filestore: Experiment file tracker for NSLS-II beamlines

Daniel B. Allan(Brookhaven National Lab) ,Thomas A. Caswell(Brookhaven National Lab), Arman Arkilic(Brookhaven National Lab), Eric Dill(Brookhaven National Lab), Bob L. Dalesio(Brookhaven National Lab)

Overview

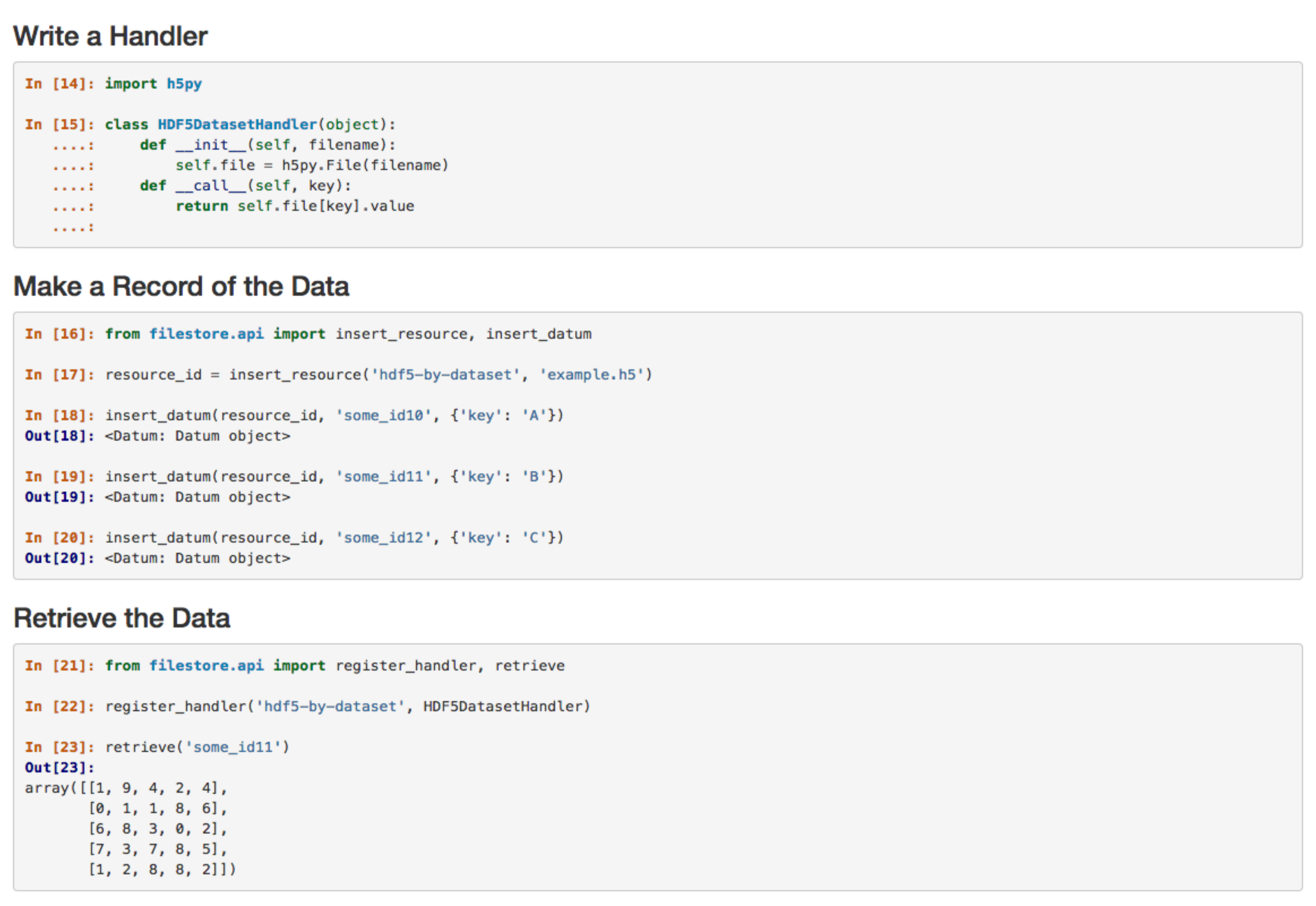
Each NSLS-II beamline can generate 72,000 data sets per day, over 2 M data sets in one year. The large amount of data files generated by our beamlines poses a massive file management challenge. In response to this challenge, we have developed filestore that provides users with an interface to stored data. By leveraging features of Python and MongoDB, filestore can store information regarding the location of a file, access and open the file, retrieve a given piece of data in that file, and provide users with a token, a unique identifier allowing them to retrieve each piece of data. Filestore does not interfere with the file source or the storage method and supports any file format, making data within files available for NSLS-II data analysis environment.

Why filestore?

Traditionally, all information regarding a experimental files have been kept in hand-written logbooks or written into experiment generated text files(e.g. spec files). Once the NSLS-I experiments were profiled, it was determined that file I/O was the biggest performance issue. This performance issue was due to opening each experiment file and searching the data file's URL within it. In order to eliminate this bottleneck, a database was designed to replace the text files.

The database of choice is MongoDB due to its flexibility and performance. Just like metadatastore, filestore has a set of required standard fields and allows users/data acquisition scripts to add any field on the fly. The unique identifier token and various fields can be used in order to query the data files and various handlers(that are available as a part of the filestore and/ or can be developed by the user) can be used to retrieve this data into data analysis environment. Since NSLS-II experiments generate 72K data sets per day and over 2 million data sets in one year, ability to query data with minimal file I/O in a flexible and performant fashion is extremely crucial in order to satisfy the needs of users in our state-of-the-art facility.

Example 2: Simple Handler for HDF5 File in GPFS



Write a Handler

```
In [14]: import h5py

In [15]: class HDF5DatasetHandler(object):
   ....:     def __init__(self, filename):
   ....:         self.file = h5py.File(filename)
   ....:     def __call__(self, key):
   ....:         return self.file[key].value
   ....:
```

Make a Record of the Data

```
In [16]: from filestore.api import insert_resource, insert_datum

In [17]: resource_id = insert_resource('hdf5-by-dataset', 'example.h5')

In [18]: insert_datum(resource_id, 'some_id10', {'key': 'A'})
Out[18]: <Datum: Datum object>

In [19]: insert_datum(resource_id, 'some_id11', {'key': 'B'})
Out[19]: <Datum: Datum object>

In [20]: insert_datum(resource_id, 'some_id12', {'key': 'C'})
Out[20]: <Datum: Datum object>
```

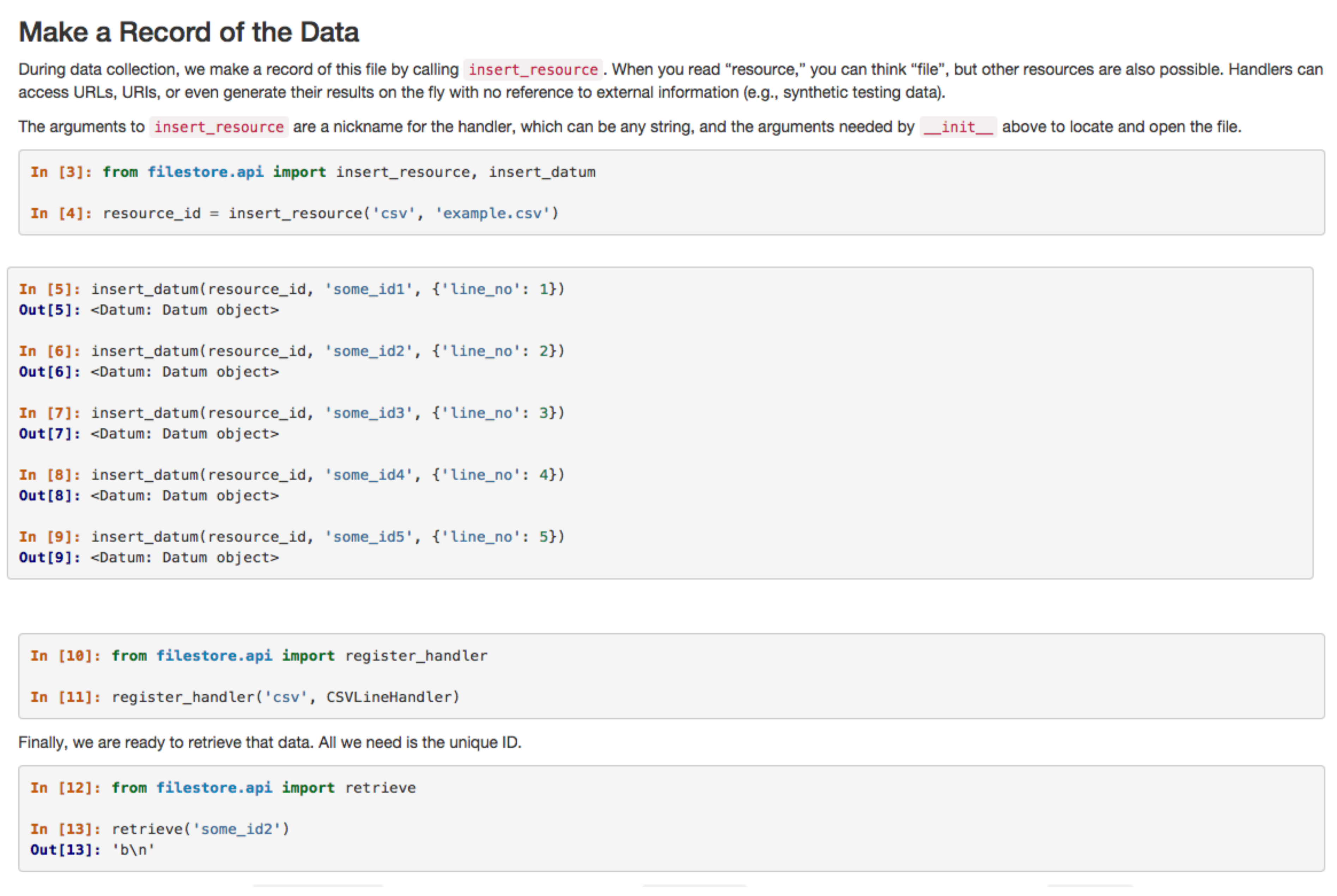
Retrieve the Data

```
In [21]: from filestore.api import register_handler, retrieve

In [22]: register_handler('hdf5-by-dataset', HDF5DatasetHandler)

In [23]: retrieve('some_id11')
Out[23]:
array([[1, 9, 4, 2, 4],
       [0, 1, 1, 8, 6],
       [6, 8, 3, 0, 2],
       [7, 3, 7, 8, 5],
       [1, 2, 8, 8, 2]])
```

What is filestore?

filestore is an interface to stored data. Files can be stored completely independent of filestore's acknowledgement. This means, filestore does not need to be aware of the file at time of creation. At some point, data acquisition script(or beamline user, using the externally facing API), should notify filestore about this data by providing two pieces of information: how to access and open the file (or, generically, "resource") and how to retrieve a given piece of data in that file. Just like metadatastore, filestore provides a token, a unique identifier, which one can use to retrieve each piece of data.

Example 1: Record and Retrieve a File



Make a Record of the Data

During data collection, we make a record of this file by calling `insert_resource`. When you read "resource," you can think "file", but other resources are also possible. Handlers can access URLs, URIs, or even generate their results on the fly with no reference to external information (e.g., synthetic testing data).

The arguments to `insert_resource` are a nickname for the handler, which can be any string, and the arguments needed by `__init__` above to locate and open the file.

```
In [3]: from filestore.api import insert_resource, insert_datum

In [4]: resource_id = insert_resource('csv', 'example.csv')
```

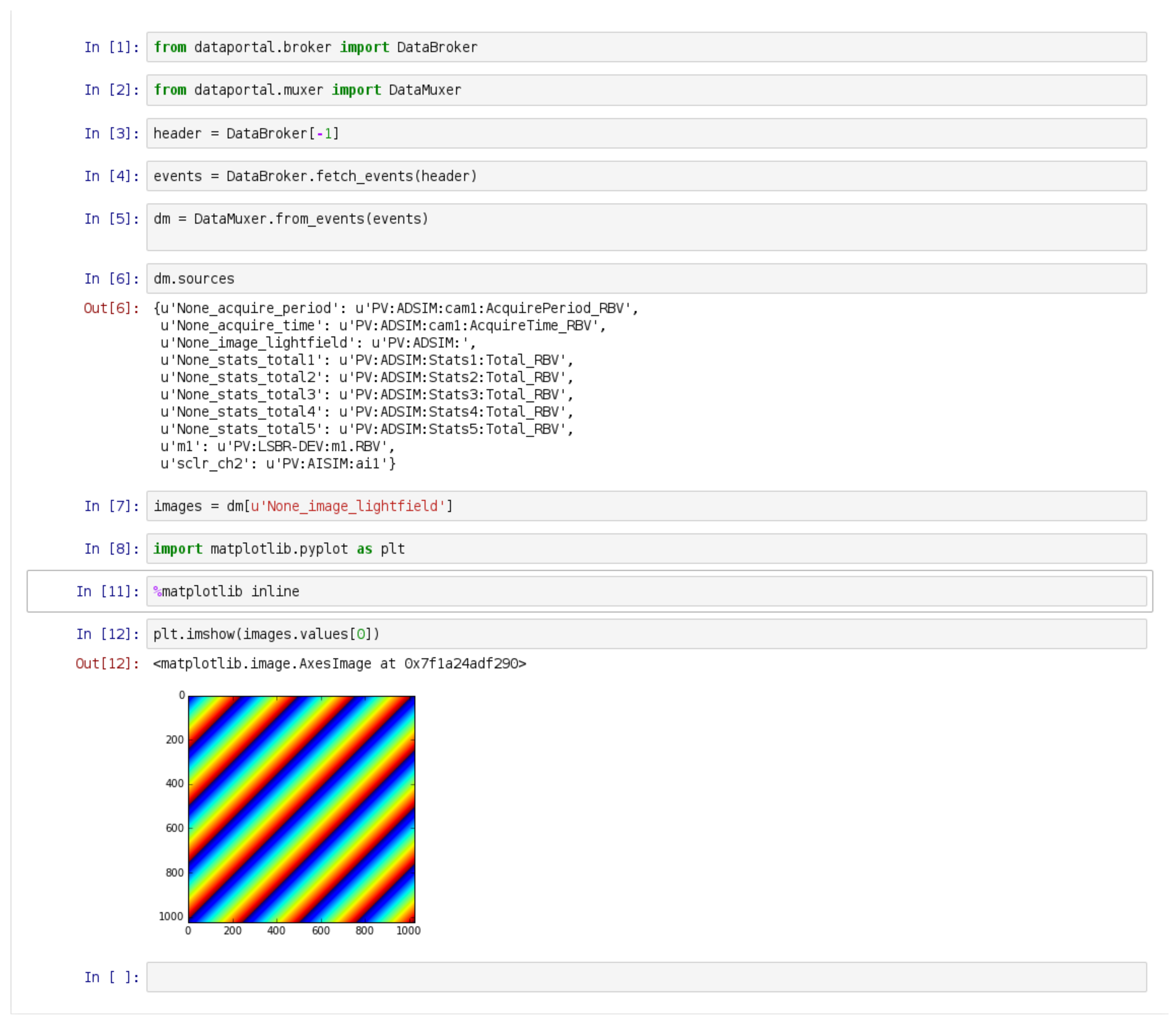
```
In [5]: insert_datum(resource_id, 'some_id1', {'line_no': 1})
Out[5]: <Datum: Datum object>

In [6]: insert_datum(resource_id, 'some_id2', {'line_no': 2})
Out[6]: <Datum: Datum object>

In [7]: insert_datum(resource_id, 'some_id3', {'line_no': 3})
Out[7]: <Datum: Datum object>

In [8]: insert_datum(resource_id, 'some_id4', {'line_no': 4})
Out[8]: <Datum: Datum object>

In [9]: insert_datum(resource_id, 'some_id5', {'line_no': 5})
Out[9]: <Datum: Datum object>
```

```
In [10]: from filestore.api import register_handler

In [11]: register_handler('csv', CSVLineHandler)
```

Finally, we are ready to retrieve that data. All we need is the unique ID.

```
In [12]: from filestore.api import retrieve

In [13]: retrieve('some_id2')
Out[13]: 'b\n'
```

Example 3: Retrieve Scan within IPython Notebook Session



```
In [1]: from dataportal.broker import DataBroker

In [2]: from dataportal.muxer import DataMuxer

In [3]: header = DataBroker[-1]

In [4]: events = DataBroker.fetch_events(header)

In [5]: dm = DataMuxer.from_events(events)

In [6]: dm.sources

Out[6]: {u'None_acquire_period': u'PV:ADSIM:cam1:AcquirePeriod_RBV',
         u'None_acquire_time': u'PV:ADSIM:cam1:AcquireTime_RBV',
         u'None_image_lightfield': u'PV:ADSIM:',
         u'None_stats_total1': u'PV:ADSIM:Stats1:Total_RBV',
         u'None_stats_total2': u'PV:ADSIM:Stats2:Total_RBV',
         u'None_stats_total3': u'PV:ADSIM:Stats3:Total_RBV',
         u'None_stats_total4': u'PV:ADSIM:Stats4:Total_RBV',
         u'None_stats_total5': u'PV:ADSIM:Stats5:Total_RBV',
         u'm1': u'PV:LSBR-DEV:m1.RBV',
         u'sclr_ch2': u'PV:AISIM:ai1'}

In [7]: images = dm[u'None_image_lightfield']

In [8]: import matplotlib.pyplot as plt

In [11]: %matplotlib inline

In [12]: plt.imshow(images.values[0])
Out[12]: <matplotlib.image.AxesImage at 0x7f1a24adf290>
```

```
In [ ]:
```

A U.S. Department of Energy laboratory managed by Brookhaven Science Asociates, Inc.
The NSLSII project is funded by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences