



Development

Preparation

Pre-knowledge

Before proceeding with the task, we assume that you are already familiar with JavaScript. Of course, if you do not know enough about JavaScript, you can refer to [Appendix: JavaScript Getting Started Guide](#), which briefly explains JavaScript syntax knowledge and lists some introductory tutorial sites.

In addition, this script will also use the [Script API](#), using these APIs you can use JavaScript to manipulate tables:

- [Space API](#) Space API, you can get datasheets under the space and do follow-up operations
- [Datasheet API](#) It can obtain datasheet information, and support operations on views, fields, and records in the data table
- [View API](#) can get view information and records under the specified view
- [Field API](#) can get the field information in the datasheet
- [Record API](#) can get the information of each record
- [UI APIs](#) can render interactive standardized UI components, such as text input box, field selector, etc., and also support rendering specified content (text, tables, etc.)

It doesn't matter if you are not familiar with the API, we will use them a lot in the follow-up practice, so you can learn it in practice.

Target

Next, we will use JavaScript to develop a search and replace widget in the script widget. This function is useful in many scenarios, such as batch replacement for a misspelling of a proper noun, and so on.

Before we start, we need to disassemble the specific work of this task. We mainly need to implement the following steps:

1. Requires user input for which data to find and replace
 - Value to find
 - Value to replace
 - In which field to find and replace
2. Extract all the data in the table, then find all the data you want to find and replace
 - [Loop](javascript#Loop statement) through each record in the table and extract the data of the specified field in the record
 - [Determine whether](javascript#Conditional statements) the data contains the lookup value, if so, you need to replace it with the replacement value, and save the replaced data as an [array](#)
 - Finally, replace the saved data one by one with the data to be replaced in the table
3. Output the final result
 - Replaced successfully
 - Replacement failed

Let's start writing this widget together! For more information on how the sample code works, please see [here](#).

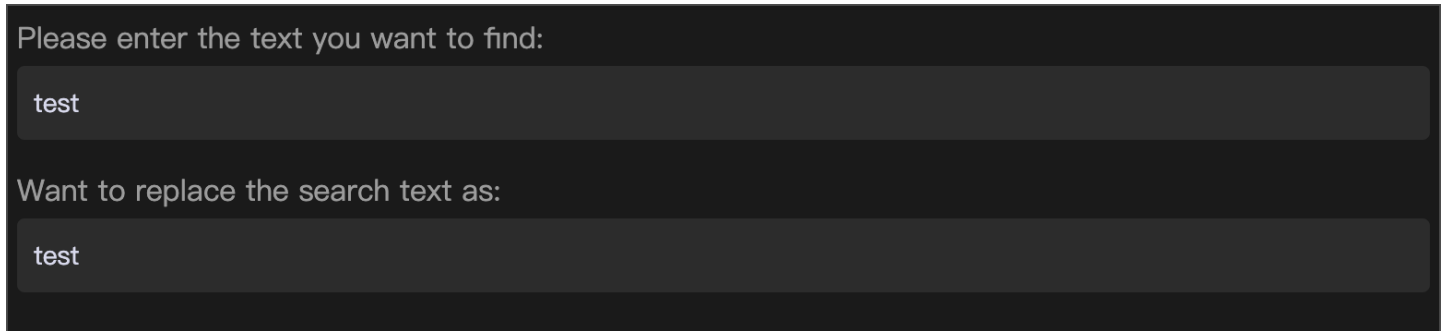
Begin to write

Use input API to input data

[Input API](#) can ask the user to enter the content, convenient for subsequent interaction. Therefore, we can use this API to ask the user to fill in the necessary information:

```
const findText = await input.textAsync(
  "Please enter the text you want to find:"
);
const replaceText = await input.textAsync(
  "Want to replace the search text as:"
);
```

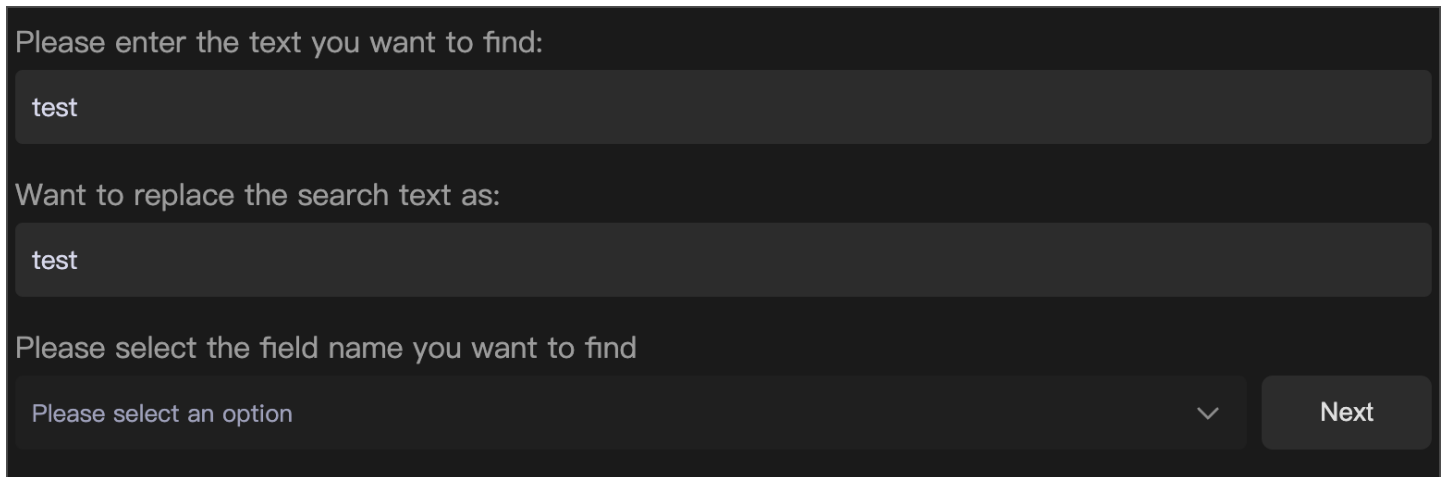
The running effect is as follows:



After the above steps are done, we can get the text that users want to find replacement. Next, we need to let the user specify a field to facilitate us to find the data. Here we can use [Input.fieldAsync](#) to complete:

```
// You need to get the current datasheet through Space API and pass it as a parameter to input.fieldAsync
const datasheet = await space.getActiveDatasheetAsync();
const field = await input.fieldAsync(
  "Please select the field name you want to find",
  datasheet
);
```

The running effect is as follows:



Match data in the datasheet

After the above steps, we have completed the acquisition of all the necessary information. At this time, we should need to write the most important logical part. This part mainly uses JavaScript knowledge. First of all, we need to get the `id` of field we need to find. At this time, we can get it based on the `field` parameter obtained above, as shown below:

```
// Get the ID of filed
const fieldId = field.id;
```

Then use [datasheet.getRecordsAsync](#) to get all the records in the current table, and create a new array to store the data that needs to be replaced:

```
// Get all record objects through the datasheet API
const records = await datasheet.getRecordsAsync();
const finalData = [];
```

At this time, the basic variable definition has been completed. The following needs to perform a loop traversal lookup for `records` and use [record.getCellValueString](#) to find the `fieldId` data:

```
// traverse records
for (let record of records) {
  const recordId = record.id;
  // Get the data of the specified field in the record as cellValue
  const cellValue = record.getCellValueString(fieldId);
}
```

Then you need to use `<String>.replaceAll(findText, replaceText)` in JavaScript to find and replace data. This method can replace all `findText` in `String` with `replaceText`. However, since this method is only applicable to string types, it is necessary to determine whether the `cellValue` is empty before use. If it is empty, it means that there is no need to search and replace, and you can directly execute the next cycle with `continue`:

```
for (let record of records) {
  ..... // If the acquired data is empty, jump out of this cycle and directly execute the next time
  if (cellValue == null) continue;

  // Replace findText in cellValue with replaceText, and use a new variable - newCellValue
  const newCellValue = cellValue.replaceAll(findText, replaceText);
}
```

Finally, we need a [conditional statement](javascript#Conditional statements) to judge whether `newCellValue` is equal to `cellValue`. If they are equal, it means that the data does not contain `findText` and no replacement is needed. `newCellValue` to the previously defined `finalData`. The saved data format needs to refer to the [updateRecordsAsync](#) parameter in the Datasheet API:

```
for (let record of records) {
  ..... // Determine whether the data before replacement is consistent with the data after replacement
  if (cellValue !== newCellValue) {
    // Inconsistency means that the corresponding record in the table needs to replace cellValue with newCellValue,
    finalData.push({
      id: recordId,
      valuesMap: { [fieldId]: newCellValue }
    })
  }
}
```

The final complete loop traversal search code is as follows:

```
// traverse records
for (let record of records) {
  const recordId = record.id;
  // Get the data of the specified field of the record as cellValue
  const cellValue = record.getCellValueString(fieldId);

  // If the acquired data is empty, jump out of this cycle and directly execute the next time
  if (cellValue == null) continue;

  // Replace findText in cellValue with replaceText, and use a new variable - newCellValue
  const newCellValue = cellValue.replaceAll(findText, replaceText);
  // Determine whether the data before replacement is consistent with the data after replacement
  if (cellValue !== newCellValue) {
    // Inconsistency means that the corresponding record in the table needs to replace cellValue with newCellValue,
    finalData.push({
      id: recordId,
      valuesMap: { [fieldId]: newCellValue },
    });
  }
}
```

At this point, we can already find all the data that needs to be replaced. At this point, we only need to use [datasheet.updateRecordsAsync](#) to replace the old data with the new data in `finalData`:

```
// Determine whether there is data in finalData, if there is no data, there is no need to replace
if (finalData.length) {
  await datasheet.updateRecordsAsync(finalData);
}
```

Output data using the Output API

In addition to supporting the [Input API](#) to allow users to input data, the script also supports the [Output API](#) to allow developers to output data and display it to users.

In the example of find and replace, we can add a reminder that the replacement is complete to the user to make the experience more complete:

```
// Determine whether there is data in finalData, if there is no data, there is no need to replace
if (finalData.length) {
  await datasheet.updateRecordsAsync(finalData);
  output.text("The replacement is complete!");
} else {
  output.text("No data found to be replaced");
}
```

Final effect and full code

New datasheet

Manager

Grid view

+ New view

Insert record

Hide fields

Filter

Group

Sort

Row height

Share

Find

Form

Mirror

API

Advanced

	Title	Options	Magic link	Check	
1	Test				
2	Test	One			
3	Test	Four			
4	APITable.com				
xukecheng@apitable.com					
6	Test	One			
7	Test	Two			
8	Test	Three			
9	+1-850-555-1234				
10	+1-800-555-1212				
11	+44-800-123-4567				
12	+81-3-1234-5678				
13	+972-3-6123456				
14	+61-2-1234-5678				
15	+33-1-23-45-67-89				
16	+49-89-12345678				
17	+31-20-123-4567				
18	+55-11-2345-6789				
19	12345-67890-1				
20	09876-54321-2				
21	98765-43210-3				
22	56789-01234-4				
23	45678-90123-5				

28 records

Script

Edit code

Stop

Please enter the text you want to find:

Next

Console

```

const findText = await input.textAsync(
  "Please enter the text you want to find:"
);
const replaceText = await input.textAsync(
  "Want to replace the find text with:"
);

// The currently active form needs to be obtained through the Space API and passed to input.fieldAsync as a paramet
const datasheet = await space.getActiveDatasheetAsync();
const field = await input.fieldAsync(
  "Please select the field name you want to find",
  datasheet
);
// Get the id of field
const fieldId = field.id;

const records = await datasheet.getRecordsAsync();
const finalData = [];

// traverse records
for (let record of records) {
  const recordId = record.id;
  // Get the data of the specified field of the record as cellValue
  const cellValue = record.getCellValueString(fieldId);

  // If the acquired data is empty, jump out of this cycle and directly execute the next time
  if (cellValue == null) continue;

  // Replace findText in cellValue with replaceText, and use a new variable - newCellValue
  const newCellValue = cellValue.replaceAll(findText, replaceText);
  // Determine whether the data before replacement is consistent with the data after replacement
  if (cellValue !== newCellValue) {
    // Inconsistency means that the corresponding record in the table needs to replace cellValue with newCellValue,
    finalData.push({
      id: recordId,
      valuesMap: { [fieldId]: newCellValue },
    });
  }
}

// Determine whether there is data in finalData, if there is no data, there is no need to replace
if (finalData.length) {
  await datasheet.updateRecordsAsync(finalData);
  output.text("The replacement is complete!");
} else {
  output.text("No data found to be replaced");
}

```

Summary

Congratulations! you have implemented a "Find and Replace" widget with no more than 30 lines of code.

In addition, we have prepared more [examples](#) for you to further understand the purpose of script and more ways of writing.

Through this tutorial, I believe you have learned the usage of some APIs, and there are more [APIs](#) waiting for you to explore and discover more usages.

[Previous](#)

[« How to install](#)

[Next](#)

[Code example »](#)

Developer Center

[Development guide](#)

[API Reference](#)

Social

[Twitter](#) 

More

[Homepage](#) 

[Help Center](#) 

[GitHub](#) 