ipysheet Documentation

Release 0.4.1

Maarten Breddels

May 17, 2019

Contents

1 Installation **1**

2 Getting started **3**

3 Events **5**

4 Cell ranges **7**
4.1 Calculations 7

5 Renderers **9**
5.1 API reference 10

Python Module Index **19**

CHAPTER 1

Installation

With conda:

```
$ conda install -c conda-forge ipysheet
```

With pip:

```
$ pip install ipysheet
```

To make it work for Jupyter lab:

```
$ jupyter labextension install ipysheet
```

If you have notebook 5.2 or below, you also need to execute:

```
$ jupyter nbextension enable --py --sys-prefix ipysheet
```

CHAPTER 2

Getting started

Although ipysheet contains an object oriented interface, we recomment using the "state machine" based interface, similar to matplotlib's pyplot/pylab interface. Comparible to matplotlib pylab interface, this interface keeps track of the current sheet. Using the *cell* function, *Cell* widgets are added to the current sheet.

Importing ipysheet and invoking the *sheet* function will create the default spreadsheet widget. The function returns a *Sheet* instance, leaving that expression as a last statement of a code cell will display it, otherwise use `display(sheet)`.

Note that this documentation is a Jupyter notebook, and you can try it out directly on Binder:

[1]:
```
import ipysheet
sheet = ipysheet.sheet()
sheet
```

```
Sheet(columns=5, layout=Layout(height='auto', width='auto'), rows=5)
```

Using the *cell* function, we can create *Cell* widgets that are directly added to the current sheet.

[2]:
```
sheet = ipysheet.sheet(rows=3, columns=4)
cell1 = ipysheet.cell(0, 0, 'Hello')
cell2 = ipysheet.cell(2, 0, 'World')
cell_value = ipysheet.cell(2,2, 42.)
sheet
```

```
Sheet(cells=(Cell(column_end=0, column_start=0, row_end=0, row_start=0, type='text',␣
↪value='Hello'), Cell(colu...
```

CHAPTER 3

Events

Using link or observe we can link widgets together, or attach event handlers

Note: The examples below contain event handler written in Python that needs a running kernel, they will not work in the pure html documentation. They do work in binder!

[3]:
```
import ipywidgets as widgets
sheet = ipysheet.sheet(rows=3, columns=2, column_headers=False, row_headers=False)
cell_a = ipysheet.cell(0, 1, 1, label_left='a')
cell_b = ipysheet.cell(1, 1, 2, label_left='b')
cell_sum = ipysheet.cell(2, 1, 3, label_left='sum', read_only=True)

# create a slider linked to cell a
slider = widgets.FloatSlider(min=-10, max=10, description='a')
widgets.jslink((cell_a, 'value'), (slider, 'value'))

# changes in a or b should trigger this function
def calculate(change):
    cell_sum.value = cell_a.value + cell_b.value

cell_a.observe(calculate, 'value')
cell_b.observe(calculate, 'value')


widgets.VBox([sheet, slider])
```

```
VBox(children=(Sheet(cells=(Cell(column_end=1, column_start=1, row_end=0, row_start=0,
↪ type='numeric', value=1...
```

CHAPTER 4

Cell ranges

Instead of referring to a single cell, we can also refer to cell ranges, rows and columns.

[4]:
```
sheet = ipysheet.sheet(rows=5, columns=4)
row = ipysheet.row(0, [0, 1, 2, 3], background_color="red")
column = ipysheet.column(1, ["a", "b", "c", "d"], row_start=1, background_color="green
↪")
cells = ipysheet.cell_range([["hi", "ola"], ["ciao", "bonjour"], ["hallo", "guten tag
↪"]],
                            row_start=1, column_start=2, background_color="yellow")
sheet
```

```
Sheet(cells=(Cell(column_end=3, column_start=0, row_end=0, row_start=0, squeeze_
↪column=False, style={'backgrou...
```

4.1 Calculations

Since this is such a common pattern, a helper decorator *calculation* is provided, shortening the above code considerably.

[5]:
```
import ipywidgets as widgets
sheet = ipysheet.sheet(rows=3, columns=2, column_headers=False, row_headers=False)
cell_a = ipysheet.cell(0, 1, 1, label_left='a')
cell_b = ipysheet.cell(1, 1, 2, label_left='b')
cell_sum = ipysheet.cell(2, 1, 3, label_left='sum', read_only=True)

# create a slider linked to cell a
slider = widgets.FloatSlider(min=-10, max=10, description='a')
widgets.jslink((cell_a, 'value'), (slider, 'value'))

@ipysheet.calculation(inputs=[cell_a, cell_b], output=cell_sum)
def calculate(a, b):
```

(continues on next page)

(continued from previous page)

```
    return a + b

widgets.VBox([sheet, slider])
```

```
VBox(children=(Sheet(cells=(Cell(column_end=1, column_start=1, row_end=0, row_start=0,
↪ type='numeric', value=1...
```

CHAPTER 5

Renderers

ipysheet is build on Handsontable, which allows custom renderers, which we also support.

[6]:
```
jscode_renderer_negative = """function (value) {
  return {
    backgroundColor: value < 0 ?  'red' : 'green'
  };
}
"""
ipysheet.renderer(code=jscode_renderer_negative, name='negative');
```

[7]:
```
import random
s = ipysheet.sheet(rows=3, columns=4)
data = [[random.randint(-10, 10) for j in range(4)] for j in range(3)]
ipysheet.cell_range(data, renderer='negative')
s
```

```
Sheet(cells=(Cell(column_end=3, column_start=0, renderer='negative', row_end=2, row_
↪start=0, squeeze_column=Fa...
```

If flexx is installed, Python code can be transpiled to JavaScript at runtime.

[8]:
```
def renderer_negative(value):
    return {
        'backgroundColor': 'orange' if value < 0 else ''
    }
ipysheet.renderer(code=renderer_negative, name='negative_transpiled');
```

[9]:
```
import random
s = ipysheet.sheet(rows=3, columns=4)
data = [[random.randint(-10, 10) for j in range(4)] for j in range(3)]
ipysheet.cell_range(data, renderer='negative_transpiled')
s
```

```
Sheet(cells=(Cell(column_end=3, column_start=0, renderer='negative_transpiled', row_
↪end=2, row_start=0, squeez...
```

5.1 API reference

Note that everything is accessible from the ipysheet namespace. For example, you can do `from ipysheet import sheet, from_dataframe`

5.1.1 ipysheet.easy

Easy context-based interface for generating a sheet and cells.

Comparable to matplotlib pylab interface, this interface keeps track of the current sheet. Using the `cell` function, `Cell` widgets are added to the current sheet.

ipysheet.easy.**sheet**(*key=None*, *rows=5*, *columns=5*, *column_width=None*, *row_headers=True*, *column_headers=True*, *stretch_headers='all'*, *cls=<class 'ipysheet.sheet.Sheet'>*, ***kwargs*)

Creates a new `Sheet` instance or retrieves one registered with key, and sets this as the 'current'.

If the key argument is given, and no sheet is created before with this key, it will be registered under this key. If this function is called again with the same key argument, that `Sheet` instance will be returned.

Parameters

- **key** (*string*) – If not used before, register the sheet under this key. If used before, return the previous `Sheet` instance registered with this key.
- **rows** (*int*) – The number of rows in the sheet
- **columns** (*int*) – The number of columns in the sheet
- **row_headers** (*bool, list*) – Either a boolean specifying if row headers should be displayed or not, or a list of strings containing the row headers
- **column_headers** (*bool, list*) – Either a boolean specifying if column headers should be displayed or not, or a list of strings containing the column headers

Returns The new `Sheet` widget, or if key is given, the previously created sheet registered with this key.

Example

```
>>> from ipysheet import sheet, current
>>>
>>> s1 = sheet('key1')
>>> s2 = sheet('key2')
>>>
>>> assert s2 is current()
>>> assert s1 is sheet('key1')
>>> assert s1 is current()
```

ipysheet.easy.**current**()

Returns the current `Sheet` instance

ipysheet.easy.**cell**(*row*, *column*, *value=0.0*, *type=None*, *color=None*, *background_color=None*, *font_style=None*, *font_weight=None*, *style=None*, *label_left=None*, *choice=None*, *read_only=False*, *numeric_format='0.000'*, *date_format='YYYY/MM/DD'*, *renderer=None*, ***kwargs*)

Adds a new `Cell` widget to the current `Sheet`

Parameters

- **row** (*int*) – Zero based row index where to put the cell in the sheet
- **column** (*int*) – Zero based column index where to put the cell in the sheet
- **value** (*int, float, string, bool, Widget*) – The value of the cell
- **type** (*string*) – Type of cell, options are: text, numeric, checkbox, dropdown, numeric, date, widget. If type is None, the type is inferred from the type of the value being passed, numeric (float or int type), boolean (bool type), widget (any widget object), or else text. When choice is given the type will be assumed to be dropdown. The types refer (currently) to the handsontable types: https://handsontable.com/docs/6.2.2/demo-custom-renderers.html
- **color** (*string*) – The text color in the cell
- **background_color** (*string*) – The background color in the cell
- **read_only** (*bool*) – Whether the cell is editable or not
- **numeric_format** (*string*) – Numbers format
- **date_format** (*string*) – Dates format
- **time_format** (*string*) – Time format
- **renderer** (*string*) – Renderer name to use for the cell

Returns The new `Cell` widget.

Example

```
>>> from ipysheet import sheet, cell
>>>
>>> s1 = sheet()
>>> cell(0, 0, 36.)            # The Cell type will be 'numeric'
>>> cell(1, 0, True)           # The Cell type will be 'checkbox'
>>> cell(0, 1, 'Hello World!') # The Cell type will be 'text'
>>> c = cell(1, 1, True)
>>> c.value = False            # Dynamically changing the cell value at row=1,␣
↪column=1
```

`ipysheet.easy.`**`calculation`**(*inputs*, *output*, *initial_calculation=True*)

A decorator that assigns to output cell a calculation depending on the inputs

Parameters

- **inputs** (*list of widgets, or (widget, 'traitname') pairs*) – List of all widget, whose values (default 'value', otherwise specified by 'traitname') are input of the function that is decorated
- **output** (*widget or (widget, 'traitname')*) – The output of the decorator function will be assigned to output.value or output.<traitname>.
- **initial_calculation** (*bool*) – When True the calculation will be done directly for the first time.

Example

```
>>> from ipywidgets import IntSlider
>>> from ipysheet import cell, calculation
>>>
>>> a = cell(0, 0, value=1)
>>> b = cell(1, 0, value=IntSlider(value=2))
>>> c = IntSlider(max=56)
>>> d = cell(3, 0, value=1)
>>>
>>> @calculation(inputs=[a, (b, 'value'), (c, 'max')], output=d)
>>> def add(a, b, c):
>>>     return a + b + c
```

ipysheet.easy.**row**(*row*, *value*, *column_start=0*, *column_end=None*, *type=None*, *color=None*, *background_color=None*, *font_style=None*, *font_weight=None*, *style=None*, *choice=None*, *read_only=False*, *numeric_format='0.000'*, *date_format='YYYY/MM/DD'*, *renderer=None*, ***kwargs*)

Create a `Cell` widget, representing multiple cells in a sheet, in a horizontal row

Parameters

- **row** (*int*) – Zero based row index where to put the row in the sheet
- **value** (*list*) – The list of cell values representing the row
- **column_start** (*int*) – Which column the row will start, default 0.
- **column_end** (*int*) – Which column the row will end, default is the last.
- **type** (*string*) – Type of cell, options are: text, numeric, checkbox, dropdown, numeric, date, widget. If type is None, the type is inferred from the type of the value being passed, numeric (float or int type), boolean (bool type), widget (any widget object), or else text. When choice is given the type will be assumed to be dropdown. The types refer (currently) to the handsontable types: https://handsontable.com/docs/6.2.2/demo-custom-renderers.html
- **color** (*string*) – The text color in the cell
- **background_color** (*string*) – The background color in the cell
- **read_only** (*bool*) – Whether the cell is editable or not
- **numeric_format** (*string*) – Numbers format
- **date_format** (*string*) – Dates format
- **time_format** (*string*) – Time format
- **renderer** (*string*) – Renderer name to use for the cell

Returns The new `Cell` widget.

Example

```
>>> from ipysheet import sheet, row
>>>
>>> s1 = sheet()
>>> row(0, [1, 2, 3, 34, 5])                    # The Cell type will be 'numeric'
>>> row(1, [True, False, True], column_start=2) # The Cell type will be 'checkbox'
```

ipysheet.easy.**column**(*column*, *value*, *row_start=0*, *row_end=None*, *type=None*, *color=None*, *background_color=None*, *font_style=None*, *font_weight=None*, *style=None*, *choice=None*, *read_only=False*, *numeric_format='0.000'*, *date_format='YYYY/MM/DD'*, *renderer=None*, ***kwargs*)

Create a `Cell` widget, representing multiple cells in a sheet, in a vertical column

Parameters

- **column** (*int*) – Zero based column index where to put the column in the sheet
- **value** (*list*) – The list of cell values representing the column
- **row_start** (*int*) – Which row the column will start, default 0.
- **row_end** (*int*) – Which row the column will end, default is the last.
- **type** (*string*) – Type of cell, options are: text, numeric, checkbox, dropdown, numeric, date, widget. If type is None, the type is inferred from the type of the value being passed, numeric (float or int type), boolean (bool type), widget (any widget object), or else text. When choice is given the type will be assumed to be dropdown. The types refer (currently) to the handsontable types: https://handsontable.com/docs/6.2.2/demo-custom-renderers.html
- **color** (*string*) – The text color in the cell
- **background_color** (*string*) – The background color in the cell
- **read_only** (*bool*) – Whether the cell is editable or not
- **numeric_format** (*string*) – Numbers format
- **date_format** (*string*) – Dates format
- **time_format** (*string*) – Time format
- **renderer** (*string*) – Renderer name to use for the cell

Returns The new `Cell` widget.

Example

```
>>> from ipysheet import sheet, column
>>>
>>> s1 = sheet()
>>> column(0, [1, 2, 3, 34, 5])                    # The Cell type will be 'numeric'
>>> column(1, [True, False, True], row_start=2) # The Cell type will be 'checkbox'
```

ipysheet.easy.**cell_range**(*value*, *row_start=0*, *column_start=0*, *row_end=None*, *column_end=None*, *transpose=False*, *squeeze_row=False*, *squeeze_column=False*, *type=None*, *color=None*, *background_color=None*, *font_style=None*, *font_weight=None*, *style=None*, *choice=None*, *read_only=False*, *numeric_format='0.000'*, *date_format='YYYY/MM/DD'*, *renderer=None*, ***kwargs*)

Create a `Cell` widget, representing multiple cells in a sheet

Parameters

- **value** (*list*) – The list of cell values representing the range
- **row_start** (*int*) – Which row the range will start, default 0.
- **column_start** (*int*) – Which column the range will start, default 0.
- **row_end** (*int*) – Which row the range will end, default is the last.

- **column_end** (*int*) – Which column the range will end, default is the last.
- **transpose** (*bool*) – Whether to interpret the value array as value[column_index][row_index] or not.
- **squeeze_row** (*bool*) – Take out the row dimensions, meaning only value[column_index] is used.
- **squeeze_column** (*bool*) – Take out the column dimensions, meaning only value[row_index] is used.
- **type** (*string*) – Type of cell, options are: text, numeric, checkbox, dropdown, numeric, date, widget. If type is None, the type is inferred from the type of the value being passed, numeric (float or int type), boolean (bool type), widget (any widget object), or else text. When choice is given the type will be assumed to be dropdown. The types refer (currently) to the handsontable types: https://handsontable.com/docs/6.2.2/demo-custom-renderers.html
- **color** (*string*) – The text color in the cell
- **background_color** (*string*) – The background color in the cell
- **read_only** (*bool*) – Whether the cell is editable or not
- **numeric_format** (*string*) – Numbers format
- **date_format** (*string*) – Dates format
- **time_format** (*string*) – Time format
- **renderer** (*string*) – Renderer name to use for the cell

Returns The new `Cell` widget.

Example

```
>>> from ipysheet import sheet, cell_range
>>>
>>> s1 = sheet()
>>> cell_range([[1, 2, 3, 34, 5], [6, 7, 8, 89, 10]])
```

ipysheet.easy.**hold_cells**(**args*, ***kwds*)

Hold adding any cell widgets until leaving this context.

This may give a better performance when adding many cells.

Example

```
>>> from ipysheet import sheet, cell, hold_cells
>>>
>>> sheet(rows=10,columns=10)
>>> with hold_cells()
>>>     for i in range(10):
>>>         for j in range(10):
>>>             cell(i, j, value=i * 10 + j)
>>> # at this line, the Cell widgets are added
```

ipysheet.easy.**renderer**(*code*, *name*)

Create a `Renderer` widget

Parameters

- **code** (*string or code or function object*) – If a string object, it is assumed to be a JavaScript snippet, else it is assumed to be a function or code object and will be transpiled to javascript using flexxui/pscript.
- **name** (*string*) – Name of the renderer

Returns The new `Renderer` widget.

Example

```
>>> from ipysheet import sheet, renderer, cell
>>>
>>> s1 = sheet()
>>>
>>> def renderer_negative(instance, td, row, col, prop, value, cellProperties):
>>>     Handsontable.renderers.TextRenderer.apply(this, arguments);
>>>     if value < 0:
>>>         td.style.backgroundColor = 'orange'
>>>     else:
>>>         td.style.backgroundColor = ''
>>>
>>> renderer(code=renderer_negative, name='negative');
>>> cell(0, 0, 36, renderer='negative')  # Will be white
>>> cell(1, 0, -36, renderer='negative') # Will be orange
```

5.1.2 ipysheet.pandas_loader

ipysheet.pandas_loader.**from_dataframe**(*dataframe*)

Helper function for creating a sheet out of a Pandas DataFrame

Parameters **dataframe** (*Pandas DataFrame*) –

Returns `Sheet` widget

Example

```
>>> import numpy as np
>>> import pandas as pd
>>> from ipysheet import from_dataframe
>>>
>>> dates = pd.date_range('20130101', periods=6)
>>> df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
>>>
>>> sheet = from_dataframe(df)
>>> display(sheet)
```

ipysheet.pandas_loader.**to_dataframe**(*sheet*)

Helper function for creating a Pandas DataFrame out of a sheet

Parameters **sheet** (*Sheet widget*) –

Returns A Pandas DataFrame

Example

```
>>> import ipysheet
>>>
>>> sheet = ipysheet.sheet(rows=3, columns=4)
>>> ipysheet.cell(0, 0, 'Hello')
>>> ipysheet.cell(2, 0, 'World')
>>>
>>> df = to_dataframe(sheet)
>>> display(df)
```

5.1.3 ipysheet.numpy_loader

ipysheet.numpy_loader.**from_array**(*array*)

Helper function for creating a sheet out of a NumPy Array

Parameters **array** (*NumPy Array*) –

Returns Sheet widget

Example

```
>>> import numpy as np
>>> from ipysheet import from_array
>>>
>>> arr = np.random.randn(6, 26)
>>>
>>> sheet = from_array(arr)
>>> display(sheet)
```

ipysheet.numpy_loader.**to_array**(*sheet*)

Helper function for creating a NumPy Array out of a sheet

Parameters **sheet** (*Sheet widget*) –

Returns A NumPy Array

Example

```
>>> import ipysheet
>>>
>>> sheet = ipysheet.sheet(rows=3, columns=4)
>>> ipysheet.cell(0, 0, 'Hello')
>>> ipysheet.cell(2, 0, 'World')
>>>
>>> arr = to_array(sheet)
>>> display(arr)
```

5.1.4 ipysheet.sheet

class ipysheet.sheet.**Sheet**(***kwargs*)

Bases: ipywidgets.widgets.domwidget.DOMWidget

cells
An instance of a Python tuple.

column_headers
A trait type representing a Union type.

column_resizing
A boolean (True, False) trait.

column_width
A trait type representing a Union type.

columns
A casting version of the int trait.

named_cells
An instance of a Python dict.

row_headers
A trait type representing a Union type.

row_resizing
A boolean (True, False) trait.

rows
A casting version of the int trait.

search_token
A trait for unicode strings.

stretch_headers
A trait for unicode strings.

class ipysheet.sheet.**Cell**(***kwargs*)
Bases: ipywidgets.widgets.widget.Widget

choice
An instance of a Python list.

column_end
A casting version of the int trait.

column_start
A casting version of the int trait.

date_format
A trait for unicode strings.

name
A trait for unicode strings.

numeric_format
A trait for unicode strings.

read_only
A boolean (True, False) trait.

renderer
A trait for unicode strings.

row_end
A casting version of the int trait.

row_start
A casting version of the int trait.

squeeze_column
A boolean (True, False) trait.

squeeze_row
A boolean (True, False) trait.

style
An instance of a Python dict.

time_format
A trait for unicode strings.

transpose
A boolean (True, False) trait.

type
A trait for unicode strings.

value
A trait which allows any value.

class ipysheet.sheet.**Range**(***kwargs*)
Bases: ipywidgets.widgets.widget.Widget

value
A trait type representing a Union type.

Python Module Index

i

ipysheet.easy, 10
ipysheet.numpy_loader, 16
ipysheet.pandas_loader, 15
ipysheet.sheet, 16

Index

C

calculation() (*in module ipysheet.easy*), 11
Cell (*class in ipysheet.sheet*), 17
cell() (*in module ipysheet.easy*), 10
cell_range() (*in module ipysheet.easy*), 13
cells (*ipysheet.sheet.Sheet attribute*), 16
choice (*ipysheet.sheet.Cell attribute*), 17
column() (*in module ipysheet.easy*), 12
column_end (*ipysheet.sheet.Cell attribute*), 17
column_headers (*ipysheet.sheet.Sheet attribute*), 17
column_resizing (*ipysheet.sheet.Sheet attribute*), 17
column_start (*ipysheet.sheet.Cell attribute*), 17
column_width (*ipysheet.sheet.Sheet attribute*), 17
columns (*ipysheet.sheet.Sheet attribute*), 17
current() (*in module ipysheet.easy*), 10

D

date_format (*ipysheet.sheet.Cell attribute*), 17

F

from_array() (*in module ipysheet.numpy_loader*), 16
from_dataframe() (*in module ipysheet.pandas_loader*), 15

H

hold_cells() (*in module ipysheet.easy*), 14

I

ipysheet.easy (*module*), 10
ipysheet.numpy_loader (*module*), 16
ipysheet.pandas_loader (*module*), 15
ipysheet.sheet (*module*), 16

N

name (*ipysheet.sheet.Cell attribute*), 17
named_cells (*ipysheet.sheet.Sheet attribute*), 17
numeric_format (*ipysheet.sheet.Cell attribute*), 17

R

Range (*class in ipysheet.sheet*), 18
read_only (*ipysheet.sheet.Cell attribute*), 17
renderer (*ipysheet.sheet.Cell attribute*), 17
renderer() (*in module ipysheet.easy*), 14
row() (*in module ipysheet.easy*), 12
row_end (*ipysheet.sheet.Cell attribute*), 17
row_headers (*ipysheet.sheet.Sheet attribute*), 17
row_resizing (*ipysheet.sheet.Sheet attribute*), 17
row_start (*ipysheet.sheet.Cell attribute*), 17
rows (*ipysheet.sheet.Sheet attribute*), 17

S

search_token (*ipysheet.sheet.Sheet attribute*), 17
Sheet (*class in ipysheet.sheet*), 16
sheet() (*in module ipysheet.easy*), 10
squeeze_column (*ipysheet.sheet.Cell attribute*), 18
squeeze_row (*ipysheet.sheet.Cell attribute*), 18
stretch_headers (*ipysheet.sheet.Sheet attribute*), 17
style (*ipysheet.sheet.Cell attribute*), 18

T

time_format (*ipysheet.sheet.Cell attribute*), 18
to_array() (*in module ipysheet.numpy_loader*), 16
to_dataframe() (*in module ipysheet.pandas_loader*), 15
transpose (*ipysheet.sheet.Cell attribute*), 18
type (*ipysheet.sheet.Cell attribute*), 18

V

value (*ipysheet.sheet.Cell attribute*), 18
value (*ipysheet.sheet.Range attribute*), 18