Release 0.5.7

Date August 16, 2019

Github https://github.com/sunpy/drms

PyPI https://pypi.python.org/pypi/drms

Python module for accessing HMI, AIA and MDI data.
The `drms` Python package can be used to access HMI, AIA and MDI data which are stored in a DRMS database system.

DRMS stands for *Data Record Management System* and is a system that was developed by the Joint Science Operation Center (JSOC), headquartered at Stanford University, to handle the data produced by the AIA and HMI instruments aboard the Solar Dynamics Observatory spacecraft.

By default the `drms` module uses the HTTP/JSON interface provided by JSOC and has similar functionality to the JSOC Lookdata website. It can be used to query metadata, submit data export requests and download data files.

This module also works well for local NetDRMS sites, as long as the site runs a web server providing the needed CGI programs `show_series` and `jsoc_info` (for the data export functionality, additional CGIs, like `jsoc_fetch`, are needed).

### 1.1 Requirements

The `drms` module supports Python 2.7 and Python 3.4 or newer. It requires the following Python packages:

- NumPy, version 1.9.0 or newer
- Pandas, version 0.15.0 or newer
- Six, version 1.8.0 or newer

The module might also work with earlier versions, but it has not been tested with any versions older than the ones listed above.

### 1.2 Installation

If you are using Anaconda, it is recommended to use the `conda-forge` package:
conda config --append channels conda-forge
conda install drms

Otherwise the drms Python package can be installed from PyPI using

```bash
pip install drms
```

**Note:** If you do not use a Python distribution, like Anaconda, and did not create an isolated Python environment using Virtualenv, you might need to add `--user` to the `pip` command:

```bash
pip install --user drms
```

## 1.3 Acknowledgements

The main author of this project has received funding from the European Research Council under the European Union’s Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 307117.

See LICENSE.txt for the license text and AUTHORS.txt for a list of contributors.
This tutorial gives an introduction on how to use the `drms` Python module. More detailed information on the different classes and functions can be found in the API Reference Manual. In addition to this tutorial, many example scripts are available in the source code package of the `drms` module.

Tip: Instead of using a plain Python interpreter session, it is highly recommended to use an interactive IPython shell or a Jupyter notebook for this tutorial.

## 2.1 Basic usage

In this first part, we start with looking at data series that are available from JSOC and perform some basic DRMS queries to obtain keyword data (metadata) and segment file (data) locations. This is essentially what you can do on the JSOC Lookdata website.

To be able to access the JSOC DRMS from Python, we first need to import the `drms` module and create an instance of the `drms.Client` class:

```python
>>> import drms

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

All available data series can be now retrieved by calling the `Client.series()` method. HMI series names start with "hmi.", AIA series names with "aia." and the names of MDI series with "mdi.".

The first (optional) parameter of this method takes a regular expression that allows you to filter the result. If you, for example, want to obtain a list of HMI series, with a name that start with the string "m_", you can write:

```python
>>> c.series(r'hmi\._m\_')
['hmi.M_45s', 'hmi.M_720s', 'hmi.m_720s_mod', 'hmi.m_720s_nrt']
```

Keep in mind to escape the dot character (`. `), like it is shown in the example above, if you want to include it in your filter string. Also note that series names are handled in a case-insensitive way.
DRMS records can be selected by creating a query string that contains a series name, followed by one or more fields, which are surrounded by square brackets. Each of those fields corresponds to a specific primekey, that is specified in the series definition. A complete set of primekeys represents a unique identifier for a record in that particular series. For more detailed information on building record set queries, including additional non-primekey fields, see the JSOC Help page about this topic.

With the drms module you can use the Client.pkeys() method to obtain a list of all primekeys of a series, e.g.:

```python
>>> c.pkeys('hmi.m_720s')
['T_REC', 'CAMERA']
```

```python
>>> c.pkeys('hmi.v_sht_modes')
['T_START', 'LMIN', 'LMAX', 'NDT']
```

A list of all (regular) keywords can be obtained using Client.keys(). You can also use the method Client.info() to get more detailed information about a series, e.g.:

```python
>>> si = c.info('hmi.v_avg120')
```

```python
>>> si.segments

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>units</th>
<th>protocol</th>
<th>dims</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>short</td>
<td>m/s</td>
<td>fits</td>
<td>4096x4096</td>
<td>Doppler mean</td>
</tr>
<tr>
<td>power</td>
<td>short</td>
<td>m2/s2</td>
<td>fits</td>
<td>4096x4096</td>
<td>Doppler power</td>
</tr>
<tr>
<td>valid</td>
<td>short</td>
<td>NA</td>
<td>fits</td>
<td>4096x4096</td>
<td>valid pixel count</td>
</tr>
<tr>
<td>Log</td>
<td>char</td>
<td>NA</td>
<td>generic</td>
<td>run log</td>
<td></td>
</tr>
</tbody>
</table>
```

All table-like structures, returned by routines in the drms module, are Pandas DataFrames. If you are new to Pandas, you should have a look at the introduction to Pandas Data Structures.

Record set queries, used to obtain keyword data and get the location of data segments, can be performed using the Client.query() method. To get, for example, the record time and the mean value for some of the HMI Doppler-grams that were recorded on April 1, 2016, together with the spacecraft’s radial velocity in respect to the Sun, you can write:

```python
>>> k = c.query('hmi.v_45s[2016.04.01_TAI/1d@6h]', key='T_REC, DATAMEAN, OBS_VR')
```

```python
T_REC    DATAMEAN   OBS_VR
0 2016-04-01 00:00:00 3313.104980 3309.268006
1 2016-04-01 06:00:00 878.075195 887.864139
2 2016-04-01 12:00:00 -2289.062500 -2284.690263
3 2016-04-01 18:00:00 128.609283 137.836168
```

JSOC time strings can be converted to a naive datetime representation using the drms.to_datetime() utility function:

```python
>>> t = drms.to_datetime(k.T_REC)
```

```python
0 2016-04-01 00:00:00
1 2016-04-01 06:00:00
2 2016-04-01 12:00:00
3 2016-04-01 18:00:00
Name: T_REC, dtype: datetime64[ns]
```

For most of the HMI and MDI data sets, the TAI time standard is used which, in contrast to UTC, does not make use of any leap seconds. The TAI standard is currently not supported by the Python standard libraries. If you need to convert timestamps between TAI and UTC, you can use the Astropy time module:
The "hmi.v_45s" series has a data segment with the name "Dopplergram", which contains Dopplergrams for each record in the series, that are stored as FITS files. The location of the FITS files for the record set query in the example above, can be obtained by using the seg parameter of the `Client.query()` method:

```
>>> s = c.query('hmi.v_45s[2016.04.01_TAI/1d@6h]', seg='Dopplergram')
```

Note that the key and seg parameters can also be used together in one `Client.query()` call, i.e.:

```
>>> k, s = c.query('hmi.v_45s[2016.04.01_TAI/1d@6h]',
...    key='T_REC, DATAMEAN, OBS_VR', seg='Dopplergram')
```

The file paths listed above are the storage location on the JSOC server. You can access these files, even if you do not have direct NFS access to the filesystem, by prepending the JSOC URL to segment file path:

```
>>> url = 'http://jsoc.stanford.edu' + s.Dopplergram[0]
```

Note that FITS files which are accessed in this way, do not contain any keyword data in their headers. This is perfectly fine in many cases, because you can just use `Client.query()` to obtain the data of all required keywords. If you need FITS files with headers that contain all the keyword data, you need to submit an export request to JSOC, which is described in the next section.

Export requests can also be useful, if you want to download more than only one or two files (even without keyword headers), because you can then use the `ExportRequest.download()` method, which takes care of creating URLs, downloading the data and (if necessary) generating suitable local filenames.

### 2.2 Data export requests

Data export requests can be interactively built and submitted on the JSOC Export Data webpage, where you can also find more information about the different export options that are available. Note that a registered email address is required for submitting export requests. You can register your email address on the JSOC email registration webpage.

It is advisable to have a closer look at the export webpage before submitting export requests using the `drms` module. It is also possible to submit an export request on the webpage and then use the Python routines to query the request status and download files.
First, we start again with importing the `drms` module and creating a `drms.Client` instance:

```python
>>> import drms

>>> c = drms.Client(email='name@example.com', verbose=True)
```

In this case we also provide an email address (which needs to be already registered at JSOC) and turn on status messages by enabling the `verbose` flag.

We now create a download directory for our downloads, in case it does not exist yet:

```python
>>> import os

>>> out_dir = 'downloads'

>>> if not os.path.exists(out_dir):
...     os.mkdir(out_dir)
```

Data export requests can be submitted using `Client.export()`. The most important parameters of this method, besides the export query string, are the parameters `method` and `protocol`. There are many different export methods and protocols available. In the following examples we confine ourselves to the methods `url_quick` and `url` and the protocols `as-is` and `fits`. You can find more examples (including other export methods and protocols) in the source code package of the `drms` module.

### 2.2.1 url_quick / as-is

The most direct and quickest way of downloading files is the combination `url_quick` / `as-is`. This (in most cases) does not create an actual export request, where you would have to wait for it being finished, but rather compiles a list of files from your data export query, which can then be directly downloaded. This also means that this kind of export usually has no `ExportID` assigned to it. The only time it is treated as a “real” export request (including an `ExportID` and some wait time) is, when the requested data segments are not entirely online, and parts of the requested files need to be restored from tape drives.

As an example, we now create an `url_quick` / `as-is` export request for the same record set that was used in the previous section. For export requests, the segment name is specified using an additional field in the query string, surrounded by curly braces. Note that `Client.export()` performs an `url_quick` / `as-is` export request by default, so you do not need to explicitly use `method='url_quick'` and `protocol='as-is'` in this case.

```python
>>> r = c.export('hmi.v_45s[2016.04.01_TAI/1d@6h]{Dopplergram}')

>>> r
<ExportRequest id=None, status=0>

>>> r.data.filename
0 /SUM58/D803708321/S00008/Dopplergram.fits
1 /SUM41/D803708361/S00008/Dopplergram.fits
2 /SUM71/D803720859/S00008/Dopplergram.fits
3 /SUM70/D803730119/S00008/Dopplergram.fits
```

Download URLs can now be generated using the `ExportRequest.urls` attribute:

```python
>>> r.urls.url[0]
'http://jsoc.stanford.edu/SUM58/D803708321/S00008/Dopplergram.fits'
```

Files can be downloaded using the `ExportRequest.download()` method. You can (optionally) select which file(s) you want to download, by using the `index` parameter of this method. The following, for example, only downloads the first file of the request:

```python
>>> r.download(out_dir, 0)
Downloading file 1 of 1...
```
Being a direct as-is export, there are no keyword data written to any FITS headers. If you need keyword data added to the headers, you have to use the fits export protocol instead, which is described below.

### 2.2.2 url / fits

Using the fits export protocol, allows you to request FITS files that include all keyword data in their headers. Note that this protocol does not convert other file formats into the FITS format. The only purpose of protocol='fits' is to add keyword data to headers of segment files, that are already stored using the FITS format.

In contrast to url_quick/as-is exports, described in the previous subsection, url/fits exports always create a “real” data export request on the server, which needs to be processed before you can download the requested files. For each request you will get an unique ExportID, which can be accessed using the ExportRequest.id attribute. In addition you will get an email notification (including the ExportID), which is sent to your registered email address when the requested files are ready for download.

In the following example, we use the hmi.sharp_720s series, which contains Spaceweather HMI Active Region Patches (SHARPs), and download some data files from this series.

First we have a look at the content of the series, by using Client.info() to get a SeriesInfo instance for this particular series:

```python
>>> si = c.info('hmi.sharp_720s')
```

```python
>>> si.note
'Spaceweather HMI Active Region Patch (SHARP): CCD coordinates'
```

```python
>>> si.primekeys
['HARPNUM', 'T_REC']
```

This series contains a total of 31 different data segments:

```python
>>> len(si.segments)
31
```

```python
>>> si.segments.index.values
array(['magnetogram', 'bitmap', 'Dopplergram', 'continuum', 'inclination',
     'azimuth', 'field', 'vlos_mag', 'dop_width', 'eta_0', 'damping',
     'src_continuum', 'src_grad', 'alpha_mag', 'chisq', 'conv_flag',
     'info_map', 'confid_map', 'inclination_err', 'azimuth_err',
     'field_err', 'vlos_err', 'alpha_err', 'field_inclination_err',
     'field_az_err', 'inclin_azimuth_err', 'field_alpha_err',
     'inclination_alpha_err', 'azimuth_alpha_err', 'disambig',
     'conf_disambig'], dtype=object)
```

Here, we are only interested in magnetograms and continuum intensity maps:

```python
>>> si.segments.loc[['continuum', 'magnetogram']]
```

<table>
<thead>
<tr>
<th>type</th>
<th>units</th>
<th>protocol</th>
<th>dims</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>continium</td>
<td>int</td>
<td>DN/s</td>
<td>fits</td>
<td>VARxVAR continuum intensity</td>
</tr>
<tr>
<td>magnetogram</td>
<td>int</td>
<td>Gauss</td>
<td>fits</td>
<td>VARxVAR magnetogram</td>
</tr>
</tbody>
</table>
which are stored as FITS files with varying dimensions.

If we now want to submit an export request for a magnetogram and an intensity map of HARP number 4864, recorded at midnight on November 30, 2014, we can use the following export query string:

```python
>>> ds = 'hmi.sharp_720s[4864][2014.11.30_00:00:00_TAI]{continuum,magnetogram}'
```

In order to obtain FITS files that include keyword data in their headers, we then need to use `protocol='fits'` when submitting the request using `Client.export()`:

```python
>>> r = c.export(ds, method='url', protocol='fits')
>>> r
<ExportRequest id="JSOC_20160921_568", status=2>
```

We now need to wait for the server to prepare the requested files:

```python
>>> r.wait()
Export request pending. [id="JSOC_20160921_568", status=2]
Waiting for 5 seconds...
Export request pending. [id="JSOC_20160921_568", status=1]
Waiting for 5 seconds...
```

Note that calling `ExportRequest.wait()` is optional. It gives you some control over the waiting process, but it can be usually omitted, in which case `ExportRequest.wait()` is called implicitly, when you for example try to download the requested files.

After the export request is finished, a unique request URL is created for you, which points to the location where all your requested files are stored. You can use the `ExportRequest.request_url` attribute to obtain this URL:

```python
>>> r.request_url
'http://jsoc.stanford.edu/SUM80/D857351442/S00000'
```

Note that this location is only temporary and that all files will be deleted after a couple of days.

Downloading the data works exactly like in the previous example, by using the `ExportRequest.download()` method:

```python
>>> r.download(out_dir)
Downloading file 1 of 2...
  record: hmi.sharp_720s[4864][2014.11.30_00:00:00_TAI]
  filename: hmi.sharp_720s.4864.20141130_000000_TAI.magnetogram.fits
  -> "downloads/hmi.sharp_720s.4864.20141130_000000_TAI.magnetogram.fits"
Downloading file 2 of 2...
  record: hmi.sharp_720s[4864][2014.11.30_00:00:00_TAI]
  filename: hmi.sharp_720s.4864.20141130_000000_TAI.continuum.fits
  -> "downloads/hmi.sharp_720s.4864.20141130_000000_TAI.continuum.fits"
```

**Tip:** If you want to access an existing export request that you have submitted earlier, or if you submitted an export request using the JSOC Export Data webpage and want to access it from Python, you can use the `Client.export_from_id()` method with the corresponding ExportID to create an `ExportRequest` instance for this particular request.
2.3 Example scripts

There are many example scripts available in the examples directory of the drms Python package source code. An archive of the latest source code release can be downloaded from the drms release page on Github.
This reference manual contains detailed information about classes and functions included in the drms module. For an introduction on how to use the drms module, see also the tutorial.

### 3.1 Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Client for remote DRMS server access.</td>
</tr>
<tr>
<td>SeriesInfo</td>
<td>DRMS series details.</td>
</tr>
<tr>
<td>ExportRequest</td>
<td>Class for handling data export requests.</td>
</tr>
</tbody>
</table>

#### 3.1.1 drms.Client

**class drms.Client (server='jsoc', email=None, verbose=False, debug=False)**

Client for remote DRMS server access.

**Parameters**

- **server** [string or ServerConfig] Registered server ID or ServerConfig instance. Defaults to JSOC.
- **email** [string or None] Default email address used for data export requests.
- **verbose** [bool] Print export status messages to stdout (disabled by default).
- **debug** [bool] Print debug output (disabled by default).

**Attributes**

- **email** [string] (string) Default email address used for data export requests.
- **verbose** [bool] (bool) Enable/disable export status output.
**debug** [bool] (bool) Enable/disable debug output.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>check_email</code></td>
<td>Check if the email address is registered for data export.</td>
</tr>
<tr>
<td><code>export</code></td>
<td>Submit a data export request.</td>
</tr>
<tr>
<td><code>export_from_id</code></td>
<td>Create an <code>ExportRequest</code> instance from an existing requestid.</td>
</tr>
<tr>
<td><code>get</code></td>
<td>This method is deprecated.</td>
</tr>
<tr>
<td><code>info</code></td>
<td>Get information about the content of a data series.</td>
</tr>
<tr>
<td><code>keys</code></td>
<td>Get a list of keywords that are available for a series.</td>
</tr>
<tr>
<td><code>pkeys</code></td>
<td>Get a list of primekeys that are available for a series.</td>
</tr>
<tr>
<td><code>query</code></td>
<td>Query keywords, segments and/or links of a record set.</td>
</tr>
<tr>
<td><code>series</code></td>
<td>List available data series.</td>
</tr>
</tbody>
</table>

**drms.Client.check_email**

method

**Client.check_email**(self, email)

Check if the email address is registered for data export. You can register your email for data exports from JSOC on the JSOC email registration webpage.

**Parameters**

- **email** [string] Email address to be checked.

**Returns**

- **result** [bool] True if the email address is valid and registered, False otherwise.

**drms.Client.export**

method

**Client.export**(self, ds[, method, protocol, ...])

Submit a data export request.

A registered email address is required for data exports. You can register your email address for data exports from JSOC on the JSOC email registration webpage.

An interactive webinterface and additional information is available on the JSOC data export webpage.

Note that export requests that were submitted using the webinterface can be accessed using the `export_from_id()` method.

**Parameters**

- **ds** [string] Data export record set query.
- **method** [string] Export method. Supported methods are: `url_quick`, `url`, `url-tar`, `ftp` and `ftp-tar`. Default is `url_quick`.
- **protocol** [string] Export protocol. Supported protocols are: `as-is`, `fits`, `jpg`, `mpg` and `mp4`. Default is `as-is`.  


**filenamefmt** [string, None or False] Custom filename format string for exported files. This is ignored for ‘url_quick’/’as-is’ data exports. If set to None (default), the format string will be generated using the primekeys of the data series. If set to False, the filename format string will be omitted in the export request.

**n** [int or None] Limits the number of records requested. For positive values, the first n records of the record set are returned, for negative values the last abs(n) records. If set to None (default), no limit is applied.

**email** [string or None] Registered email address. If email is None (default), the current default email address is used, which in this case has to be set before calling export() by using the `Client.email` attribute.

**requestor** [string, None or False] Export user ID. Default is None, in which case the user name is determined from the email address. If set to False, the requestor argument will be omitted in the export request.

**Returns**

result [ExportRequest]

---

**drms.Client.export_from_id**

method

`Client.export_from_id(self, requestid)`

Create an ExportRequest instance from an existing requestid.

**Parameters**

requestid [string] Export request ID.

**Returns**

result [ExportRequest]

---

**drms.Client.get**

method

`Client.get(self, ds, key=None, seg=None, link=None, convert_numeric=True, skip_conversion=None)`

This method is deprecated. Use `query()` instead.

---

**drms.Client.info**

method

`Client.info(self, ds)`

Get information about the content of a data series.

**Parameters**

ds [string] Name of the data series.

**Returns**

---

3.1. Classes
result  [SeriesInfo] SeriesInfo instance containing information about the data series.

drms.Client.keys
method
Client.keys (self, ds)
Get a list of keywords that are available for a series. Use the info() method for more details.

Parameters

- **ds**  [string] Name of the data series.

Returns

- **result**  [list] List of keywords available for the selected series.

drms.Client.pkeys
method
Client.pkeys (self, ds)
Get a list of primekeys that are available for a series. Use the info() method for more details.

Parameters

- **ds**  [string] Name of the data series.

Returns

- **result**  [list] List of primekeys available for the selected series.

drms.Client.query
method
Client.query (self, ds, key=None, seg=None, link=None, convert_numeric=True, skip_conversion=None, pkeys=False, rec_index=False, n=None)
Query keywords, segments and/or links of a record set. At least one of the parameters key, seg, link or pkeys needs to be specified.

Parameters

- **ds**  [string] Record set query.

  - **key**  [string, list of strings or None] List of requested keywords, optional. If set to None (default), no keyword results will be returned, except when pkeys is True.

  - **seg**  [string, list of strings or None] List of requested segments, optional. If set to None (default), no segment results will be returned.

  - **link**  [string, list of strings or None] List of requested Links, optional. If set to None (default), no link results will be returned.

  - **convert_numeric**  [bool] Convert keywords with numeric types from string to numbers. This may result in NaNs for invalid/missing values. Default is True.

  - **skip_conversion**  [list of strings or None] List of keywords names to be skipped when performing a numeric conversion. Default is None.

  - **pkeys**  [bool] If True, all primekeys of the series are added to the key parameter.
rec_index  [bool] If True, record names are used as index for the resulting DataFrames.

n  [int or None] Limits the number of records returned by the query. For positive values, the first n records of the record set are returned, for negative values the last abs(n) records. If set to None (default), no limit is applied.

Returns

res_key  [pandas.DataFrame, optional] Keyword query results. This DataFrame is only returned, if key is not None or pkeys is set to True.

res_seg  [pandas.DataFrame, optional] Segment query results. This DataFrame is only returned, if seg is not None.

res_link  [pandas.DataFrame, optional] Link query results. This DataFrame is only returned, if link is not None.

drms.Client.series

method

Client.series(self, regex=None, full=False)
List available data series.

Parameters

regex  [string or None] Regular expression, used to select a subset of the available series. If set to None, a list of all available series is returned.

full  [bool] If True, return a pandas.DataFrame containing additional series information, like description and primekeys. If False (default), the result is a list containing only the series names.

Returns

result  [list or pandas.DataFrame] List of series names or DataFrame containing name, primekeys and a description of the selected series (see parameter full).

3.1.2 drms.SeriesInfo

class drms.SeriesInfo(d, name=None)
DRMS series details.

Use Client.info() to create an instance.

Attributes

name  [string] Series name.

primekeys  [list of strings] Series primekeys.


segments  [pandas.DataFrame] Details about series segments.

note  [string] Series description.

dbindex  [list of strings] Series database index.

retention  [int] Default retention time.
unitsize  [int] Storage unit size.
archive  [int] Series archive flag.
tapegroup [int] Tape group.

3.1.3 drms.ExportRequest

class drms.ExportRequest (d, client)
Class for handling data export requests.

Use Client.export() or Client.export_from_id() to create an instance.

Attributes

id    [string] (string) Request ID.
status [int] (int) Export request status.
urls  [pandas.DataFrame] (pandas.DataFrame) URLs of all downloadable files.
request_url [string] (string) URL of the export request.
method [string] (string) Export method.
protocol [string] (string) Export protocol.
data  [pandas.DataFrame] (pandas.DataFrame) Records and filenames of the export request.
dir   [string] (string) Common directory of the requested files on the server.
tarfile [string] (string) Filename, if a TAR file was requested.
keywords [string] (string) Filename of textfile containing record keywords.

Methods

download(self, directory[, index, ...]) Download data files.
has_failed(self[, skip_update]) Check if the export request has finished unsuccess-
fully.
has_finished(self[, skip_update]) Check if the export request has finished.
has_succeeded(self[, skip_update]) Check if the export request has finished successfully.
wait(self[, timeout, sleep, ...]) Wait for the server to process the export request.

drms.ExportRequest.download

method

ExportRequest.download (self, directory, index=None, fname_from_rec=None, verbose=None)
Download data files.

By default, the server-side filenames are used as local filenames, except for export method ‘url_quick’,
where the local filenames are generated from record names (see parameter fname_from_rec). In case a
file with the same name already exists in the download directory, an ascending number is appended to the
filename.

Note: Downloading data segments that are directories, e.g. data segments from series like
“hmi.rdVflows_fd15_frame”, is currently not supported. In order to download data from series like this,
you need to use the export methods ‘url-tar’ or ‘ftp-tar’ when submitting the data export request.
Parameters

directory [string] Download directory (must already exist).

index [int, list of ints or None] Index (or indices) of the file(s) to be downloaded. If set to None (the default), all files of the export request are downloaded. Note that this parameter is ignored for export methods 'url-tar' and 'ftp-tar', where only a single tar file is available for download.

fname_from_rec [bool or None] If True, local filenames are generated from record names. If set to False, the original filenames are used. If set to None (default), local filenames are generated only for export method 'url_quick'. Exceptions: For exports with methods 'url-tar' and 'ftp-tar', no filename will be generated. This also applies to movie files from exports with protocols 'mpg' or 'mp4', where the original filename is used locally.

verbose [bool or None] Set to True if status messages should be printed to stdout. If set to None (default), the Client.verbose flag of the associated client instance is used instead.

Returns

result [pandas.DataFrame] DataFrame containing the record string, download URL and local location of each downloaded file (DataFrame columns: ‘record’, ‘url’ and ‘download’).

drms.ExportRequest.has_failed

method

ExportRequest\.has_failed(\self, skip_update=False) Check if the export request has finished unsuccessfully.

Parameters

skip_update [bool] If set to True, the export status will not be updated from the server, even if it was in pending state after the last status update.

Returns

result [bool] True if the export request has finished unsuccessfully or False if the request has succeeded or is still pending.

drms.ExportRequest.has_finished

method

ExportRequest\.has_finished(\self, skip_update=False) Check if the export request has finished.

Parameters

skip_update [bool] If set to True, the export status will not be updated from the server, even if it was in pending state after the last status update.

Returns

result [bool] True if the export request has finished or False if the request is still pending.
**drms.ExportRequest.has_succeeded**

*method*

```
ExportRequest.has_succeeded(self, skip_update=False)
```

Check if the export request has finished successfully.

**Parameters**

- **skip_update** [bool] If set to True, the export status will not be updated from the server, even if it was in pending state after the last status update.

**Returns**

- **result** [bool] True if the export request has finished successfully or False if the request failed or is still pending.

**drms.ExportRequest.wait**

*method*

```
ExportRequest.wait(self, timeout=None, sleep=5, retries_notfound=5, verbose=None)
```

Wait for the server to process the export request. This method continuously updates the request status until the server signals that the export request has succeeded or failed.

**Parameters**

- **timeout** [number or None] Maximum number of seconds until this method times out. If set to None (the default), the status will be updated indefinitely until the request succeeded or failed.
- **sleep** [number or None] Time in seconds between status updates (defaults to 5 seconds). If set to None, a server-supplied value is used.
- **retries_notfound** [int] Number of retries in case the request was not found on the server. Note that it usually takes a short time until a new request is registered on the server, so a value too low might cause an exception to be raised, even if the request is valid and will eventually show up on the server.
- **verbose** [bool or None] Set to True if status messages should be printed to stdout. If set to None (default), the `Client.verbose` flag of the associated client instance is used instead.

**Returns**

- **result** [bool] True if the request succeeded or False if a timeout occurred. In case of an error an exception is raised.

### 3.2 Constants and utility functions

**const**

Constants for DRMS queries.

**to_datetime** *(tstr[, force])*  
Parse JSOC time strings.

#### 3.2.1 drms.const

```
const = <drms.json.JsocInfoConstants object>
```

Constants for DRMS queries.
Attributes

all = '*.ALL**'
none = '*.NONE**'
recdir = '*.recdir**'
dirmttime = '*.dirmttime**'
logdir = '*.logdir**'
recnum = '*.recnum**'
sunum = '*.sunum**'
siize = '*.size**'
online = '*.online**'
retain = '*.retain**'
archive = '*.archive**'

3.2.2 drms.to_datetime

drms.to_datetime(tstr, force=False)
Parse JSOC time strings.

In general, this is quite complicated, because of the many different (non-standard) time strings supported by the DRMS. For more (much more!) details on this matter, see Rick Bogart’s notes.

The current implementation only tries to convert typical HMI time strings, with a format like “%Y.%m.%d_%H:%M:%S_TAI”, to an ISO time string, that is then parsed by pandas. Note that “_TAI”, as well as other timezone indentifiers like “Z”, will not be taken into account, so the result will be a naive timestamp without any associated timezone.

If you know the time string format, it might be better calling pandas.to_datetime() directly. For handling TAI timestamps, e.g. converting between TAI and UTC, the astropy.time package can be used.

Parameters

tstr [string or list/Series of strings] DateTime strings.
force [bool] Set to True to omit the endswith('_TAI') check.

Returns

result [pandas.Series or pandas.Timestamp] Pandas series or a single Timestamp object.

3.3 Exceptions

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrmsError</td>
<td>Unspecified DRMS run-time error.</td>
</tr>
<tr>
<td>DrmsQueryError</td>
<td>DRMS query error.</td>
</tr>
<tr>
<td>DrmsExportError</td>
<td>DRMS data export error.</td>
</tr>
</tbody>
</table>

3.3.1 drms.DrmsError

exception drms.DrmsError
Unspecified DRMS run-time error.
### 3.3.2 drms.DrmsQueryError

**exception** `drms.DrmsQueryError`

DRMS query error.

### 3.3.3 drms.DrmsExportError

**exception** `drms.DrmsExportError`

DRMS data export error.
Index

C
check_email() (drms.Client method), 14
Client (class in drms), 13
const (in module drms), 20

D
download() (drms.ExportRequest method), 18
drms (module), 13
DrmsError, 21
DrmsExportError, 22
DrmsQueryError, 22

E
export() (drms.Client method), 14
export_from_id() (drms.Client method), 15
ExportRequest (class in drms), 18

G
get() (drms.Client method), 15

H
has_failed() (drms.ExportRequest method), 19
has_finished() (drms.ExportRequest method), 19
has_succeeded() (drms.ExportRequest method), 20

I
info() (drms.Client method), 15

K
keys() (drms.Client method), 16

P
pkeys() (drms.Client method), 16

Q
query() (drms.Client method), 16

S
series() (drms.Client method), 17

SeriesInfo (class in drms), 17

to_datetime() (in module drms), 21

wait() (drms.ExportRequest method), 20