Example: downloading data from JSOC

Reference http://docs.sunpy.org/en/stable/guide/acquiring_data/jsoc.html

```
%load_ext autoreload
%autoreload 2

import astropy.units as u

from sunpy.net import Fido, attrs

# Suppress warnings of pandas, astropy
import warnings
warnings.simplefilter("ignore", category=FutureWarning)
from astropy.utils.exceptions import AstropyDeprecationWarning
warnings.simplefilter("ignore", category=AstropyDeprecationWarning)
```

To be clear, here we use

```
from sunpy.net import Fido, attrs
```

instead of examples in http://docs.sunpy.org/en/stable/guide/acquiring_data/jsoc.html

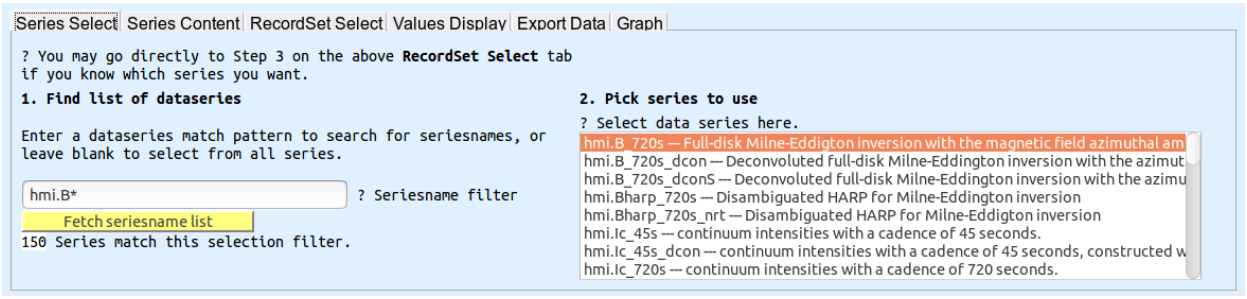
```
from sunpy.net import Fido, attrs as a
```

Getting information

`drms` - Python module for accessing HMI, AIA and MDI data

- https://github.com/kbg/drms
- http://drms.readthedocs.io/en/stable/tutorial.html

Series



Import the `drms` module and create an instance of the `drms.Client` class:

```
import drms
c = drms.Client()
```

List available `Series`:

```
c.series(r'aia\..*')  # regex, case-insensitive
```

```
['aia.flatfield',
 'aia.lev1',
 'aia.lev1_euv_12s',
 'aia.lev1_uv_24s',
 'aia.lev1_vis_1h',
 'aia.master_pointing3h',
 'aia.response',
 'aia.temperature_summary_300s']
```

```
c.series(r'hmi\.b.*')  # regex, case-insensitive
```

```
['hmi.B_720s',
 'hmi.B_720s_dcon',
 'hmi.B_720s_dconS',
 'hmi.Bharp_720s',
 'hmi.Bharp_720s_nrt',
 'hmi.b_135s',
 'hmi.b_720s_e15w1332_cea',
 'hmi.b_720s_e15w1332_cutout',
 'hmi.b_synoptic',
 'hmi.b_synoptic_small',
 'hmi.bmap_lowres_latlon_720s']
```

PrimeKeys

Find out the `PrimeKeys` supported in any `Series`:

```
c.pkeys('aia.lev1_euv_12s')  # case-insensitive
```

```
['T_REC', 'WAVELNTH']
```

```
c.pkeys('hmi.B_720s')  # case-insensitive
```

```
['T_REC']
```

Segments

Find out the `Segments` supported in any `Series`:

- 'aia.lev1_euv_12s'

```
si = c.info('aia.lev1_euv_12s')
si.segments
```

	type	units	protocol	dims	note
name					
image	int	None	link via lev1	None	AIA level 1 image
spikes	int	None	link via lev1	None	Cosmic ray information

```
si.segments.index.values  # To a `ndarray`
```

```
array(['image', 'spikes'], dtype=object)
```

```
si.segments.index[0]  # Use slice to pick a value
```

```
'image'
```

- 'hmi.b_720s'

```
si = c.info('hmi.b_720s')
si.segments[:4]  # Use slice to pick rows
```

	type	units	protocol	dims	note
name					
inclination	int	None	link via MDATA	None	Inclination
azimuth	int	None	link via MDATA	None	Azimuth before disambiguation
disambig	char		fits	4096x4096	Flag for 180 degree change in azimuth
field	int	None	link via MDATA	None	Field Strength

Basic usage

See http://docs.sunpy.org/en/stable/guide/acquiring_data/jsoc.html

> There are two ways of downloading JSOC data.
>
> - One way is using Sunpy's unified search interface, known as `Fido`.
>   `Fido` supplies a single, easy and consistent way to to obtain most forms of solar physics data.
> - An alternative way to fetch data from JSOC is by using the underlying `JSOCClient`.
>   This option can be preferred when the complex searches are to be made, or when you need to separate the staging and downloading steps, which is not supported by `Fido`.

Fido

Searching for data

```
response = Fido.search(
    attrs.jsoc.Time('2014-01-01T00:00:00', '2014-01-01T00:00:30'),
    attrs.jsoc.Notify('lydiazly@nju.edu.cn'),
    attrs.jsoc.Series('aia.lev1_euv_12s'),
    attrs.jsoc.Wavelength(304 * u.AA) | attrs.jsoc.Wavelength(171 * u.AA)
)
response
```

Results from 2 Providers:3 Results from the JSOCClient:*Table length=3*

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_4	304	2145
2014-01-01T00:00:13Z	SDO/AIA	AIA_4	304	2145
2014-01-01T00:00:25Z	SDO/AIA	AIA_4	304	2145

3 Results from the JSOCClient:*Table length=3*

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145
2014-01-01T00:00:13Z	SDO/AIA	AIA_3	171	2145
2014-01-01T00:00:25Z	SDO/AIA	AIA_3	171	2145

```
type(response)
```

```
sunpy.net.fido_factory.UnifiedResponse
```

Indexing

- First index: client (still necessary even if results are only found for a single client)
- Second index: rows (must be **iterable**, different from [Example: downloading data using Fido])

```
response[1]
```

Results from 1 Provider:3 Results from the JSOCClient:*Table length=3*

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145
2014-01-01T00:00:13Z	SDO/AIA	AIA_3	171	2145
2014-01-01T00:00:25Z	SDO/AIA	AIA_3	171	2145

```
response[1, ::2]
```

Results from 1 Provider:2 Results from the JSOCClient:*Table length=2*

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145
2014-01-01T00:00:25Z	SDO/AIA	AIA_3	171	2145

```
response[0, [2]]  # Not `response[0, 2]` !
```

Results from 1 Provider:1 Results from the JSOCClient:*Table length=1*

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:25Z	SDO/AIA	AIA_4	304	2145

Downloading data

Download the entire results:

```
res = Fido.fetch(response, path='./data')  # Will output a progress bar by default
```

JSOCClient

> The JSOC stages data before you can download it, so a JSOC query is a **three** stage process.
>
> 1. **`search`**: query the JSOC for records and a table of these records is returned.
>   The result is `UnifiedResponse` for `Fido` while `JSOCResponse` for `JSOCClient`
> 2. **`request_data`**: request these records to be staged for download.
> 3. **`get_request`**: download.
>
> (`Fido` combines the stages into 2, `search` and `fetch`.)

Searching for data

```
from sunpy.net import jsoc
client = jsoc.JSOCClient()
```

```
response = client.search(
    attrs.jsoc.Time('2014-01-01T00:00:00', '2014-01-01T00:00:20'),
    attrs.jsoc.Notify('lydiazly@nju.edu.cn'),
    attrs.jsoc.Series('aia.lev1_euv_12s'),
    attrs.jsoc.Wavelength(304 * u.AA) | attrs.jsoc.Wavelength(171 * u.AA)
)
response
```

Table length=4

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_4	304	2145
2014-01-01T00:00:13Z	SDO/AIA	AIA_4	304	2145
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145
2014-01-01T00:00:13Z	SDO/AIA	AIA_3	171	2145

Note here we get a single table.

```
type(response)
```

```
sunpy.net.jsoc.jsoc.JSOCResponse
```

Indexing

An integer get a `Row`:

```
response[2]
```

Row index=2

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145

A **iterable** get a `Table`:

```
response[[2]]
```

Table length=1

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145

Convert a `Row` to a `Table`:

```
from astropy.table.table import Table
Table(response[2].table)
```

Table length=1

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145

Other examples:

```
response[::2]
```

Table length=2

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_4	304	2145
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145

```
response[:2]
```

Table length=2

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_4	304	2145
2014-01-01T00:00:13Z	SDO/AIA	AIA_4	304	2145

```
response[[0, 2]]
```

Table length=2

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str20	**str7**	**str5**	**int64**	**int64**
2014-01-01T00:00:01Z	SDO/AIA	AIA_4	304	2145
2014-01-01T00:00:01Z	SDO/AIA	AIA_3	171	2145

Downloading data

Download the entire results:

```
res = client.fetch(response, path='./data')
```

```
Export request pending. [id="JSOC_20180710_1151", status=2]
Waiting for 0 seconds...
Export request pending. [id="JSOC_20180710_1151", status=1]
Waiting for 5 seconds...
4 URLs found for download. Full request totalling 17MB
Export request pending. [id="JSOC_20180710_1152", status=2]
Waiting for 0 seconds...
4 URLs found for download. Full request totalling 24MB
```

```
res.wait()  # Show the progress bar & wait to finish downloading.
```

Or separate out the staging to `request_data` & `get_request`:

```
requests = client.request_data(response)
```

```
requests.id
```

```
'JSOC_20180708_435'
```

```
requests.status
```

```
0
```

```
res = client.get_request(requests, path='./data')
```

```
res.wait()
```

So far, the best way to download a subset of the results is to **make a query again**.

Or use **`JSOCClient`** (and a custom function) instead of **`Fido`** to search for data.

See details below:

If slice the result as `response[2]`, everything is OK except `query_args` is lost.

```
response.query_args
```

```
[{'wavelength': <Quantity 304. Angstrom>,
  'start_time': datetime.datetime(2014, 1, 1, 0, 0, 35),
  'end_time': datetime.datetime(2014, 1, 1, 0, 0, 55),
  'notify': 'lydiazly@nju.edu.cn',
  'series': 'aia.lev1_euv_12s'},
 {'wavelength': <Quantity 171. Angstrom>,
  'start_time': datetime.datetime(2014, 1, 1, 0, 0, 35),
  'end_time': datetime.datetime(2014, 1, 1, 0, 0, 55),
  'notify': 'lydiazly@nju.edu.cn',
  'series': 'aia.lev1_euv_12s'}]
```

```
response[2].query_args
```

This will cause an error in `fetch`.

Constructing complex queries

Time

http://docs.sunpy.org/en/stable/guide/acquiring_data/jsoc.html

> Start and end times for the query (any date/time format understood by SunPy's `parse_time` function can be used to specify dates and time). **The Time attribute takes UTC time, as default.**
> If you need to pass a Time in some other time scale, such as TAI, pass an Astropy Time object

See http://docs.sunpy.org/en/stable/guide/time.html#parse-time

```
from sunpy.time import parse_time, is_time, is_time_in_given_format
```

```
is_time('2014-01-01T00:00:00')
```

```
True
```

```
is_time_in_given_format('2014-01-01T00:00:00Z', "%Y-%m-%dT%H:%M:%SZ")
```

```
True
```

```
parse_time('2015.08.27_06:00:00_TAI')
```

```
datetime.datetime(2015, 8, 27, 6, 0)
```

```
parse_time('2014-01-01T00:00:00Z')
```

```
datetime.datetime(2014, 1, 1, 0, 0)
```

If `sunpy.time` doesn't inlude this format: "%Y-%m-%dT%H:%M:%SZ", use a bash script to add it:

```
!sunpy-add-time-fmt
```

```
Added this format to
/home/lydia/miniconda3/lib/python3.6/site-packages/sunpy/time/time.py:

    "%Y-%m-%dT%H:%M:%SZ",       # Example 2007-05-04T21:08:12Z

=== Done!(sunpy-add-time-fmt) ===
```

```
parse_time('2014-01-01T00:00:00Z')
```

```
datetime.datetime(2014, 1, 1, 0, 0)
```

For convenience, use a custom function to pass a time string in 'TAI' scale:

```
from usr_sunpy import tai
```

```
tai('2014-01-01T00:00:00')  # Returns a single object
```

```
<Time object: scale='tai' format='datetime' value=2014-01-01 00:00:00>
```

```
tai('2015.08.27_06:00:00_TAI')
```

```
<Time object: scale='tai' format='datetime' value=2015-08-27 06:00:00>
```

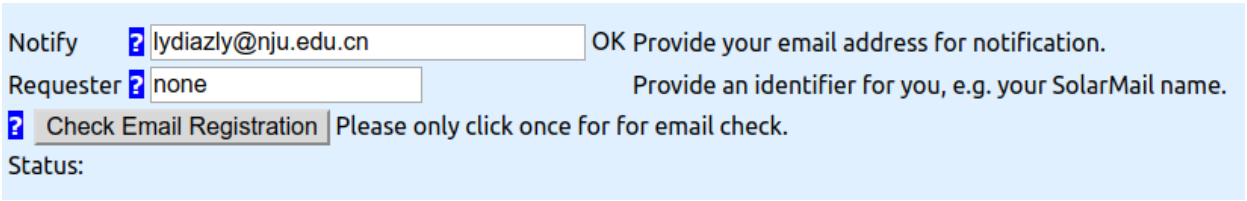
```
tai('2015.08.27_06:00:00_TAI', '2015.08.27_06:01:00_TAI')  # Returns a list
```

```
[<Time object: scale='tai' format='datetime' value=2015-08-27 06:00:00>,
 <Time object: scale='tai' format='datetime' value=2015-08-27 06:01:00>]
```

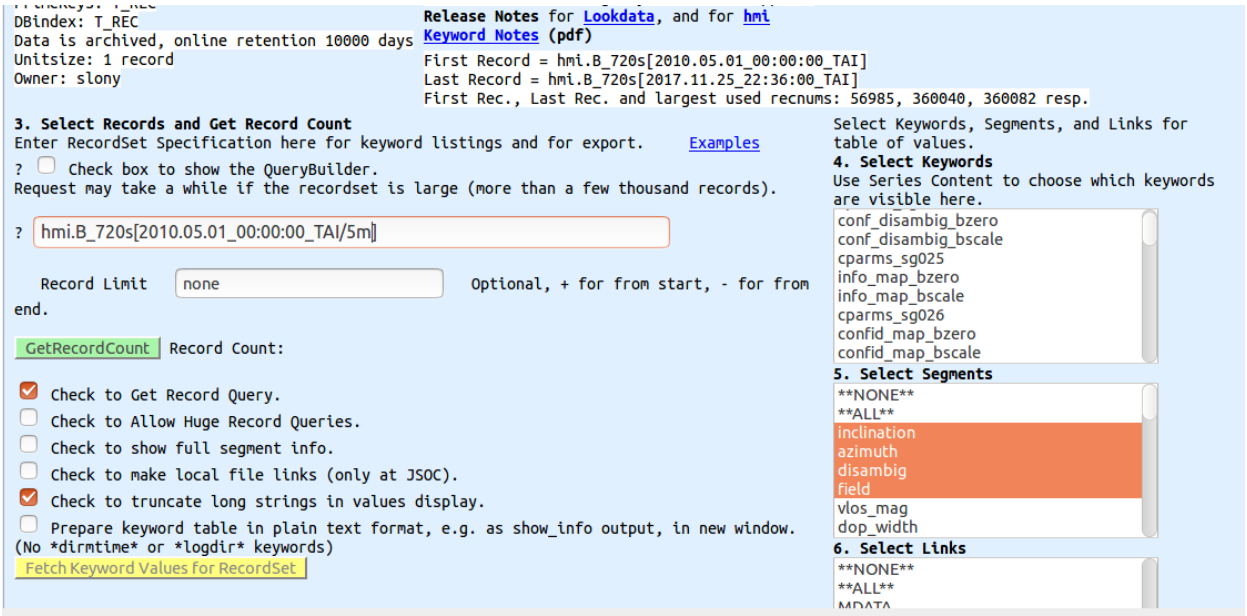
Searching for data

JSOC http://jsoc.stanford.edu/ajax/lookdata.html

- To make a request, register your email first :



Example of online JSOC interface :



RequestID JSOC_20171203_991 This is the ID tag for your export request. Use the Status Request button below to retrieve the links to the data.
Status Data Ready, size=61MB
Data Location

JSOC Data Export Status and Retrieval

RequestID JSOC_20171203_991 This is the ID tag for your export request. Success
Submit Status Request Please only click once for status request.
Clear Request Clear old status RequestID
List formats are index.html, index.json, and index.txt
export script file is JSOC_20171203_991.drmsrun
Status Data Ready, size=61MB
Data Location http://jsoc.stanford.edu/SUM86/D996127985/S00000/

File	Record	Filename
1	hmi.B_720s[2015.08.27_05:24:00_TAI]	hmi.B_720s.20150827_052400_TAI.inclination.fits
2	hmi.B_720s[2015.08.27_05:24:00_TAI]	hmi.B_720s.20150827_052400_TAI.azimuth.fits
3	hmi.B_720s[2015.08.27_05:24:00_TAI]	hmi.B_720s.20150827_052400_TAI.disambig.fits
4	hmi.B_720s[2015.08.27_05:24:00_TAI]	hmi.B_720s.20150827_052400_TAI.field.fits

Use

```
attrs.jsoc.Segment('...') &  attrs.jsoc.<attr>('...') ...
```

```
attrs.jsoc.Segment('...') |  attrs.jsoc.<attr>('...') ...
```

or

```
from sunpy.net.attr import AttrAnd, AttrOr
```

```
AttrAnd(list(map(attrs.jsoc.<attr>, <list>)))
```

```
AttrOr(list(map(attrs.jsoc.<attr>, <list>)))
```

to pass multiple attributes.

Note:

> The attributes which support `&` are `PrimeKey` and `Segment`. Using `&` with any other attributes will throw an error.

- Specify attributes:

e.g.

```
trange = tai('2015-08-27T05:00:00', '2015-08-27T06:00:00')

segments = ['inclination', 'azimuth', 'disambig', 'field']

series = 'hmi.B_720s'  # 'hmi.b_720s' is OK

interval = 10 * u.min  # every 10 min.

email = 'lydiazly@nju.edu.cn'
```

- Making a query:

```
from sunpy.net.attr import AttrAnd, AttrOr
```

```
response = Fido.search(
    attrs.jsoc.Time(*trange),
    attrs.jsoc.Series(series),
    attrs.jsoc.Notify(email),
    attrs.Sample(interval),
    AttrAnd(list(map(attrs.jsoc.Segment, segments)))
    # i.e. attrs.jsoc.Segment('...') &  attrs.jsoc.Segment('...') ...
)
response
```

Results from 1 Provider:6 Results from the JSOCClient:*Table length=6*

T_REC	TELESCOP	INSTRUME	WAVELNTH	CAR_ROT
str23	**str7**	**str9**	**float64**	**int64**
2015.08.27_05:00:00_TAI	SDO/HMI	HMI_SIDE1	6173.0	2167
2015.08.27_05:12:00_TAI	SDO/HMI	HMI_SIDE1	6173.0	2167
2015.08.27_05:24:00_TAI	SDO/HMI	HMI_SIDE1	6173.0	2167
2015.08.27_05:36:00_TAI	SDO/HMI	HMI_SIDE1	6173.0	2167
2015.08.27_05:48:00_TAI	SDO/HMI	HMI_SIDE1	6173.0	2167
2015.08.27_06:00:00_TAI	SDO/HMI	HMI_SIDE1	6173.0	2167