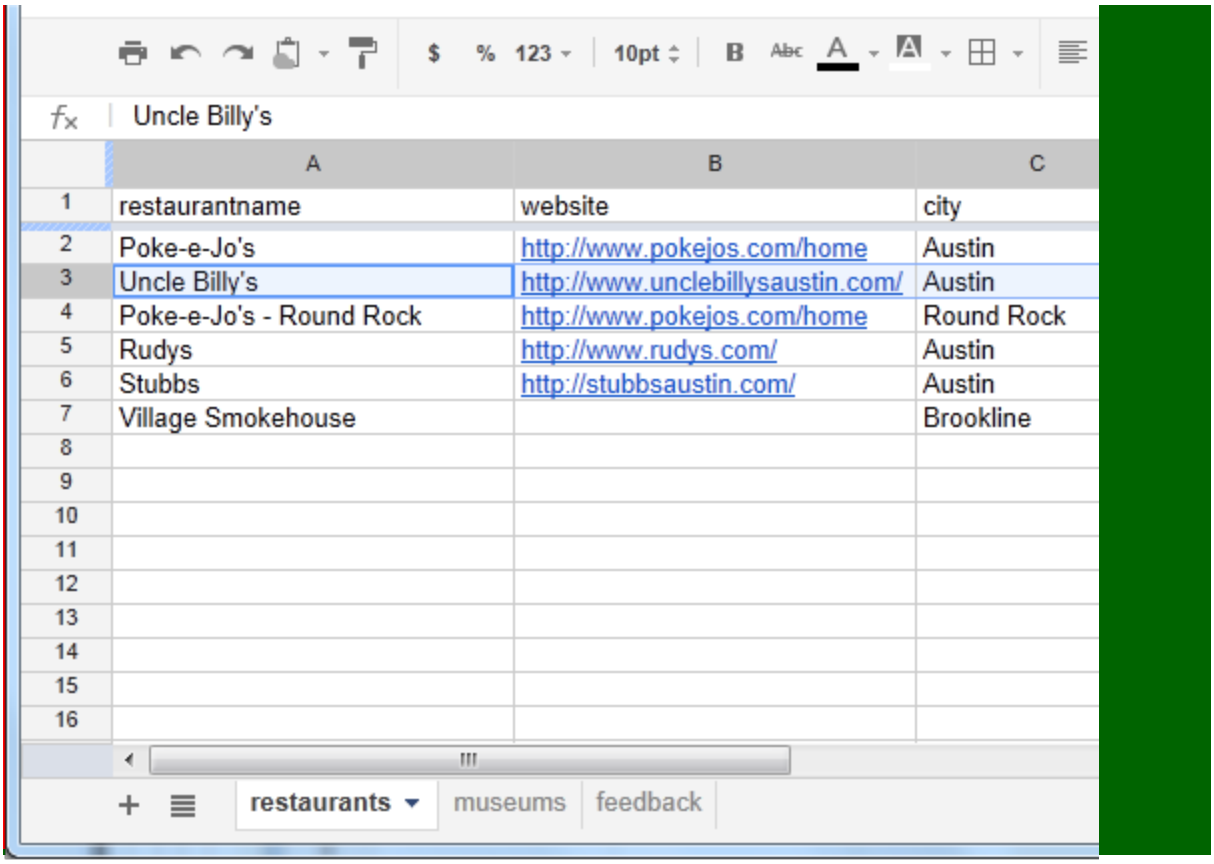
Example - Sorting Data In a Table

The example of user-added entries in apps ended by noting the problem that table entries will appear in the order in which the entries are made. (See "Example - User-Added Text" on page 90) This will rarely be alphabetical, which makes it difficult to find an entry in a long list.

For example, the figure below shows the list of restaurants in the Austin, TX, barbecue app.



The list isn't alphabetized. It shows the entries as they were added to the spreadsheet, shown below.

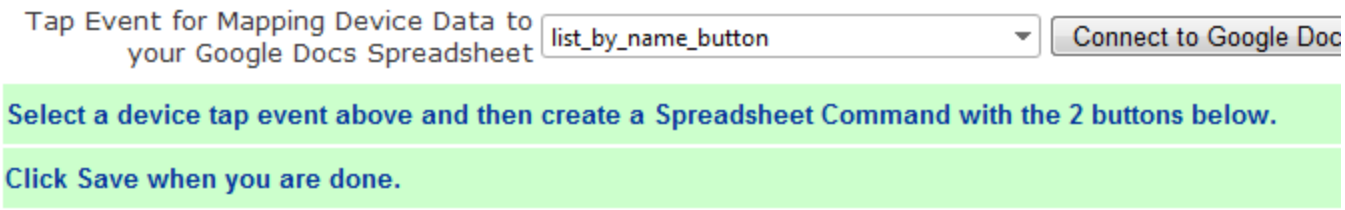


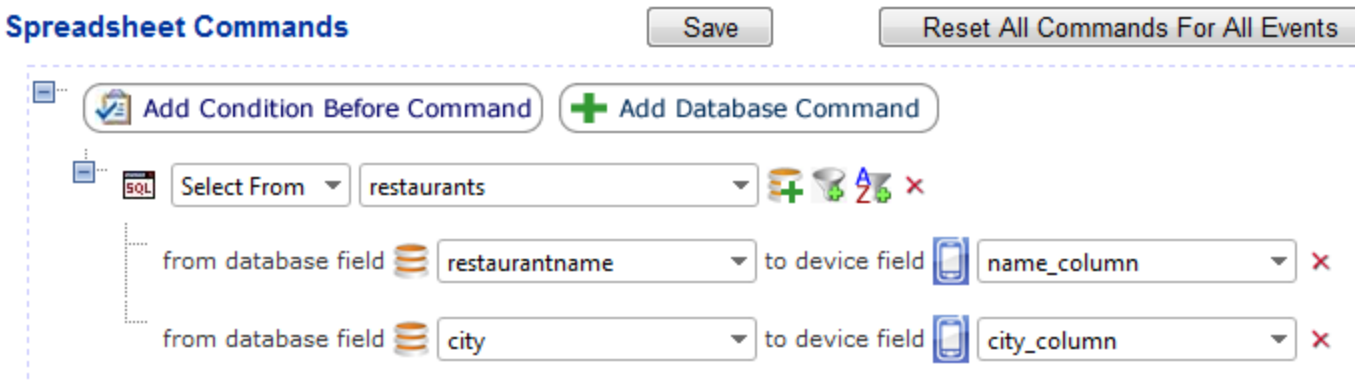
This list is so short that it's easy to find entries even if they're not in alphabetical order, but it would be more convenient if they were. How can we get the entries in alphabetical order?

- Alphabetize the entries in the spreadsheet. However, this is inefficient since it requires sorting the spreadsheet every time a new entry gets added.
- Use the sort feature in ViziApps' data management to dynamically present the entries in alphabetical order when they're displayed in the app, no matter what their order is in the spreadsheet.

The second approach is obviously the best. Let's see how to do it by taking the data management commands for the barbecue app one step further.

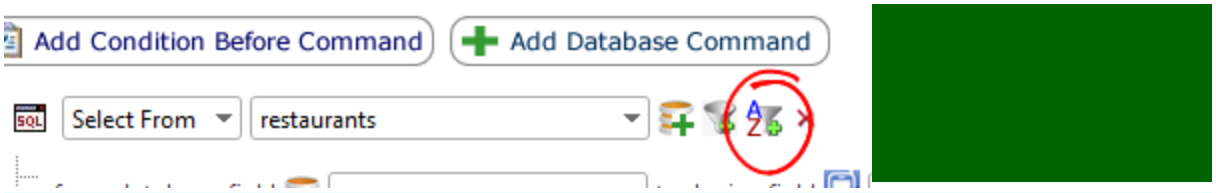
Here are the data management commands for the listing by name:



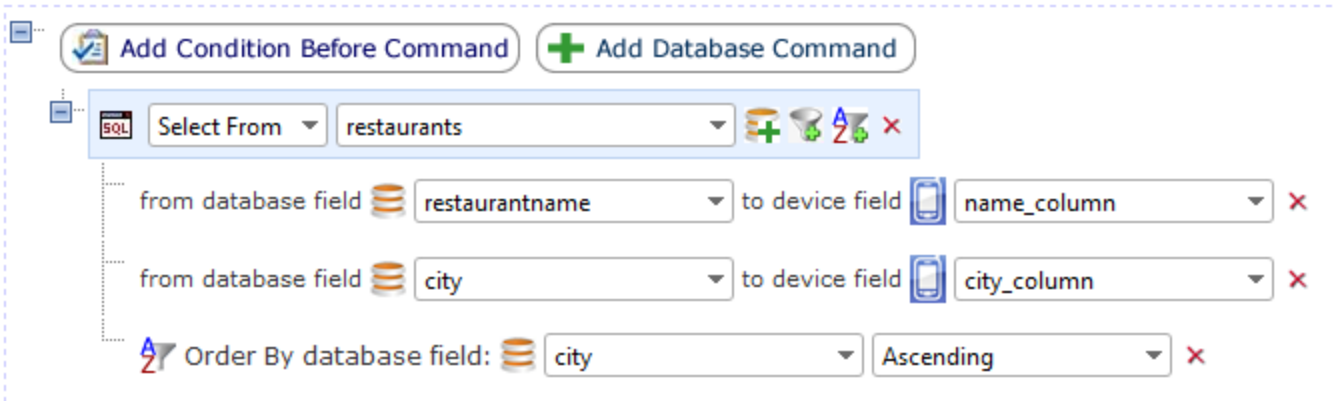


We need to add a sort function so that when the user taps the *list_by_name_button* button, the app reads the entries in the *restaurantname* and *city* fields on the *restaurants* worksheet *and* automatically displays them in alphabetical order in the *name_column* and *city_column* fields on the mobile device. Here's how.

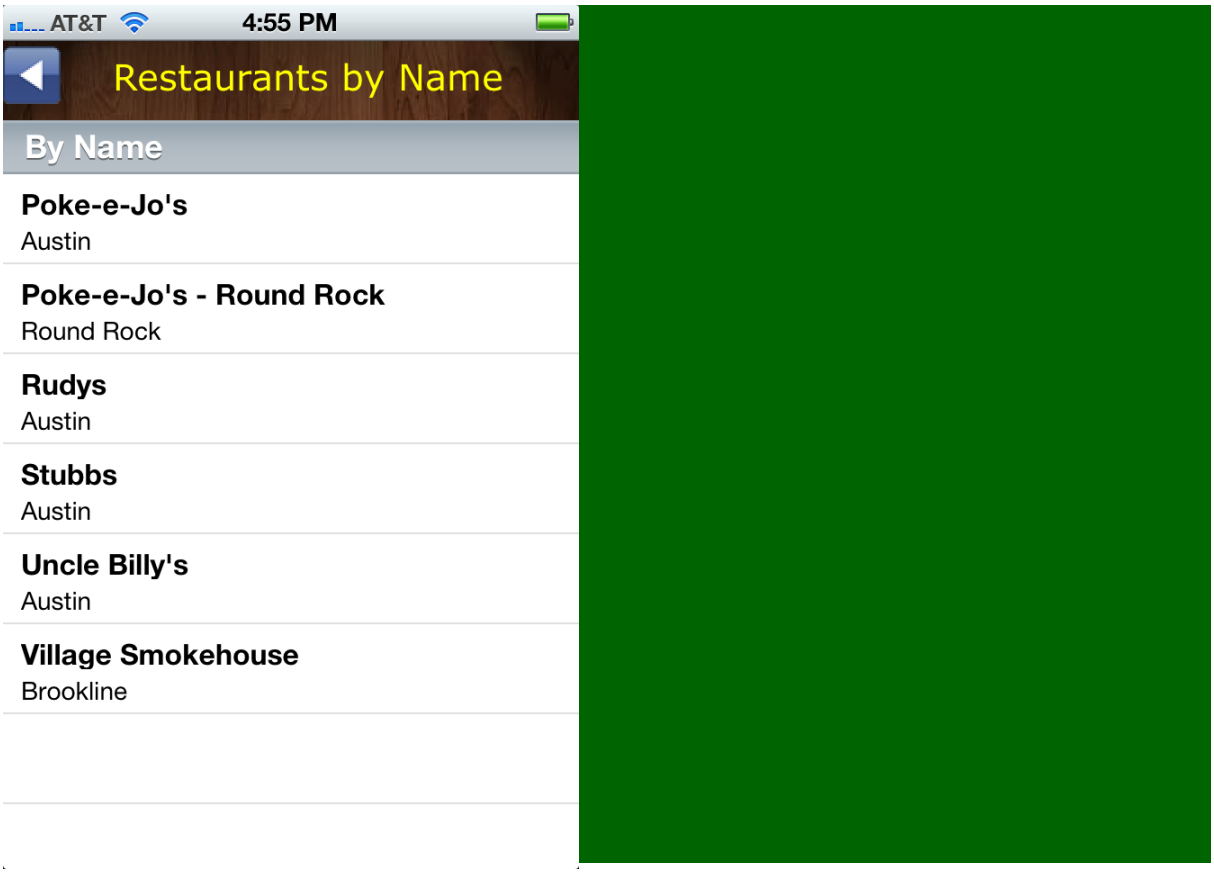
Click the Add Order By icon to the right of the worksheets field, shown below.



This adds an Order by Database Field command that lets us specify the field to sort by and whether to sort in ascending or descending (normal or reverse) order.



If we change this to order by the *restaurantname* field rather than the *city* field, but leave the order as Ascending, here's the result when we view the list of restaurants.



It's now alphabetized even though the entries in the spreadsheet are not. From now on, whenever users add a new restaurant, it will be added at the bottom of the list in the spreadsheet but will always automatically display in alphabetical order when viewed in a table on the mobile device.

Example - User-Added Images

We can add participation to apps by letting users add their own entries; this adds a social media element to the app. For example, people who correspond with others in the same profession via email may never meet in person except at a conference. (A common comment about conferences is that "it let me put a face to the names".)

One way to add this social media element is to let users take pictures, annotate the pictures - "So this is what Jane Doe looks like!" - and store them for review after the conference. This feature is straightforward, as the following example shows. (The example was in an app for a conference in Chicago in May 2012. All personal pictures have been removed.)

This feature had three components:

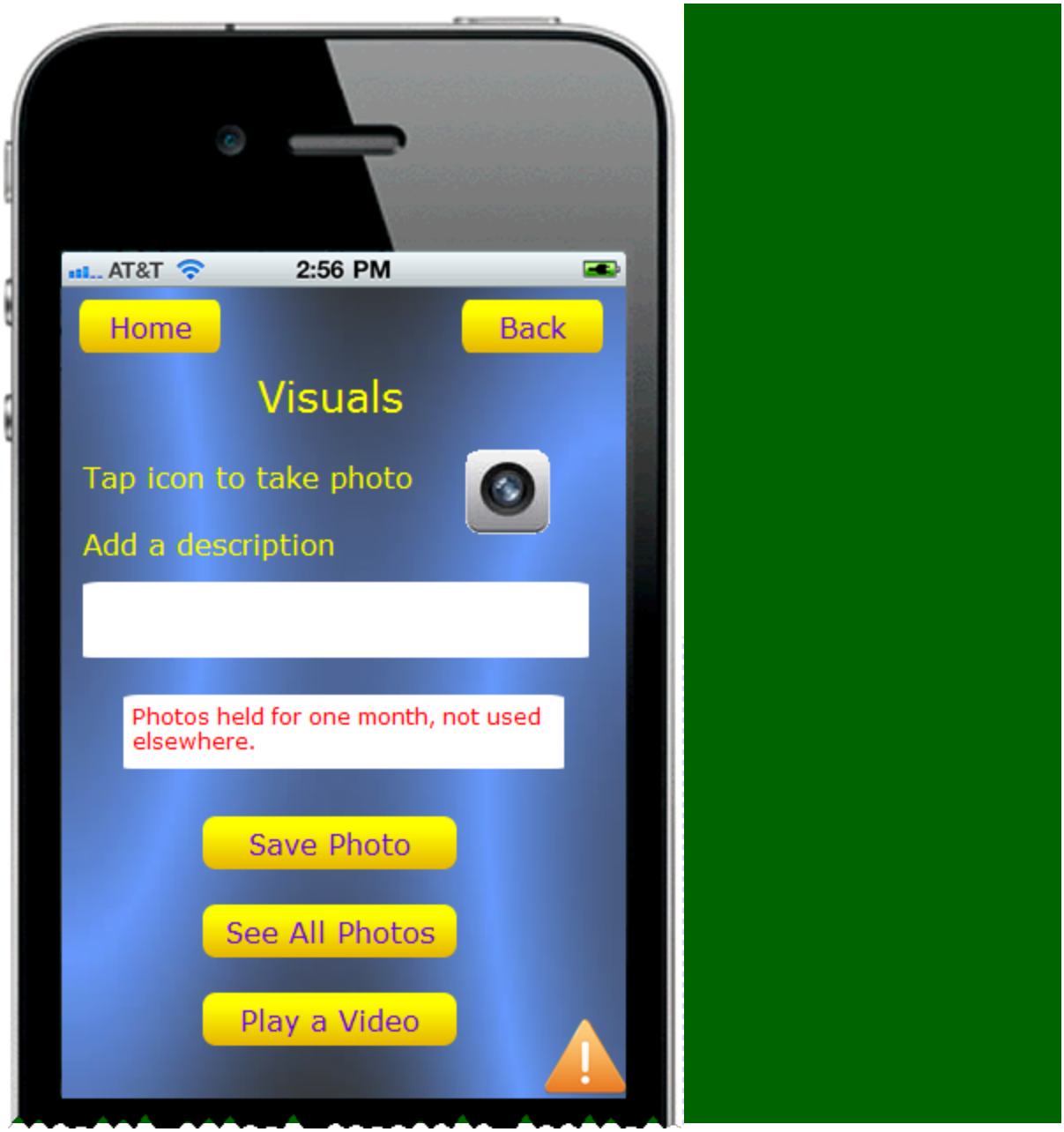
1. Take and annotate a picture and store the picture and annotation. See "Take, Annotate, and Store" on page 81
2. View a list of all the pictures and annotations. See "View a List of All Pictures" on page 84
3. Get a larger view of one picture. See "Get A Larger View of One Picture" on page 87

Note: This feature works very well mechanically, but it is sensitive to the quality of the network because graphic files are large. If network connectivity is poor or fair, you may not want to use this feature because it will slow down the app.

Let's go through each component.

Take, Annotate, and Store

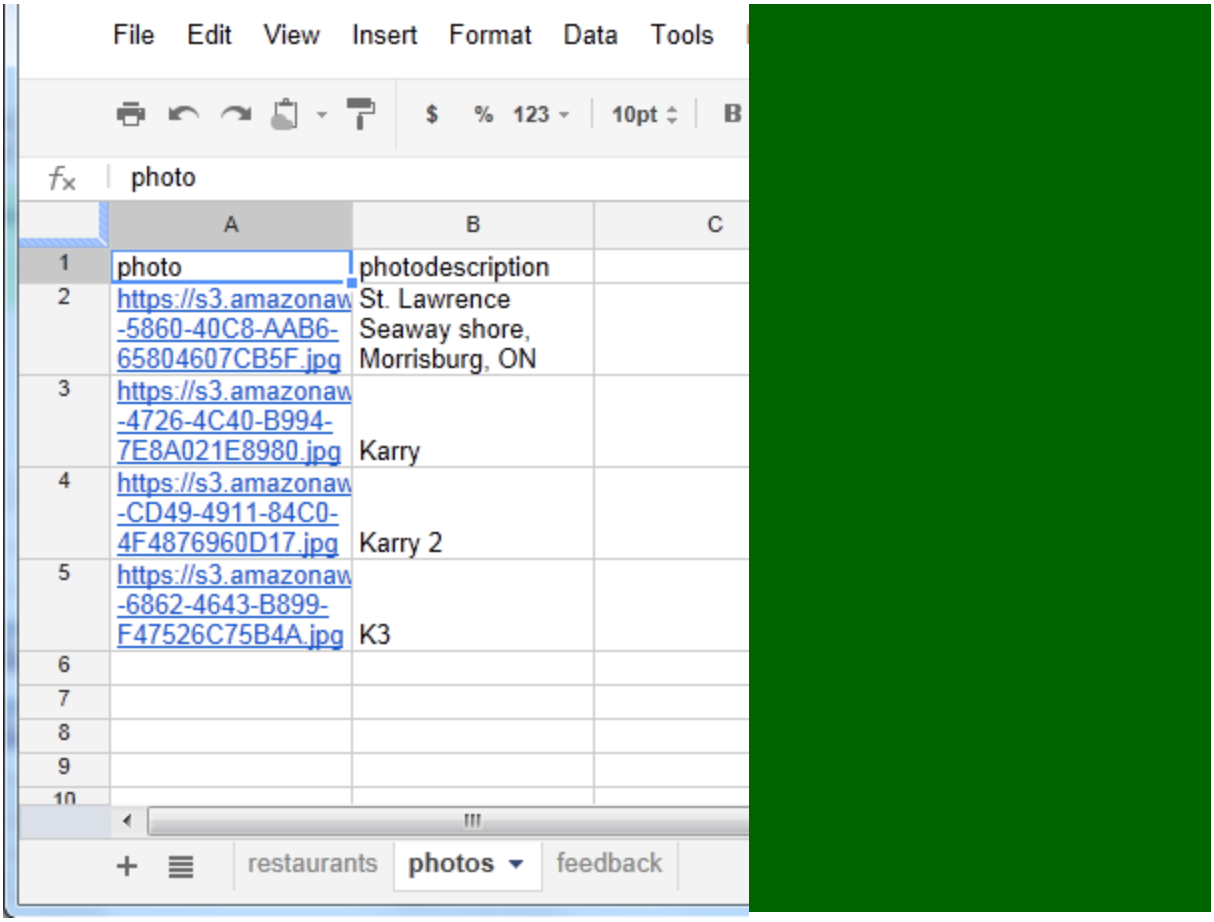
Here's the "take and annotate a picture" page in the app, in ViziApps.



The relevant fields are:

- The camera icon at the top right, which opens the iPhone camera and lets you take a picture. Its internal name is *camera*.
- The add a description field at the top, with the internal name *photo_description_field*.
- The Save Photo button, with the internal name *save_photo_button*.
- The See All Photos button, with the internal name *see_all_photos_button*.
- The alert, at the bottom right, with the internal name *photo_feedback*.

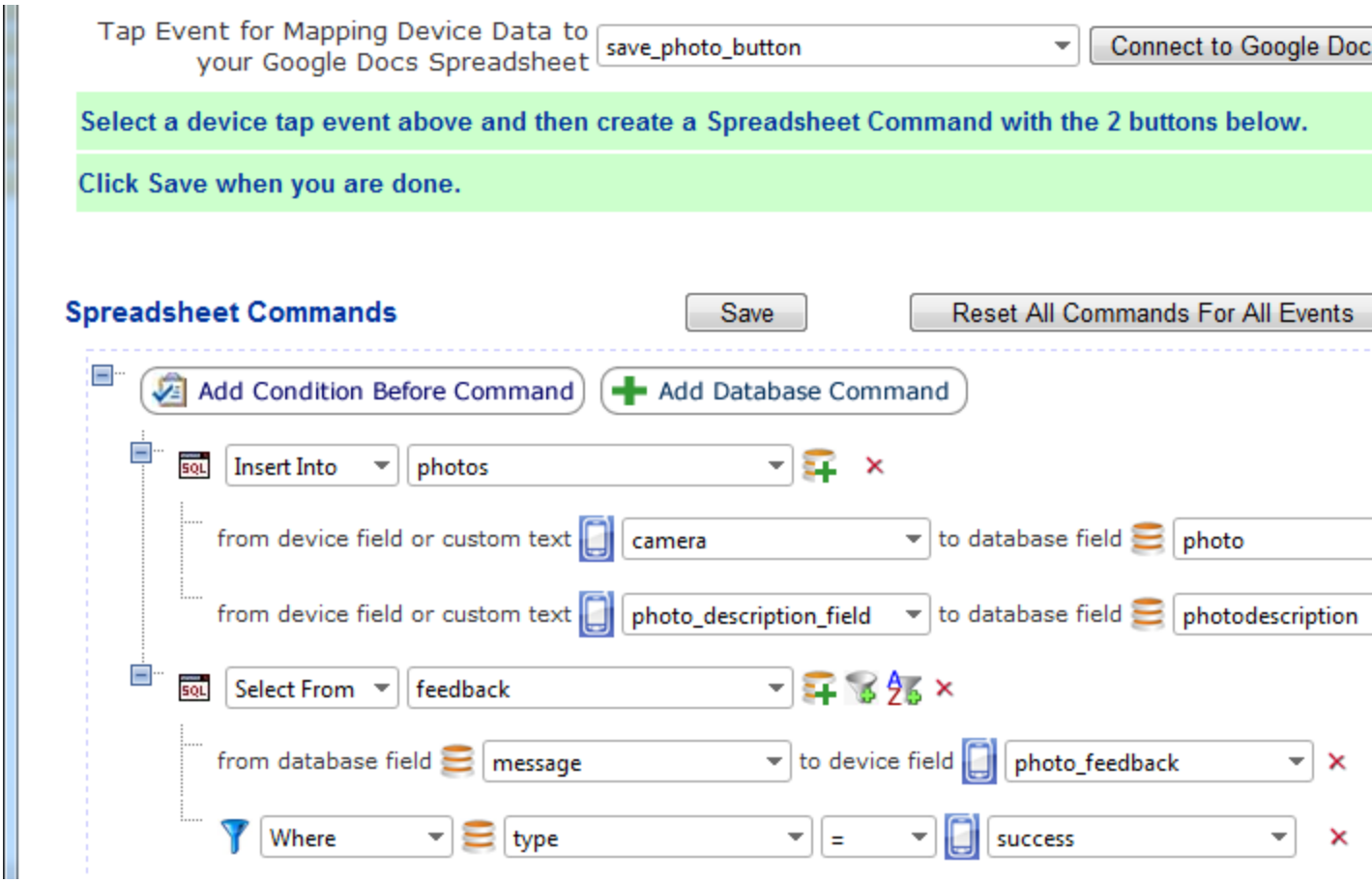
Here's the related spreadsheet worksheet, called *photos*:



The relevant fields are:

- *photo*
- *photodescription*

And here are the data management commands:

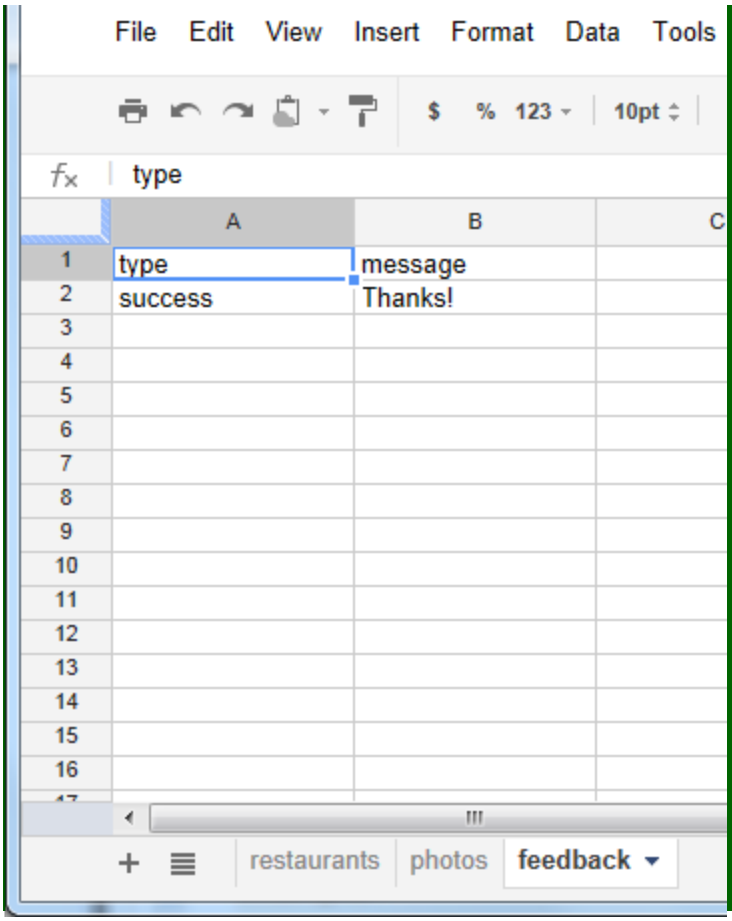


When the user takes a picture, it's stored in the *camera* field on the device. The optional description is stored in the *photo_description_field* field on the device.

When the user taps the *save_photo_button* button:

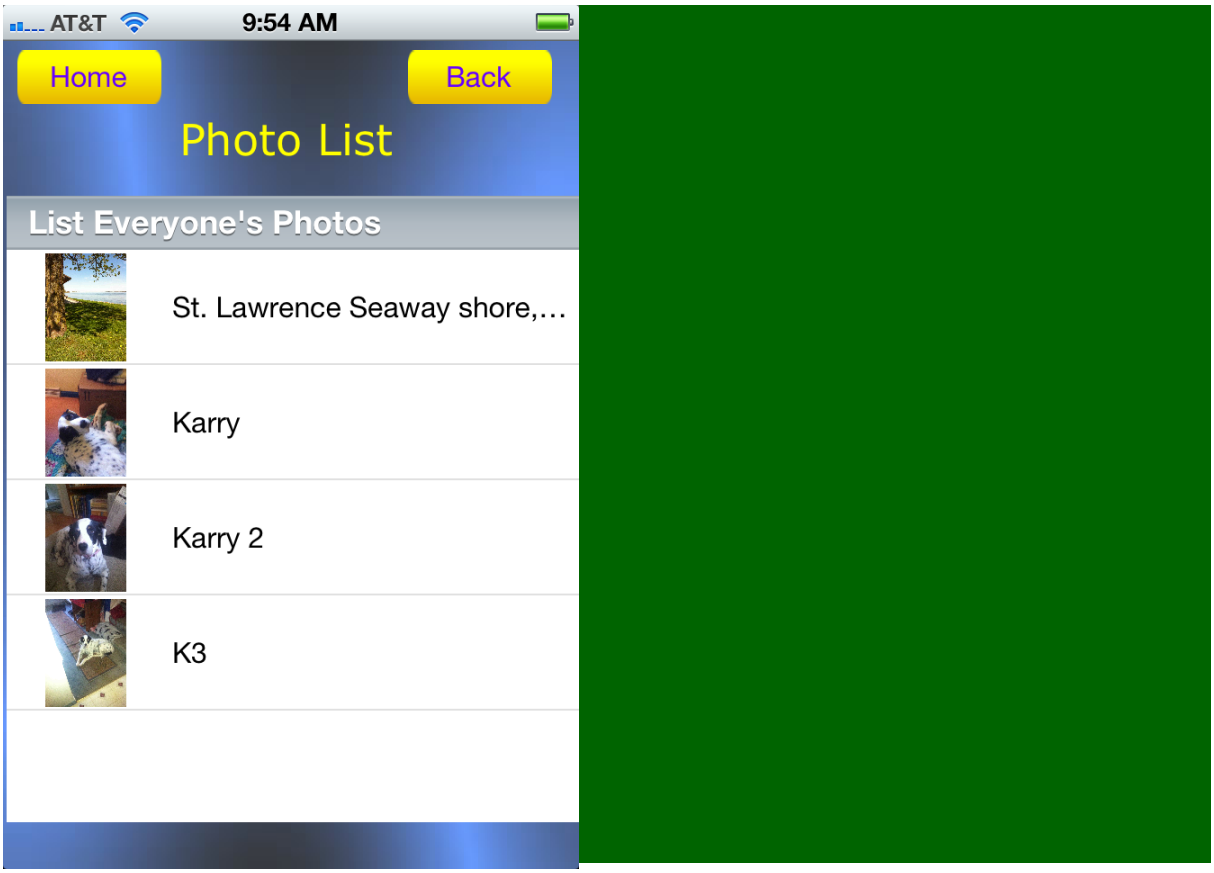
1. The app stores the photo in the cloud, inserts its URL in the *photo* field in the spreadsheet, and stores the description in the *photodescription* field in the spreadsheet.
2. The app displays a feedback message to the user, confirming that the data was added to the spreadsheet.

The image below shows the *feedback* worksheet for the example.

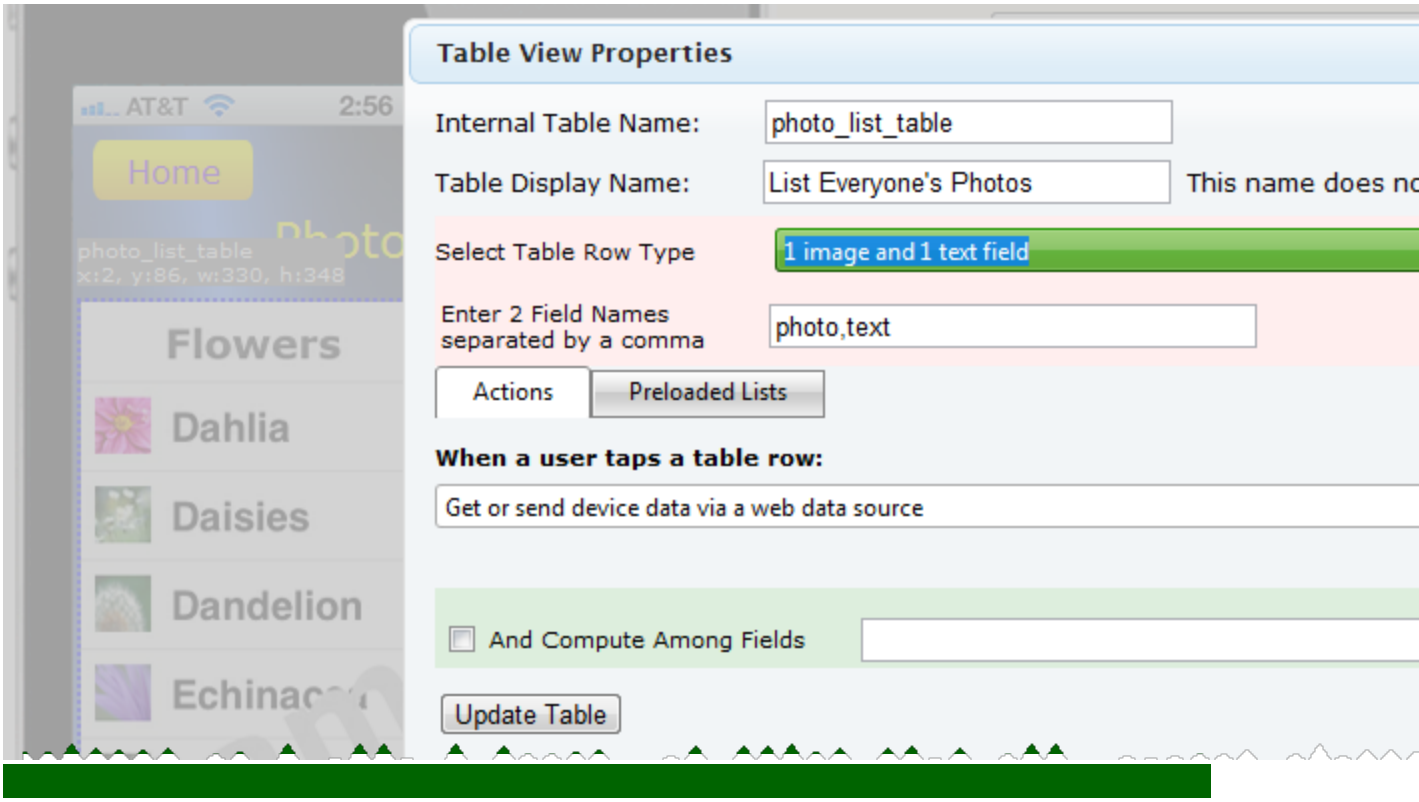


View a List of All Pictures

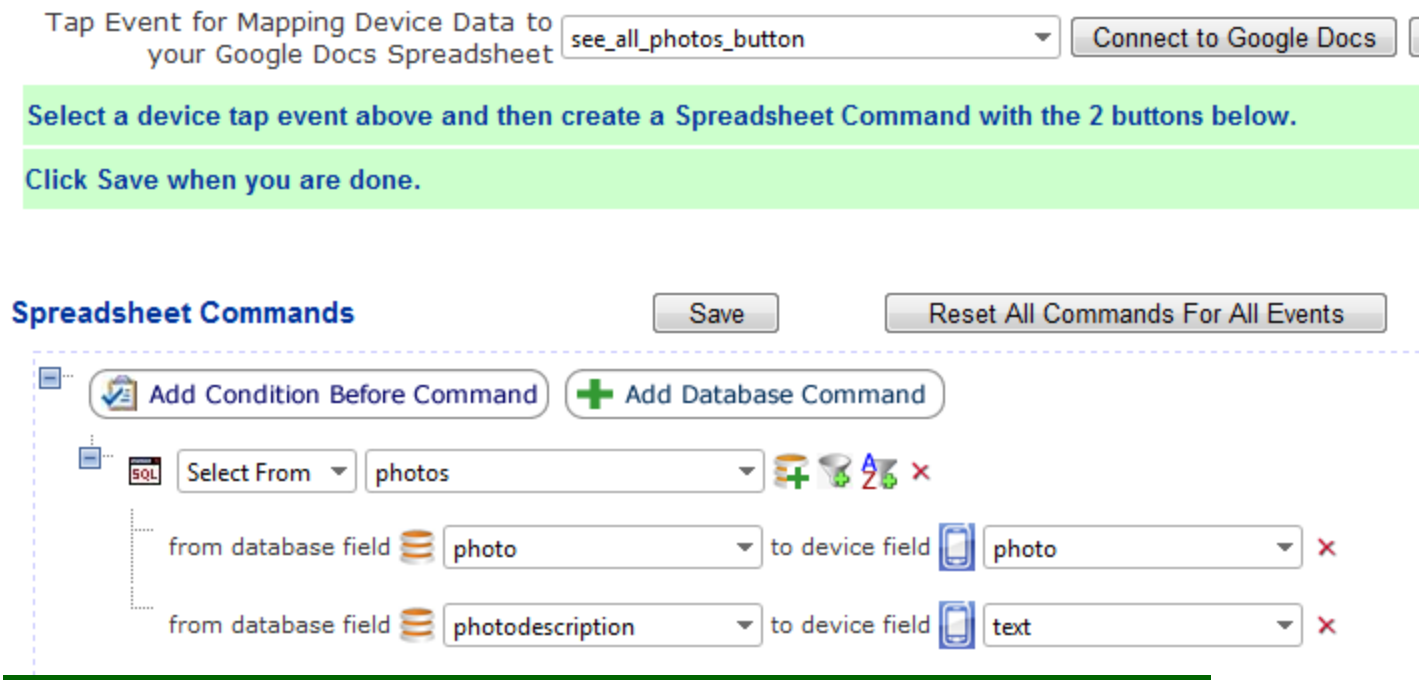
Once the pictures are stored, users can see a list of all pictures taken by all users. Here's the list, live in the app.



Here's the table and table properties in ViziApps:



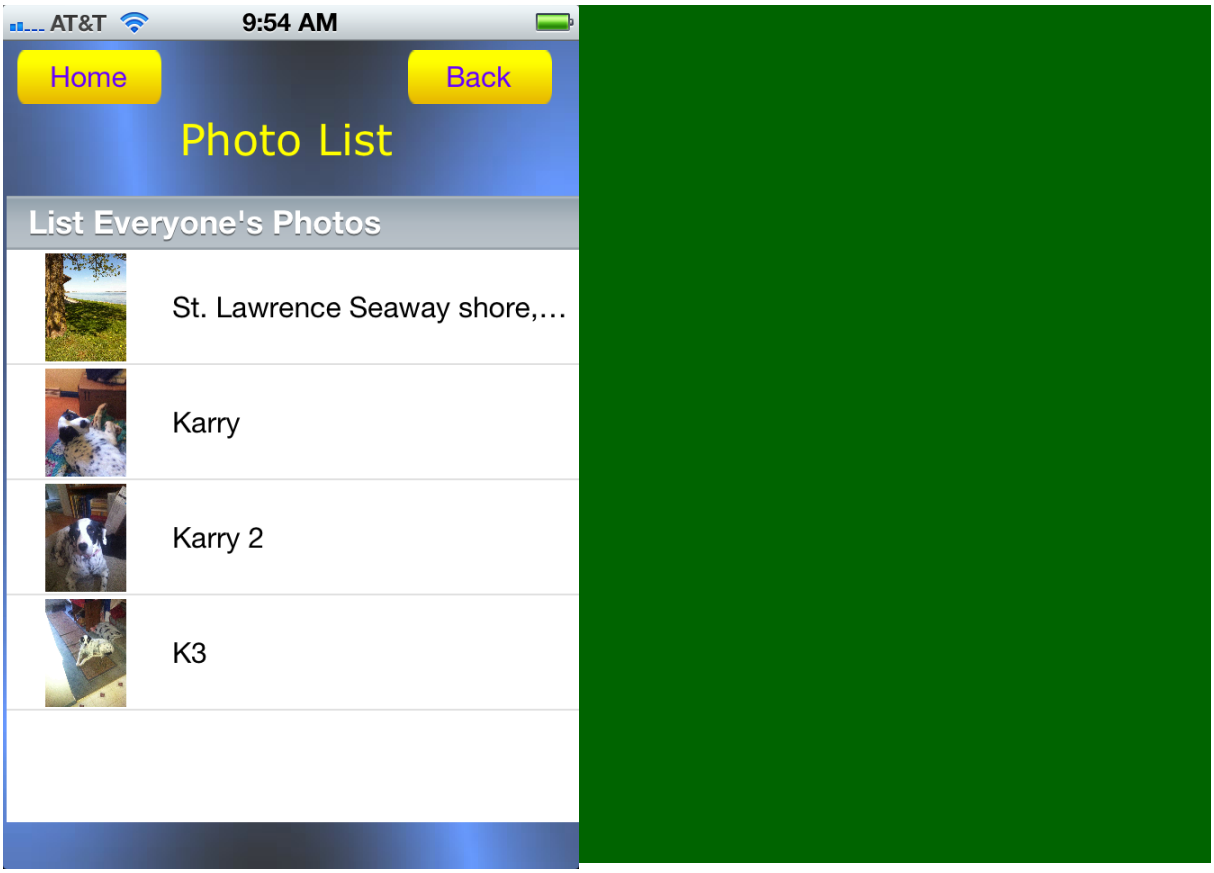
The two fields' internal names - the column names in this case - are *photo* and *text*.
Here are the data management commands:



When the user taps the *see_all_photos_button* button:

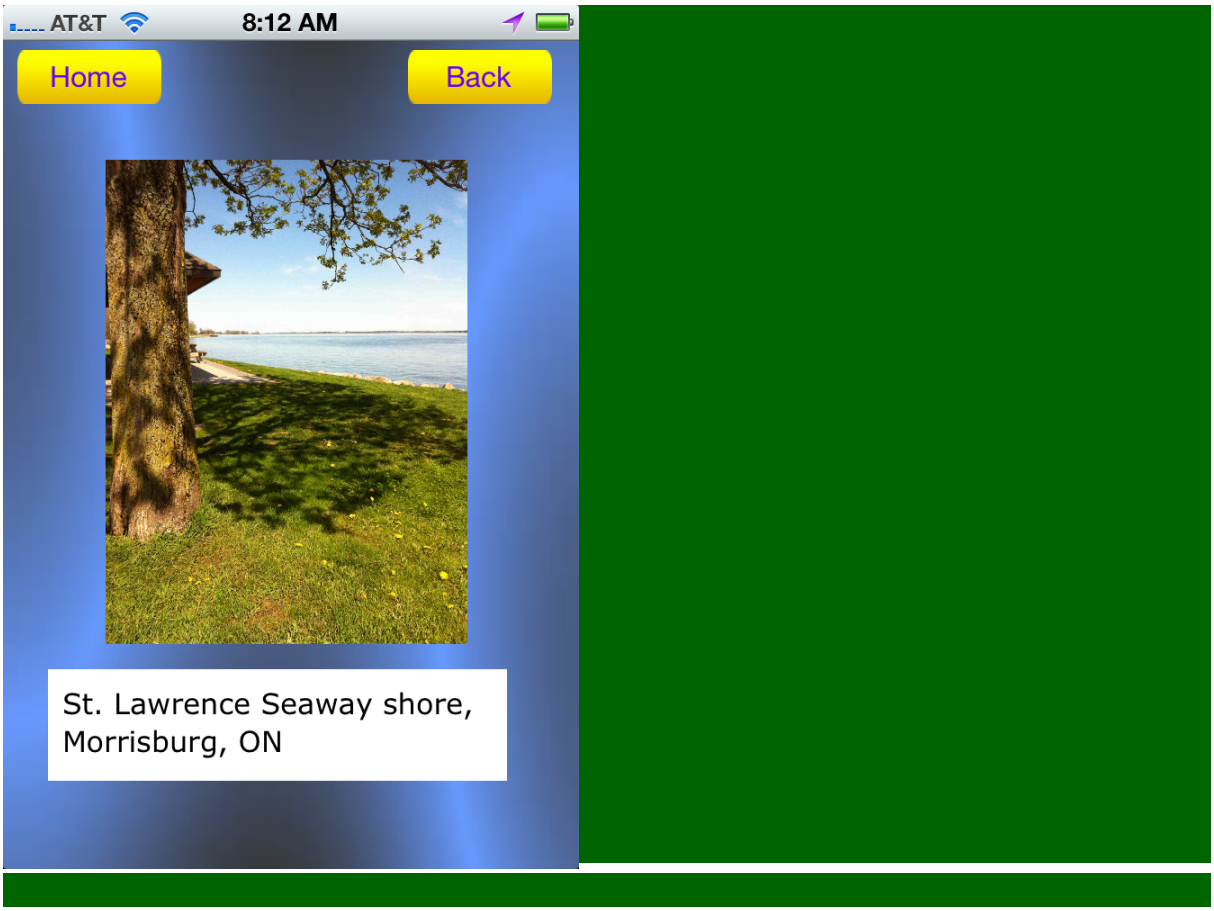
1. The app reads the URLs in the *photo* field in the spreadsheet, finds those photos in the cloud, and displays them in the *photo* field on the device - column 1 in the table.
2. The app reads the entry in the *photodedescription* field in the spreadsheet and displays that in the *text* field on the device - column 2 in the table.

The result is this:

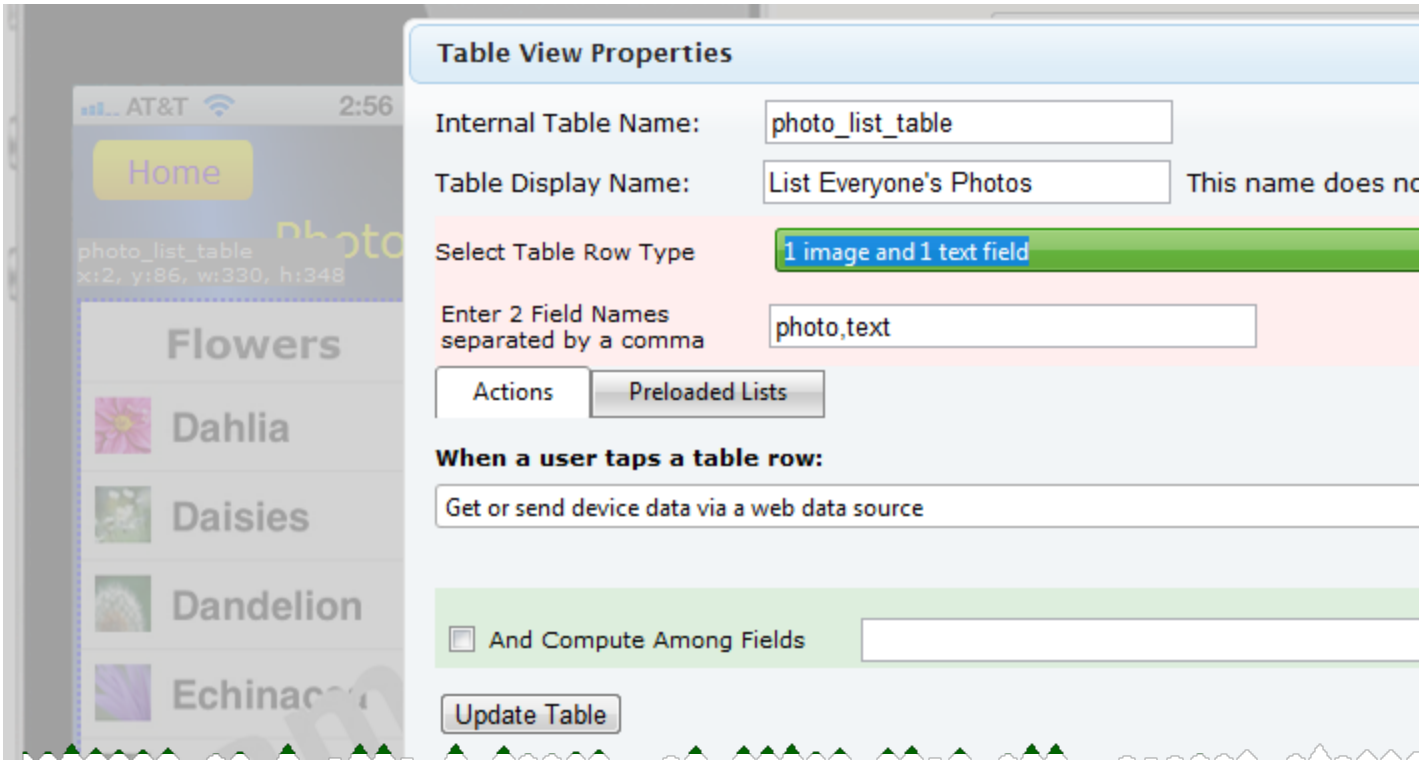


Get A Larger View of One Picture

Once the list of pictures displays, users may want to see a larger view of a particular one. To do so, they can just tap on the desired picture, which displays in a larger form, as shown below.

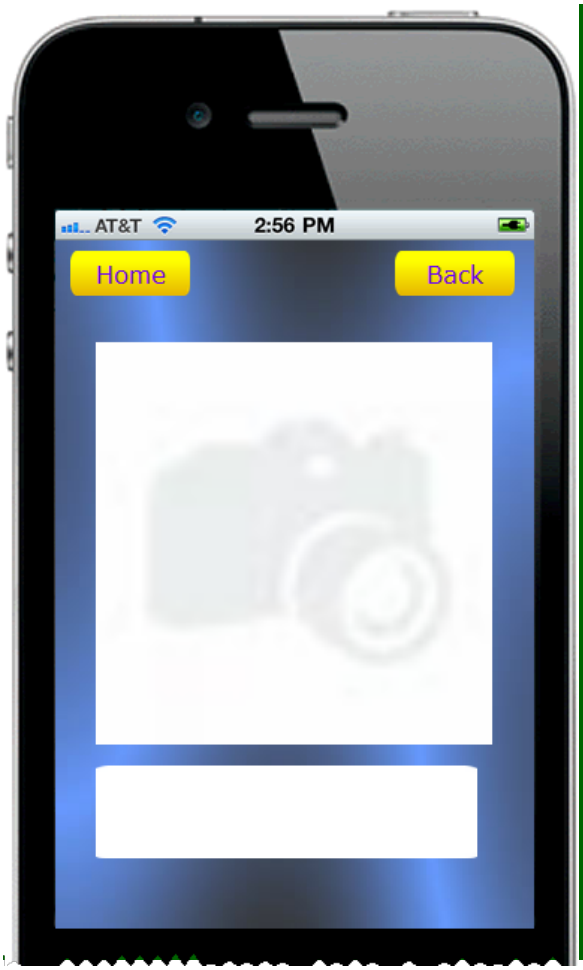


Here's the table and table properties in ViziApps:



Note that the two fields' internal names - the column names - are *photo* and *text*.

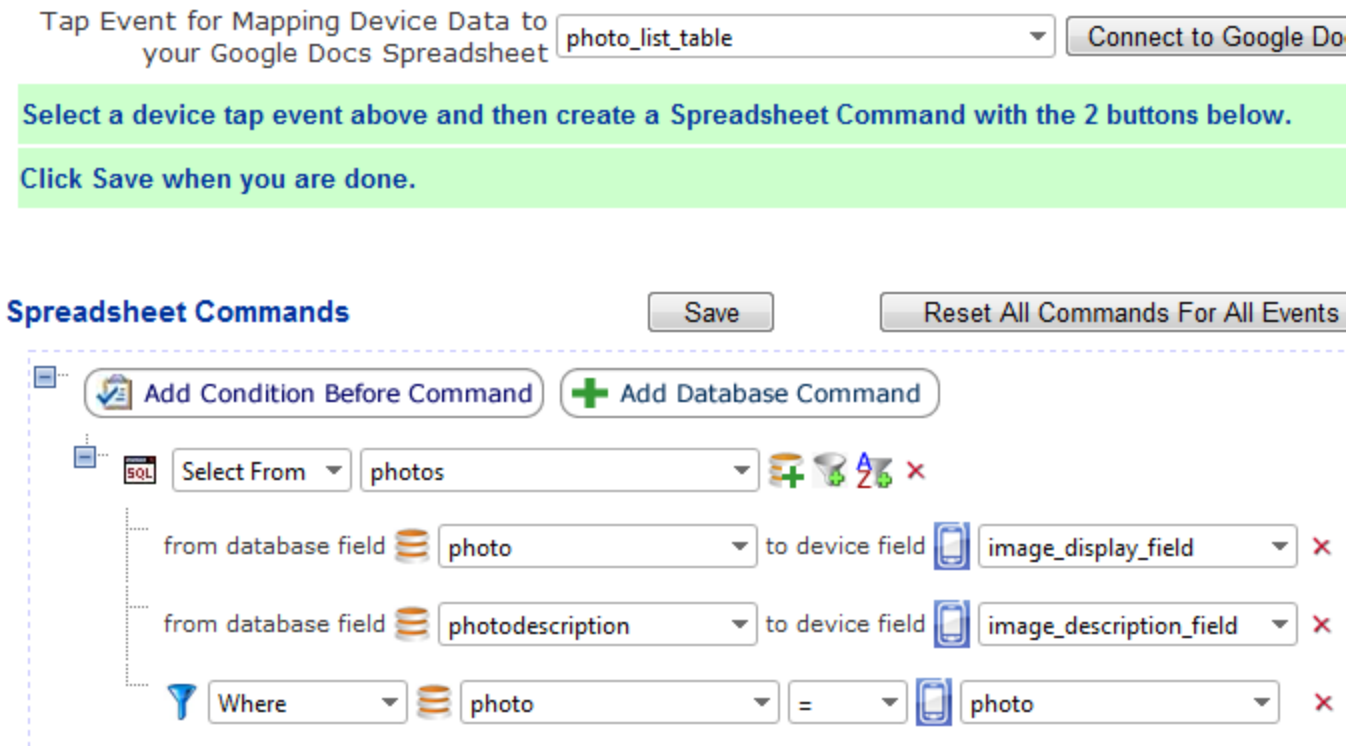
Here is the page on which the selected image will display in a larger format.



The relevant fields are:

- The large top field with the camera icon. The image will display in this field, whose internal name is *image_display_field*. You can modify its size.
- The small lower field in which the image's description displays. It's internal name is *image_description_field*.

Here are the data management commands:



When the user taps the table row containing the desired photo, the *photo* field:

1. The app selects the desired photo from the *photos* worksheet, extracts its URL from the *photo* field in the database, finds that picture and displays it in the large *image_display_field* field on the device.
2. The app selects the related *photodescription field* entry from the spreadsheet and displays it in the smaller *image_description_field* field on the device.
3. The Where condition specifies that the app should select the URL from the spreadsheet equal to the photo that the user clicked in the table on the device.

Example - User-Added Text

We can add participation to apps by letting users add their own entries; this adds a social media element. For example, a conference app might offer a list of barbecue restaurants in the conference city to help attendees decide how to spend their free time. Allowing conference attendees, not just the app developer, to add to the list offers two benefits.

- It takes advantage of local attendees' familiarity with the restaurant scene that an out-of-town app developer won't have.
- It adds the social media element that lets attendees add entries and visibly take credit for their entries by offering them a credit line.

This feature is straightforward, as the following example shows. (The example is from an app for a conference in Austin, TX in 2011. One part of the app provided a list of local barbecue restaurants and let users add their own listings.)

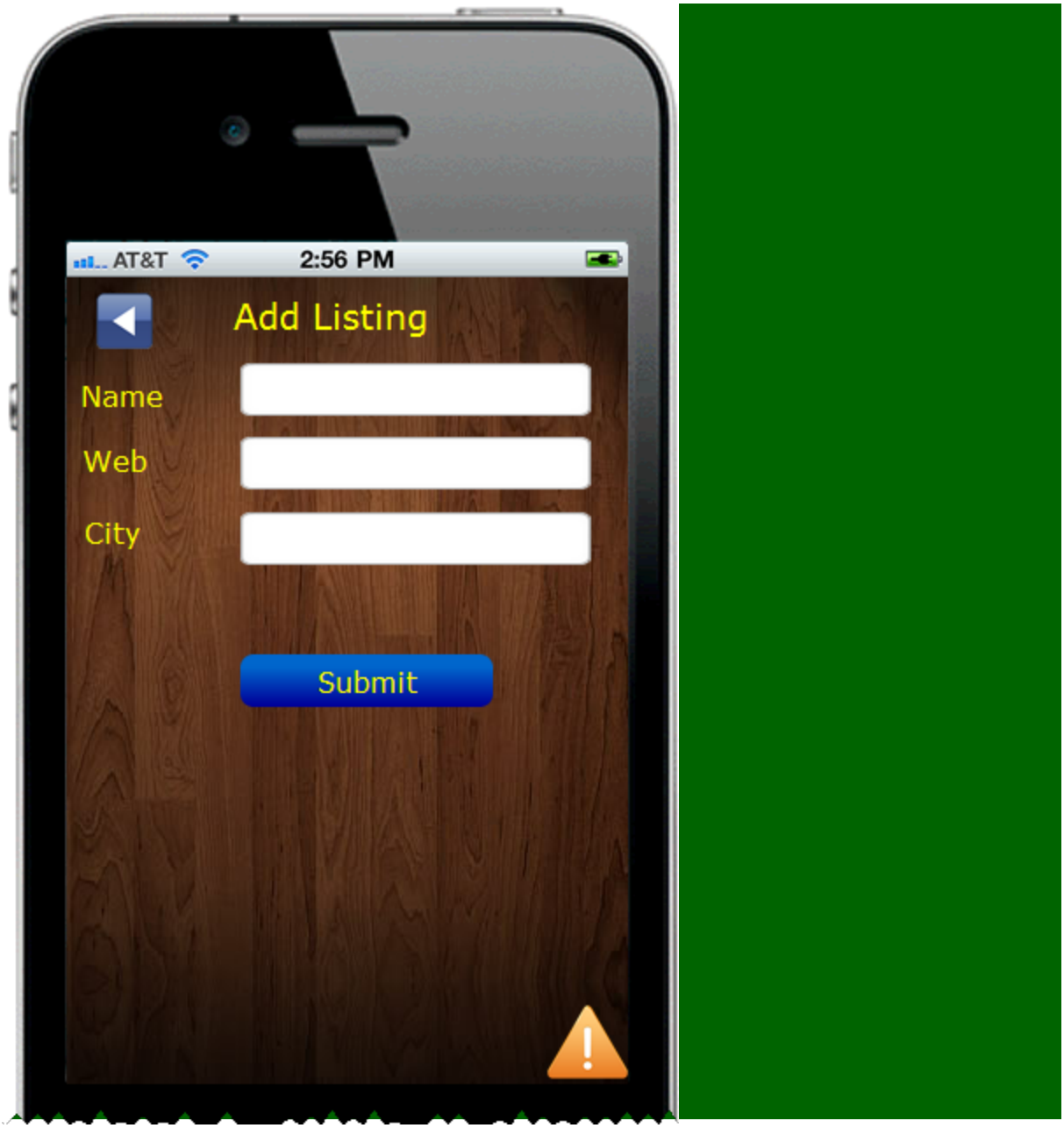
This feature had two components:

1. Add a restaurant to the predefined list. See "Add a Restaurant to the List" on page 91
2. View a list of the restaurants by name or city. See "View a List of the Restaurants" on page 94

Let's go through each component.

Add a Restaurant to the List

Here's the "add a listing" page in the app, in ViziApps.

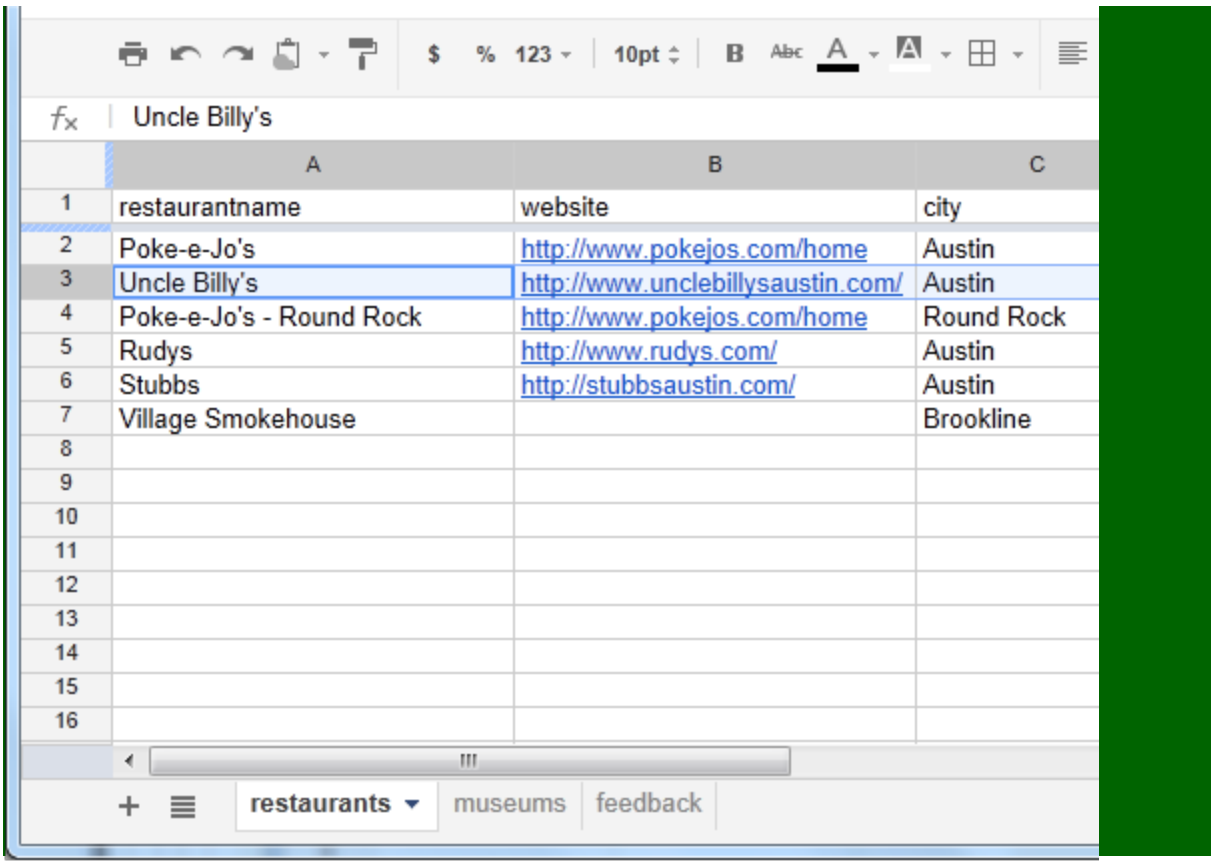


The relevant fields are:

- The Name, Web, and City fields with the internal names *add_name_field*, *add_web_site_field*, and *add_city_field*.

- The Submit button, with the internal name *add_listing_submit_button*.
- The alert, at the bottom right, with the internal name *confirmation*.

Here's the related spreadsheet worksheet, called *restaurants*. Notice the order in which the restaurants are listed. They're in the order in which users entered them rather than in alphabetical order. We'll discuss the effect at the end of this topic.



fx | Uncle Billy's

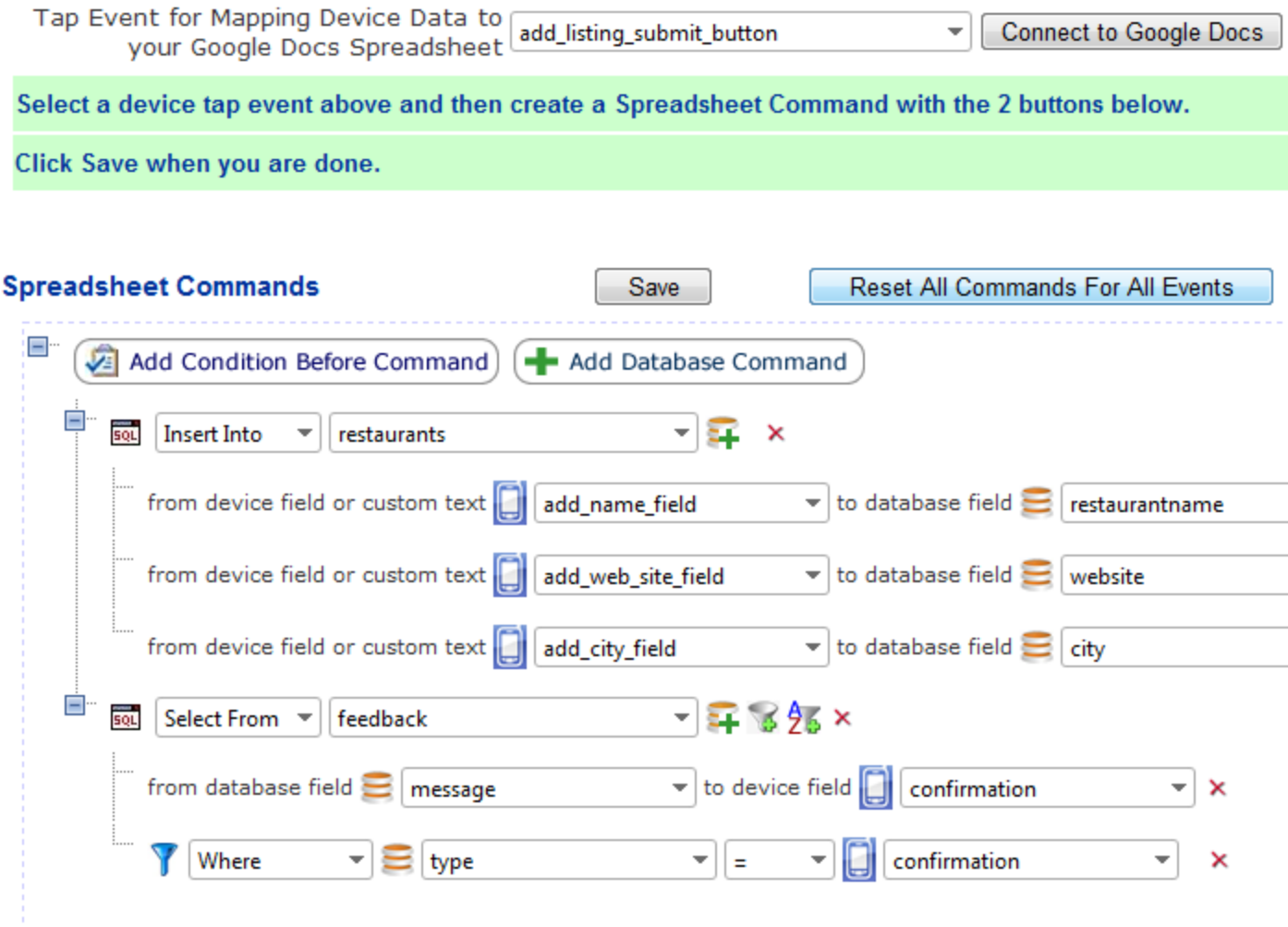
	A	B	C
1	restaurantname	website	city
2	Poke-e-Jo's	http://www.pokejos.com/home	Austin
3	Uncle Billy's	http://www.unclebillysaustin.com/	Austin
4	Poke-e-Jo's - Round Rock	http://www.pokejos.com/home	Round Rock
5	Rudys	http://www.rudys.com/	Austin
6	Stubbs	http://stubbsaustin.com/	Austin
7	Village Smokehouse		Brookline
8			
9			
10			
11			
12			
13			
14			
15			
16			

restaurants | museums | feedback

The relevant fields are:

- *restaurantname*
- *website*
- *city*

Here are the data management commands:

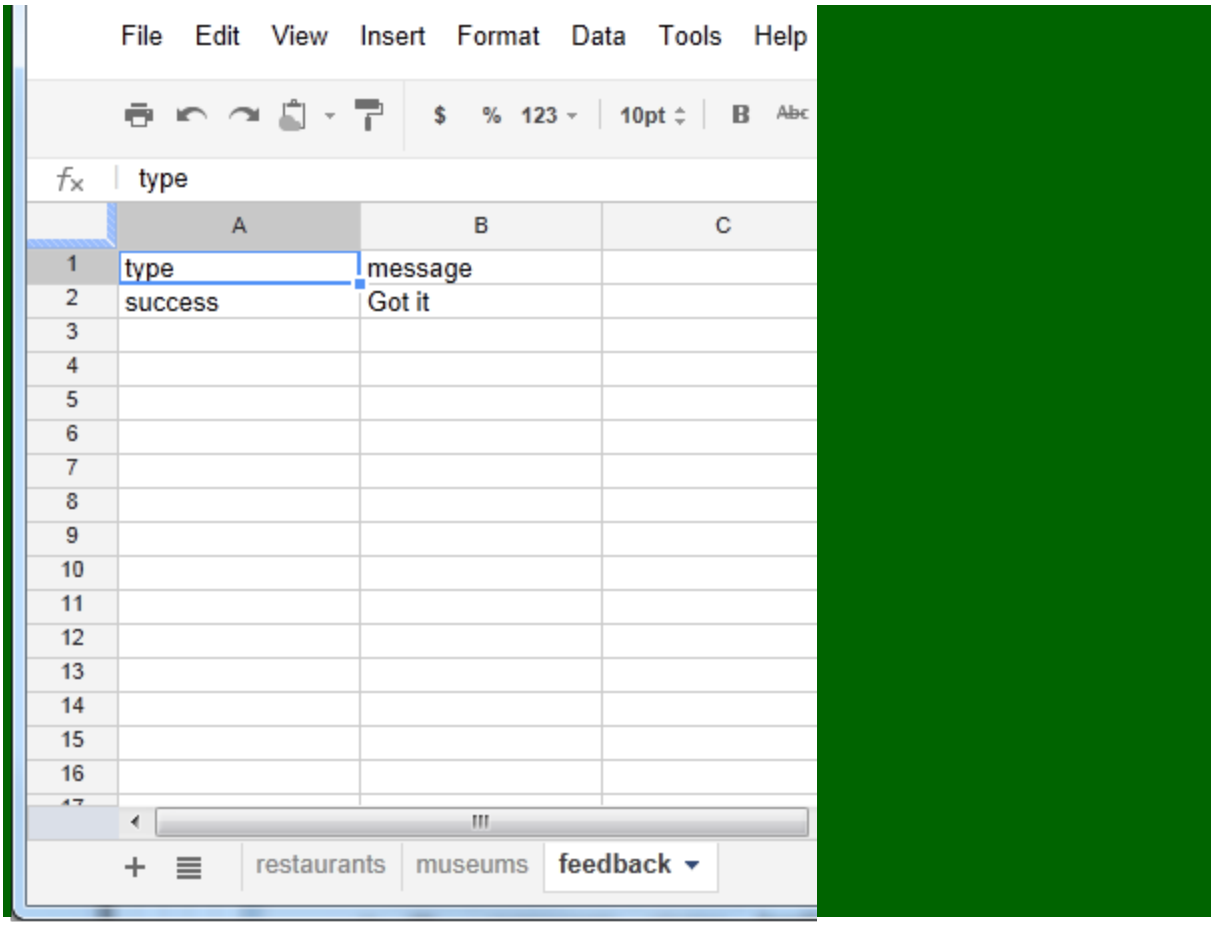


To add a restaurant to the list, the user makes the appropriate entries in the add_name_field, add_web_site_field, and add_city_field fields on the mobile device.

When the user taps the *add_listing_submit_button* button:

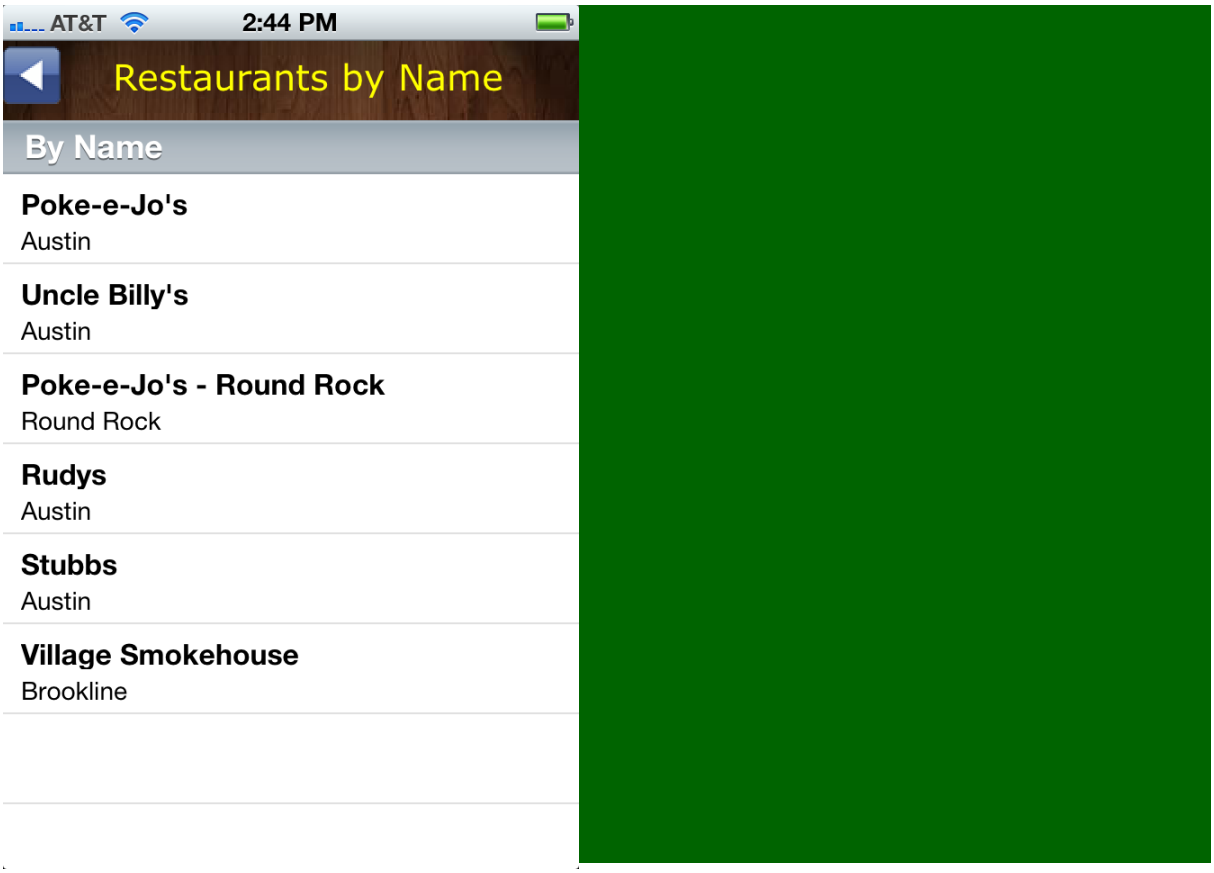
1. The app inserts the entries in the *add_name_field*, *add_web_site_field*, and *add_city_field* fields in the equivalent spreadsheet fields - restaurantname, website, and city in the *restaurants* worksheet.
2. The app displays a feedback message to the user, confirming that the data was added to the spreadsheet.

The image below shows the *feedback* worksheet for the example.

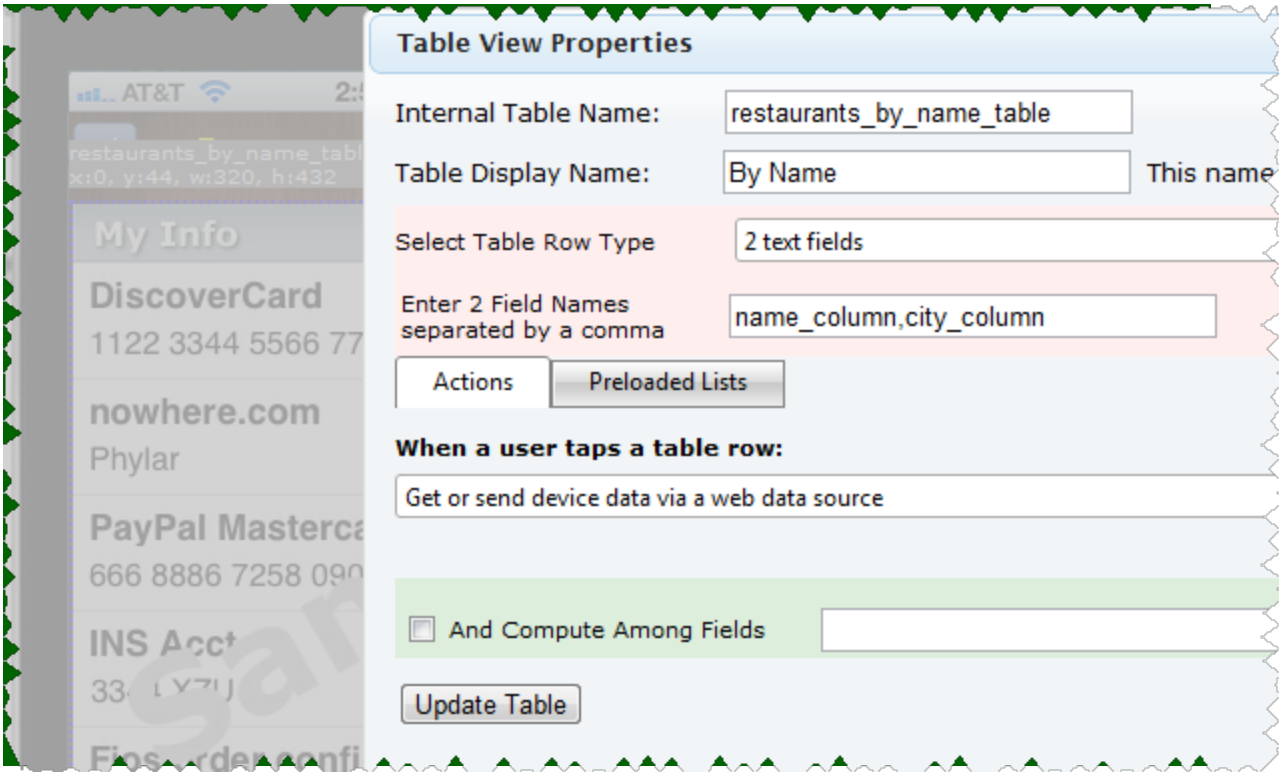


View a List of the Restaurants

Once the restaurant listed are stored, users can see a list of them by name (or by city or any other criterion set up by the app developer). Here's the list, live in the app.

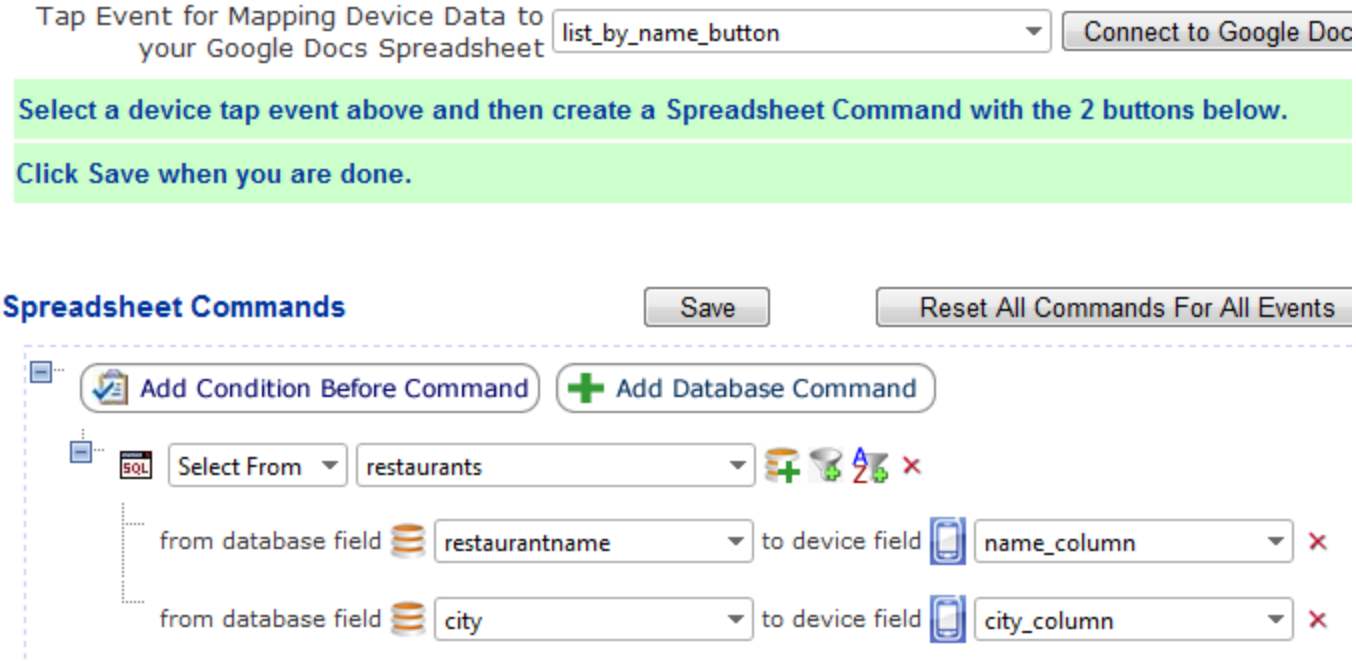


Here's the table and table properties in ViziApps:



The two fields' internal names - the column names in this case - are *name_column* and city_column.

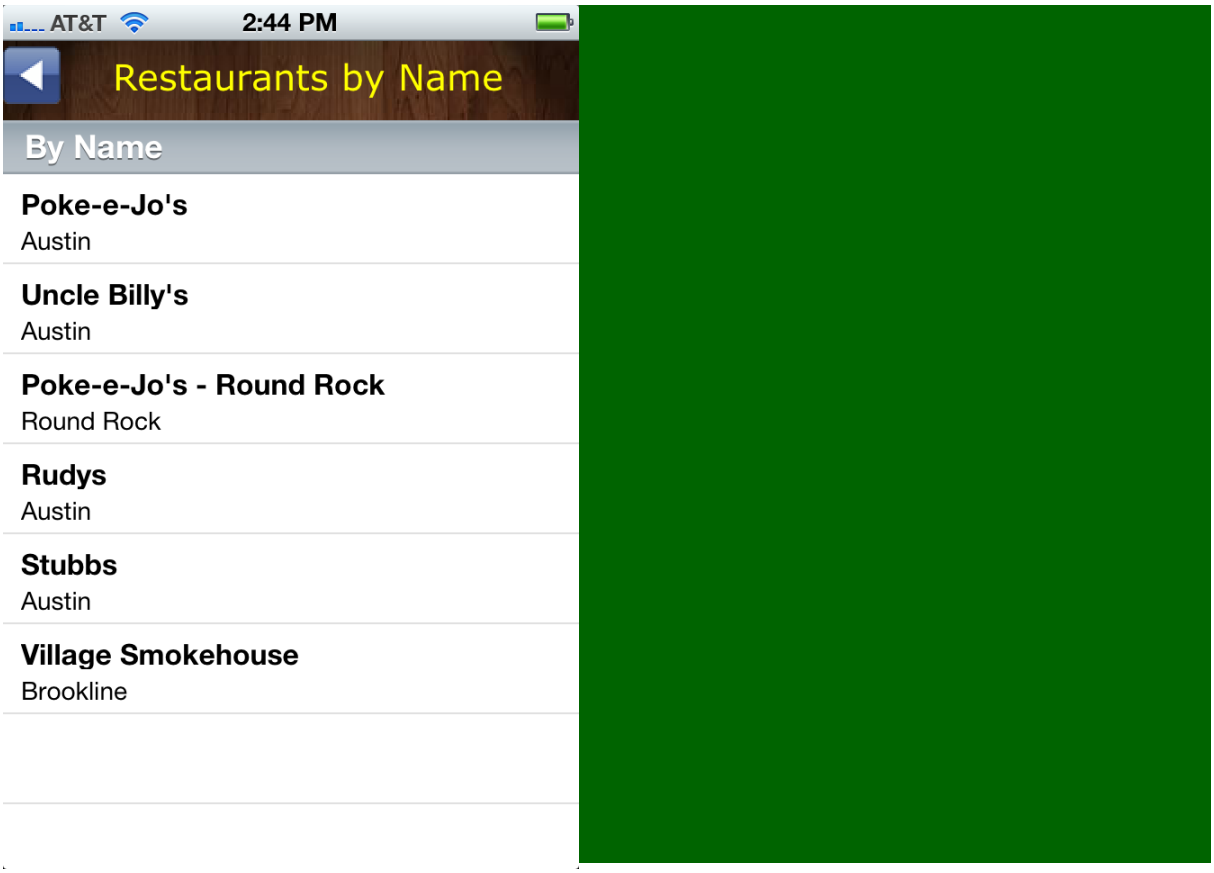
Here are the data management commands:



When the user taps the *list_by_name_button* button:

1. The app reads the entries in the *restaurantname* and *city* fields on the *restaurants* worksheet and displays them in the *name_column* and *city_column* fields on the mobile device.

The result is this:



Note: This list is correct but has one problem. The restaurants are listed in their order rather than alphabetically, making it hard to find a particular restaurant. The app developer could sort them alphabetically in the spreadsheet, but would have to resort them every time a user entered a new restaurant in the list. Instead, we'd like to set up the data management so that the entries are always in alphabetical order on the device no matter what order they appear in on the spreadsheet. We can do that using ViziApps' sort function, described in the sorting example. See "Example - Sorting Data In a Table" on page 77