bento-dash

Release 0.0.9

Derek Larson

Sep 04, 2020

PACKAGE DOCUMENTATION

1 Introduction **1**

2 Quickstart **3**

2.1 Installation . . . 3
2.2 Run Demo . . . 3
2.3 Modify Demo . . . 3
2.4 Fallback . . . 4
2.5 Bento Builder . . . 4

3 Concepts within Bento **5**

3.1 Data . . . 5
3.2 Pages . . . 5
3.3 Banks . . . 6
3.4 Connections . . . 6
3.5 Layout . . . 6

4 Beginner Tutorial **7**

4.1 Features . . . 7
4.2 Data . . . 8
4.3 Environment . . . 8
4.4 Descriptor . . . 9

5 API Reference **13**

5.1 Bento class . . . 13
5.2 Descriptor schema . . . 14
5.3 Utility functions . . . 16

Python Module Index **17**

Index **19**

CHAPTER

ONE

INTRODUCTION

Bento is a tool for building interactive dashboard applications, powered by a templating engine built on Plotly Dash. It lets you write a high-level description of your desired interactive dashboard and generates the application code for you. By providing a set of common building blocks and abstracting away some of the more complicated aspects, it aims to flatten the learning curve of dashboarding!

Bento follows a few principles in its goal of improving user productivity:

- Focus on one powerful, high-level object called the **descriptor**
- Allow users a selection of ready-made building blocks called **banks**
- Automatically handle component interactivity infrastructure
- Provide a simple theming and layout experience

An analogy can be drawn between Bento and desktop computer hardware. A motherboard ties together several different components in a way such that assembly can be performed without advanced knowledge of each piece. The user is just tasked with understanding what they want out of the finished product. In a similar way, creating the Bento descriptor should be as simple as understanding what you want the dashboard to do and then finding the right banks from the catalog.

For some inspiration, check out some sample Bento apps in this gallery.

To get a sense of what Plotly Dash can do, visit their app gallery.

CHAPTER

TWO

QUICKSTART

To get started with Bento, we're going to walk through a few quick steps:

- Installation
- Run the demo dashboard
- Modify the demo
- Fall back to straight Plotly Dash
- Clone Bento Builder

2.1 Installation

Dependencies: Python 3.7+

Bento is available on PyPI, and the latest version can be installed with:

```
$ pip3 install bento-dash
```

2.2 Run Demo

To make sure everything is working, try running the demo app:

```
$ bento-demo
```

And navigate in your browser to *localhost:8050*.

2.3 Modify Demo

To get a quick taste of the Bento experience, now run the following:

```
$ python3 -m bento.dashboards.demo
$ python3 -m bento.launcher
```

The first will write out a copy of the demo descriptor to your current dir. You can then open that up and modify it while the server from the second command is up.

It'll be hard to know what to change, so here's a few quick things to try:

- Comment out the `"theme": "dark"` line (Refresh your browser so the CSS updates!)
- Comment out and of the pages from the descriptor top-level
- Reorder the layouts
- Comment out any bank
- Disconnect a bank by removing it from the `connections`

2.4 Fallback

At any point, if you've started an app with Bento but can't figure out how to go further, you can always continue by editing the base app code. After either of the last steps, you will have a `bento_app.py` file plus associated files in `assets/` (check the log output while running Bento). These can be edited, and then you can run the changes with:

```
$ python3 bento_app.py
```

Note: Bento does make use of several utility functions included in the bento Python package. To fully be independent, you'd need to copy those over as well.

2.5 Bento Builder

If you've jumped through the above hoops and are looking for a way to develop efficiently, I recommend cloning/templating from the Bento Builder repo! Some quickstart steps are available in the readme.

CHAPTER

THREE

CONCEPTS WITHIN BENTO

These two example dashboard descriptors will be useful for reference:

- Simple
- Demo

3.1 Data

Everything begins with data, and Bento doesn't try to assist with the data processing. It is assumed you will supply your dashboard with prepared data in a pandas DataFrame. This helps decouple your visualization code from any data preparation code. However, Bento does inform the dashboard by what's in the data. For example, dropdown menus for filters can be automatically populated by what is in the dataframe.

You define all datasets used in the dashboard up front in the "data" key of the descriptor. Each dataset needs a unique ID (uid) and a module that contains the code to load it. In the *Simple* example, we loaded the data by the uid "covid" via the module `bento.sample_data.covid` which contains a method "load" that returns an object containing the dataframe.

In your generated Bento app, the data is stored in a global variable and accessed by key. This tends to be simplest at the page level, through the "dataid" key. This can be overridden at the bank level, however, if needed.

3.2 Pages

To Bento, a page is just about what you'd expect: everything associated with a given URL. Bento tries to make page navigation easy out of the box, with an autogenerated appbar. The downside to this: you still have to define your "pages" even if you want a single-page app.

Overall, the page definition is rather easy–each page needs a unique ID. The main descriptor "pages" key defines a dictionary with these unique IDs as keys and the individual page descriptors as values.

The *Simple* example demonstrates a single page while the *Demo* has multiple.

3.3 Banks

The banks are the building blocks of your Bento app.

You define your banks under a page's "banks" key, each with a uid and a dictionary containing at least the "type" key, which would be chosen from the list below.

3.4 Connections

One of the trickiest parts of making an interactive dashboard in Plotly Dash is getting components to properly work together. At the same time, it's one of the areas which tends to be the most repetitive once you understand it. This is a big part of what makes Bento useful. Bento abstracts away any need for writing callbacks by defining "connections": which banks feed which other banks. If you want a bank to affect another one, connect them.

In a page descriptor, the "connections" key is just the set of "source" bank uids which defines, for each, the set of "sink" bank uids. In the *Simple* example, our axis_controls named "axis" feeds our graph called "trend".

3.5 Layout

This is the easiest part. Just write a 2D-list of the bank ids, imagining a grid. Each page descriptor should have this in a key "layout", such as in *Simple* where we have two rows, one bank per row.

Bento uses the CSS Grid system to size the banks, and also comes with a sense of how big each bank should be depending on its inputs. For example, an axis_control bank might be 2 rows by 2 columns (2x2) for 1 axis, and it will try for 2x6 if you have 3 axes defined. If you try to pack in a lot in a row, Bento will trim from each until they fit.

CHAPTER

FOUR

BEGINNER TUTORIAL

This is intended as a fast, but not too furious, introduction to the Bento system. You're going build a functional, interactive dashboard in a matter of minutes. Not to worry, you will be handed all of the pieces you need with instructions on where they go and a brief explanation of how the pieces fit. And, each step will give a visual result which you can check against some of the checkpoint images. The goal is to convey the fundamentals of the system without going into details.

As a straightforward example, let's build a dashboard that visualizes historical stock prices for a few large tech companies. As perhaps one of the most commonly experienced chart types in the world, most people should have some idea of what to expect here. You can see what we're gonna end up with by running `$ bento-demo` and clicking to the stock page on the app bar. (Make sure you've followed the Bento Quickstart)

We can break this project up into four stages:

- Plan the basic features
- Prepare the data we are using
- Set up our Bento dev environment
- Write the descriptor (which we'll break out into pieces)

Note: For the foreseeable future (how long is that, really?), only 'Nix environments are gonna be supported (namely Mac and Linux) so we'll take some liberties in assuming that.

4.1 Features

What should our dashboard do? It's a tough question if you are trying to predict the end product–there are a lot of unknowns (Who is the target user? What interactions will work well? Will the dashboard get cluttered? What does Bento even support doing?) so let's just try to start with a good first draft. We'd like to at least be able to plot comparative traces of stock gains. For this, we'll need the ability to select the ticker symbols to show. A way to choose a period in time for comparison would also be useful. Lastly, we would also need a way to normalize the prices into percentage gains from the start of our interval.

This translates into 4 total Bento banks:

- A `selector` – a multi-selection dropdown tied stock ticker symbols
- A `date_control` – a two-sided slider bar, letting one choose a date interval
- An `analytics_set` – a toolkit that contains a normalization option
- Lastly, a standard `graph` for the plotting of the time series

4.2 Data

Bento leaves the data preparation to the user, by design. But for this example, we will simply use the sample stock dataset packaged with Bento (I suppose it is a stock stock dataset. . . apologies). Not only is this data prepared, but it is also supplied in the expected structure:

```
{
    "df": your_dataframe_object,
    "keys": ["key_column_1", "key_column_2"],
    "types": {"measurement_column_1": int}
}
```

The structure above contains the data as a Pandas DataFrame and adds descriptive info: identifying the columns that are keys (which help locate rows of interest) and for those that contain metrics with their types (usually date, float, int). The metadata helps Bento supply automated defaults, helping ensure your app works right away. For example, sample stock data metadata looks something like

```
{
    "df": <the DataFrame object>,
    "keys": ["symbol"],
    "types": {"open": float, "close": float, "volume": int}
}
```

Can you guess how Bento incorporates this?

Note: Exploring a sample dataset is straightforward, for example try:

```
>>> from bento.sample_data import stock
>>> dataset = stock.load()
>>> print(dataset['df'])
```

Ultimately, Bento requires a module that can be imported to load the data. So in this case that is `bento.sample_data.stock`, which we'll use shortly!

4.3 Environment

Now that you're palpably excited from visualizing the dashboard, it's best to channel some of that energy into setting up your development environment. This will save a lot of time and headache down the road. I recommend going through the Quickstart steps at the Bento Builder repo. Make sure `$ ./build.py simple_example -dbu` works, because we'll be using the same flags.

Once that is done, make a new directory called beginner and open a new file `descriptor.py` for editing in your favorite editor:

```
$ mkdir beginner
$ vi beginner/descriptor.py
```

Note: If you want some extra credit, add a version file to avoid a later warning

$ echo '__version__ = "0.0.1"' > beginner/_version.py

4.4 Descriptor

And now for the main course, let's write the Bento descriptor. This is the piece that really ties the room together.

4.4.1 Step 1

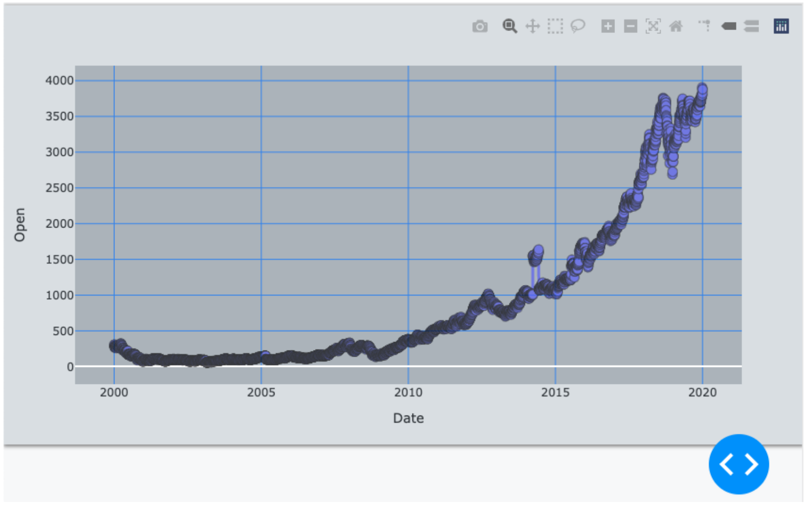
We're going to start with the most basic, functional skeleton to start. This entails declaring the data source (the stock sample data, as above) and defines a single page containing a single bank (of type "graph").

```
# beginner/descriptor.py
descriptor = {
    "data": {"stock": {"module": "bento.sample_data.stock"}},
    "pages": {"main_page": {"dataid": "stock", "banks": {"traces": {"type": "graph"}}}
↪}
}
```

This should get us a graph that displays our DataFrame blindly. Go ahead and paste that into the descriptor file. Now you can run the build script:

```
$ ./build.py beginner -dbu
```

And if all is good, you can go to localhost:7777 and see something (ugly) like this:



Don't worry, this will clean up pretty quick.

So here's what's important to know about what we did:

- We named the key in `data` something unique and relevant ("stock")
- The key matches the value of the `dataid` for our page
- The `module` for our data entry is set to the sample data
- Our page has a `banks` key with a valid dictionary of our single bank
- The `main_page` and `traces` strings just represent unique names we can make up

4.4.2 Step 2

Let's add a few quick aesthetic improvements:

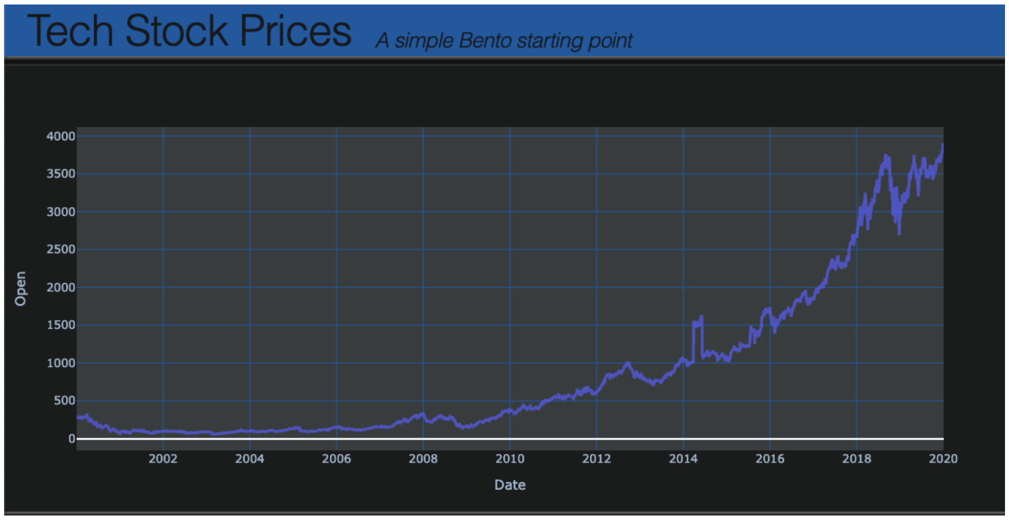
- Add an `appbar` to the descriptor, which sets a title/subtitle and contains any nav links
- Change the theme to `dark`
- Set the graph trace mode to lines
- Also let's break out the page definition from the main dictionary

Try them in any order by looking at the completed version below:

```python
# beginner/descriptor.py
main_page = {
    "dataid": "stock",
    "banks": {
        "traces": {"type": "graph", "mode": "lines"},
    },
}

descriptor = {
    "name": "beginner_tutorial",
    "theme": "dark",
    "appbar": {
        "title": "Tech Stock Prices",
        "subtitle": "A simple Bento starting point",
    },
    "data": {"stock": {"module": "bento.sample_data.stock"}},
    "pages": {"main": main_page},
}
```

The full result should look like:



4.4.3 Step 3

Now let's start cooking with gas. First, we'll add all the banks we had planned. Simply add these lines to the `banks` dict:

```
"analytics": {"type": "analytics_set"},
"interval": {"type": "date_slider", "variant": "range"},
"symbols": {"type": "selector", "columns": ["symbol"]},
```

You should now get some new blocks showing up, but they aren't very well-organized. As in, they are just stacked on top of each other, rather lazily. We can fix that by supplying a layout. This should be intuitive, just add this to the page dict and see if it makes sense:

```
"layout": [["symbols", "interval", "analytics"], ["traces"]],
```

Currently, Bento expects a 2-D array of bank IDs, but a generalization to N-D could be in the cards.

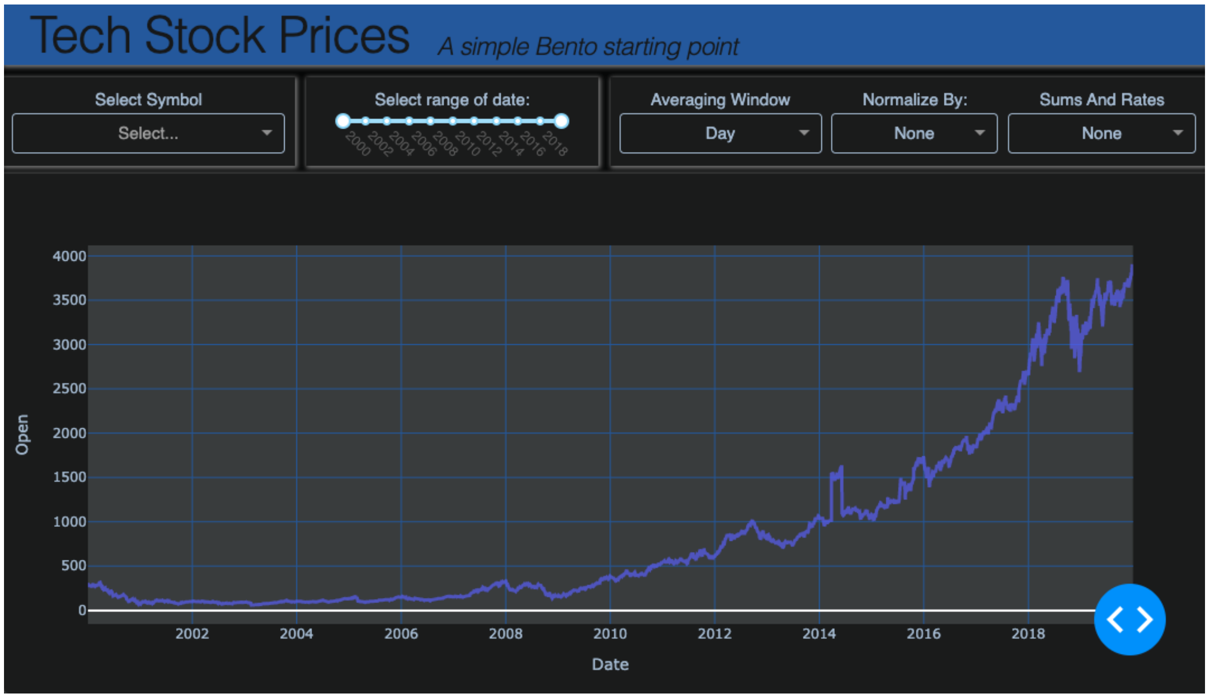
Perhaps you're also frustrated that these don't *do* anything yet. That's because we haven't told the app to connect the banks. This part is, I think, delightfully straightforward–just define the many-to-many graph of connections between banks. In this case, it's just the following new dictionary keyed into the page:

```
"connections": {
    "analytics": {"traces"},
    "interval": {"traces"},
    "symbols": {"traces"},
},
```

Now stuff should happen. And there was much rejoicing. If not, double-check against the full descriptor below

```
main_page = {
    "dataid": "stock",
    "banks": {
        "traces": {"type": "graph", "mode": "lines"},
        "analytics": {"type": "analytics_set"},
        "interval": {"type": "date_slider", "variant": "range"},
        "symbols": {"type": "selector", "columns": ["symbol"]},
    },
    "layout": [["symbols", "interval", "analytics"], ["traces"]],
    "connections": {
        "interval": {"traces"},
        "symbols": {"traces"},
        "analytics": {"traces"},
    },
}

descriptor = {
    "name": "beginner_tutorial",
    "theme": "dark",
    "appbar": {
        "title": "Tech Stock Prices",
        "subtitle": "A simple Bento starting point",
    },
    "data": {"stock": {"module": "bento.sample_data.stock"}},
    "pages": {"main": main_page},
}
```


Tech Stock Prices
A simple Bento starting point
Select Symbol
Select...
Select range of date:
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
Averaging Window
Day
Normalize By:
None
Sums And Rates
None
Open
4000
3500
3000
2500
2000
1500
1000
500
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
Date

CHAPTER

FIVE

API REFERENCE

- *Bento class*
- *Descriptor schema*
- *Utility functions*

5.1 Bento class

The fundamental Bento class that ties all of Bento's functionality together.

class `bento.bento.`**`Bento`**(*descriptor: Dict*, *init_only: bool = False*)

Acts as the gateway to all Bento functionality.

descriptor [dict] The descriptor contains all of the user-supplied information defining the desired dashboard. See the user guide for creating a descriptor.

init_only [bool] Supply True in order to halt the automatic processing of the descriptor. This can be useful for debugging or modifying the standard app-creation process.

`desc`

This stores the normalized version of the input descriptor.

Type dict

`data`

Contains the data as a dictionary. TODO

Type dict

`valid`

Whether the descriptor meets the schema. If False, will block processing.

Type bool

`context`

Specifies the app for consumption by a Jinja template.

Type dict

Examples

Simple:

```
>>> bento = Bento(my_descriptor)
>>> bento.write()
```

Advanced:

```
>>> bento = Bento(my_descriptor, init_only=True)
>>> test_page = bento.desc["pages"]["test"]
>>> bento.create_page("test", test_page)
>>> bento.context["pages"]["test"] = alternate_grid_method(test_page)
>>> bento.connect_page(test_page)
>>> bento.write(app_output="modified_test_layout.py")
```

`connect_page`(*page: Dict*)

Applies the requested connections for the page to the Jinja context

A page is a set of connected banks, and the connections are defined by a supplied directed graph (dict of sets) of bank_ids. For example, *{'axes': {'map', 'counter'}, 'colors': {'map'}}* tells us the axes bank should feed both the map and counter banks, while our colors bank should feed just the map.

`create_page`(*pageid: str*, *page: Dict*)

Generates and lays out all banks defined for a page and updates the context

A page is composed of a set of banks, arranged based on a supplied layout object (an array of 2+ dim).

`is_valid`(*descriptor: Dict*) → bool

Ensures the descriptor meets the Cerberus schema (see schema.py)

`normalize`(*descriptor: Dict*) → Dict

Auto-trims and -fills the descriptor.

- Removes any dangling bankids, assuming 'banks' keys as source of truth
- Generates full bankid (pageid + bankname)
- Handle most defaults here so they aren't scattered about

`write`(*app_output: str = 'bento_app.py'*, *css_folder: str = 'assets'*)

Creates all of the standard Bento output files.

This is a convenience wrapper that allows for one simple call to generate all of the application code for the output Bento app, usually a set of Python and CSS files.

5.2 Descriptor schema

To get a quick sense of the schema, I recommend looking at the following examples:

- A bare-bones dash - Run `python3 -m bento.dashboards.simple` then `python3 bento_app.py` to view
- The demo dash - Run `bento-demo` to see it in action

The full schema follows, which, for now, is a simple dump of the Cerberus schema:

```
# Page and bank ids, are strings of word characters only with length 2+
# No trailing, leading, or double underscores
bento_uid_regex = r"^(?!_|.*_$|.*__.*)[a-z0-9_]{2,}$"

# One or more words separated by a space
words_regex = r"^\w+( \w+)*$"

page_schema = {
    "banks": {
        "type": "dict",
        "required": True,
        "allow_unknown": True,
        "minlength": 1,
        "keysrules": {"type": "string", "regex": bento_uid_regex},
        "valuesrules": {
            "type": "dict",
            "allow_unknown": True,
            "schema": {
                "type": {"type": "string", "required": True, "regex": bento_uid_regex}
↪,
                "width": {"type": "integer"},
                "args": {"type": "dict"},
            },
        },
    },
    "connections": {"type": "dict"},
    "dataid": {"type": "string"},
    "layout": {"type": "list"},
    "sidebar": {"type": "list"},
    "title": {"type": "string"},
    "subtitle": {"type": "string"},
}

descriptor_schema = {
    "name": {"type": "string"},
    "theme": {"type": "string", "regex": words_regex},
    "theme_dict": {"type": "dict"},
    "appbar": {
        "type": "dict",
        "schema": {
            "title": {"type": "string"},
            "subtitle": {"type": "string"},
            "image": {"type": "string"},
        },
    },
    "data": {"type": "dict"},
    "show_help": {"type": "boolean"},
    "pages": {
        "type": "dict",
        "allow_unknown": True,
        "required": True,
        "minlength": 1,
        "keysrules": {"type": "string", "regex": bento_uid_regex},
        "valuesrules": {"type": "dict", "schema": page_schema},
    },
}
```

5.3 Utility functions

`bento.util.desnake(text)`

Turns underscores into spaces

PYTHON MODULE INDEX

b

bento.bento, 13

INDEX

B

Bento (*class in bento.bento*), 13
bento.bento
    module, 13

C

connect_page() (*bento.bento.Bento method*), 14
context (*bento.bento.Bento attribute*), 13
create_page() (*bento.bento.Bento method*), 14

D

data (*bento.bento.Bento attribute*), 13
desc (*bento.bento.Bento attribute*), 13
desnake() (*in module bento.util*), 16

I

is_valid() (*bento.bento.Bento method*), 14

M

module
    bento.bento, 13

N

normalize() (*bento.bento.Bento method*), 14

V

valid (*bento.bento.Bento attribute*), 13

W

write() (*bento.bento.Bento method*), 14