
hips Documentation

Release 0.2

HiPS developers

Oct 28, 2017

I	About	3
II	Installation	13
III	Getting started	25
IV	Reference/API	41
V	Changelog	63
VI	Develop	69
VII	HiPS tile drawing	79

This is the documentation page for hips. A Python astronomy package for HiPS : Hierarchical Progressive Surveys.

Part I

About

CHAPTER 1

Description

HiPS (Hierarchical Progressive Surveys) is a way to store large astronomical survey sky image and catalog datasets on servers (such as [HiPS at CDS](#)), that allows clients to efficiently fetch only the image tiles or catalog parts for a given region of the sky they are interested in. Similar to Google maps, but for astronomy (see the [HiPS paper](#)).

This is a Python package to fetch and draw HiPS data.

It was just started in summer of 2017 and isn't stable or feature complete yet. Feedback and contributions welcome!

CHAPTER 2

Links

- Code : <https://github.com/hipspy/hips>
- Docs : <https://hips.readthedocs.io>
- Contributors : <https://github.com/hipspy/hips/graphs/contributors>
- Releases: <https://pypi.python.org/pypi/hips>

CHAPTER 3

Other resources

- GSoC 2017 blog by Adeel: <https://adl1995.github.io>
- [HiPS at CDS](#) (contains a list and preview of available HiPS data)
- [HiPS paper](#)
- [HiPS IVOA recommendation](#)
- A Jupyter widget for Aladin Lite: <https://github.com/cds-astro/ipyaladin>
- Small example HiPS datasets we use for testing and docs examples: <https://github.com/hipsipy/hips-extra>

(If you have a HiPS-related webpage or tool or resource you'd like mentioned here, let us know!)

CHAPTER 4

Thanks

This package is being developed as part of Google Summer of Code 2017 program by Adeel Ahmad, with Thomas Boch (CDS, Strasbourg) and Christoph Deil (MPIK, Heidelberg) as mentors. We would like to thank Google, CDS, MPIK for their support!

If you're interested, you should follow Adeel's blog: <https://adl1995.github.io/>

Also: thanks to the Astropy team for developing and maintaining the affiliated package-template and the ci-helpers! The recently introduced cookie-cutter makes it even quicker to set up a new package like this one in a good, maintainable way.

Part II

Installation

The **hips** package works with Python 3.6 or later, on Linux, MacOS and Windows.

Installing the latest stable version is possible either using pip or conda.

How to install the latest development version is described on the [Develop](#) page.

CHAPTER 5

Using pip

To install hips with `pip` from `PyPI`, run:

```
pip install hips --no-deps
```

Note: The `--no-deps` flag is optional, but highly recommended if you already have Numpy installed, since otherwise pip will sometimes try to “help” you by upgrading your Numpy installation, which may not always be desired.

CHAPTER 6

Using conda

To install hips with [Anaconda](#) from the [conda-forge](#) channel on [anaconda.org](#) simply run:

```
conda install -c conda-forge hips
```


CHAPTER 7

Check installation

To check if you have hips installed, where it was installed and which version you have:

```
$ python
>>> import hips # doctest: +SKIP
>>> hips.__version__ # doctest: +SKIP
# -> prints which version you have
>>> hips # doctest: +SKIP
# -> prints where hips is installed
```

To see if you have the latest stable, released version of hips, you can find that version here:

- <https://pypi.python.org/pypi/hips>
- <https://anaconda.org/conda-forge/hips>

Next you could try running the examples at *Getting started* and see if you get the expected output.

It's usually not necessary, but if you find that your hips installation gives errors or unexpected results for examples that should work, you can run the hips automated tests via:

```
python -c 'import hips; hips.test()'
```

For more information on automated tests, see the *Develop* page.

Dependencies

The hips package has the following requirements:

- Python 3.6 or later!
- [Numpy](#) 1.11 or later
- [Astropy](#) 1.2 or later
- [astropy-healpix](#) 0.2 or later
- [scikit-image](#) 0.12 or later. (Older versions could work, but aren't tested.)
- [Pillow](#) 4.0 or later. (Older versions could work, but aren't tested.) Pillow is the friendly Python Imaging Library (PIL) fork, for JPEG and PNG tile I/O.

In addition, the following packages are needed for optional functionality:

- [Matplotlib](#) 2.0 or later. Used for plotting in examples.
- [tqdm](#). Used for showing progress bar either on terminal or in Jupyter notebook.
- [aiohttp](#). Used for fetching HiPS tiles.

We have some info at [Why only Python 3?](#) on why we don't support legacy Python (Python 2).

Part III

Getting started

This is a quick getting started guide for the Python [hips](#) package.

Make a sky image

To draw a sky image from HiPS image tiles with the `hips` package, follow the following three steps:

1. Specify the sky image geometry you want by creating a `WCSGeometry` object:

```
from astropy.coordinates import SkyCoord
from hips import WCSGeometry

geometry = WCSGeometry.create(
    skydir=SkyCoord(0, 0, unit='deg', frame='galactic'),
    width=2000, height=1000, fov="3 deg",
    coordsys='galactic', projection='AIT',
)
```

2. Specify the HiPS survey you want. You just need to provide a valid HiPS survey ID.

A good address that lists available HiPS data is <http://aladin.u-strasbg.fr/hips/list>

```
hips_survey = 'CDS/P/DSS2/red'
```

3. Call the `make_sky_image` function to fetch the HiPS data and draw it, returning an object of `HipsDrawResult`. By default a progress bar is shown for fetching and drawing HiPS tiles. For batch processing, this can be turned off by passing a keyword argument: `progress_bar=False`:

```
from hips import make_sky_image
result = make_sky_image(geometry, hips_survey, 'fits')
```

Of course, you could change the parameters to choose any sky image geometry and available HiPS survey you like!

CHAPTER 10

Work with the result

The `HipsDrawResult` object from the last section makes it easy for you to plot, save or analyse the sky image. To generate a quick-look plot of the sky image, with rectangles outlining the HiPS tiles that were fetched and drawn to create the sky image:

```
result.plot()
```

this will result in the following plot:

To save the sky image to a file:

```
result.write_image('my_image.fits')
```

To analyse the data, or make a publication-quality plot, you can get the sky image pixel data as a `numpy.ndarray`:

```
>>> result.image
```

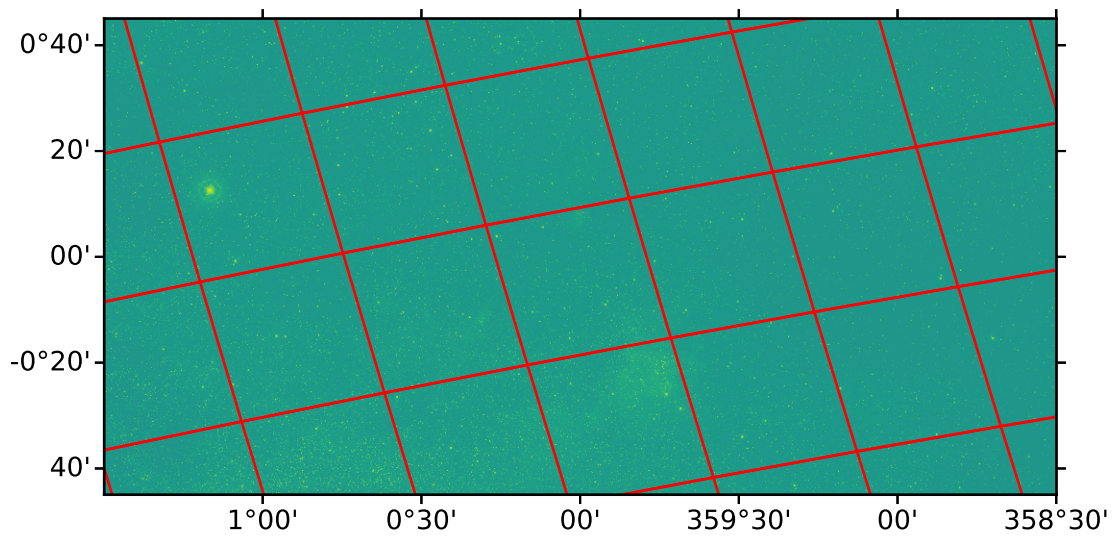
and the sky image `astropy.wcs.WCS` mapping pixel to sky coordinates via:

```
>>> result.geometry.wcs
```

To print out summary information about the result:

```
>>> print(result)
```

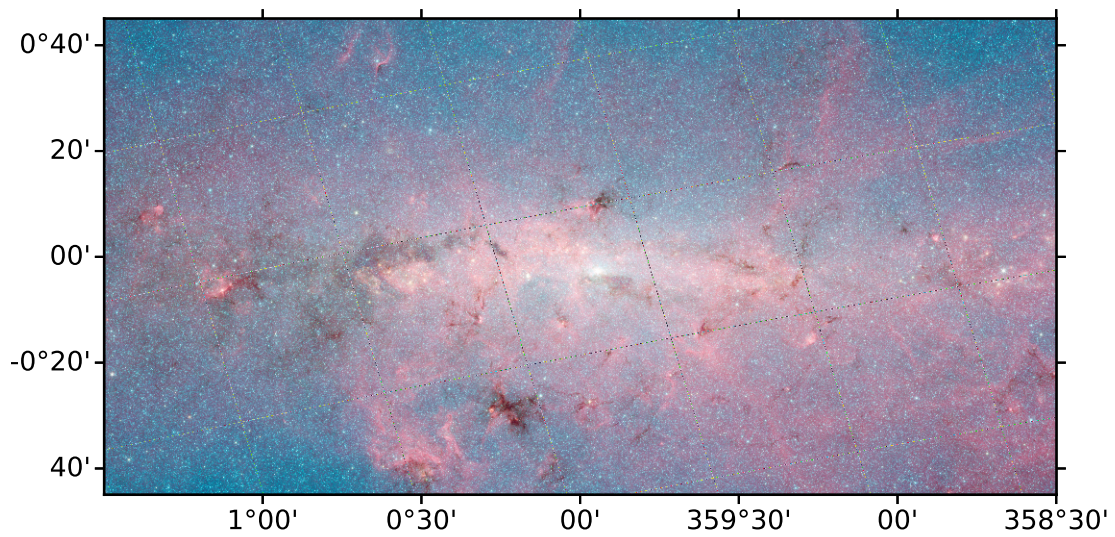
The `HipsDrawResult` object also gives access to the `HipsTile` objects that were used for drawing the sky image, as well as other things.



Make a color sky image

HiPS supports color images in jpg and png format. Making a color sky image works the same as the grayscale image example above, except that you get back a 3-dim Numpy array with (R, G, B) channels for jpg or (R, G, B, A) channels (A is transparency) for png.

Here's an example using jpg and <http://alasky.u-strasbg.fr/Fermi/Color/> :



CHAPTER 12

HiPS data

We plan to implement functionality to manage HiPS data, i.e. download it and cache it on a local disk. This isn't available yet, at the moment we simply use Python lists of `HipsTile` objects, which have a `read` method for a given filename and a `fetch` method for a given URL.

CHAPTER 13

More advanced examples

This package is under heavy development, it's changing quickly.

We'll add advanced examples and detailed documentation once things have stabilised a bit.

For now, if you know Python, you can look at the code and tests to see what's available: <https://github.com/hipspy/hips>

CHAPTER 14

What next?

That's it, now you've seen the main features of the [hips](#) package. Note that there is API documentation explaining all available functions, classes and parameters.

If you have any questions, or find something not working or a missing feature, please get in touch by posting on our Github issue tracker.

Part IV

Reference/API

A Python astronomy package for HiPS : Hierarchical Progressive Surveys.

At the moment a client for HiPS images, but other contributions (HiPS catalogs or HiPS image generation) welcome!

- Code : <https://github.com/hips/hips>
- Docs : <https://hips.readthedocs.io>
- License : BSD-3 (see licenses folder for license file)

15.1 Functions

<code>fetch_tiles(...)</code>	Fetch a list of HiPS tiles.
<code>make_sky_image((geometry: typing.Union[dict, ...])</code>	Make sky image: fetch tiles and draw.
<code>test([package, test_path, args, plugins, ...])</code>	Run the tests using <code>py.test</code> .

15.1.1 fetch_tiles

`hips.fetch_tiles(tile metas: typing.List[hips.tiles.tile.HipsTileMeta], hips_survey: hips.tiles.survey.HipsSurveyProperties, progress_bar: bool = True, n_parallel: int = 5, timeout: float = 10, fetch_package: str = 'urllib') → typing.List[hips.tiles.tile.HipsTile]`

Fetch a list of HiPS tiles.

This function fetches a list of HiPS tiles based on their URLs, which are generated using `hips_survey` and `tile_metas`.

The tiles are then fetched asynchronously using `urllib` or `aiohttp`.

Parameters

tile_metas : list

Python list of `HipsTileMeta`

hips_survey : `HipsSurveyProperties`

HiPS survey properties

progress_bar : bool

Show a progress bar for tile fetching and drawing

n_parallel : int

Number of tile fetch web requests to make in parallel

timeout : float

Seconds to timeout for fetching a HiPS tile

fetch_package : {'urllib', 'aiohttp'}

Package to use for fetching HiPS tiles

Returns

tiles : list

A Python list of [HipsTile](#)

Examples

Define a list of tiles we want:

```
from hips import HipsSurveyProperties, HipsTileMeta
from hips import fetch_tiles
url = 'http://alaska.unistra.fr/DSS/DSS2Merged/properties'
hips_survey = HipsSurveyProperties.fetch(url)
tile_indices = [69623, 69627, 69628, 69629, 69630, 69631]
tile metas = []
for healpix_pixel_index in tile_indices:
    tile_meta = HipsTileMeta(
        order=7,
        ipix=healpix_pixel_index,
        frame=hips_survey.astropy_frame,
        file_format='fits',
    )
    tile_metas.append(tile_meta)
```

Fetch all tiles (in parallel):

```
tiles = fetch_tiles(tile_metas, hips_survey)
```

15.1.2 make_sky_image

```
hips.make_sky_image(geometry: typing.Union[dict, hips.utils.wcs.WCSGeometry], hips_survey: typing.Union[str, _ForwardRef('HipsSurveyProperties')], tile_format: str, precise: bool = False, progress_bar: bool = True, fetch_opts: dict = None) → hips.draw.ui.HipsDrawResult
```

Make sky image: fetch tiles and draw.

The example for this can be found on the [Getting started](#) page.

Parameters

geometry : dict or WCSGeometry

Geometry of the output image

hips_survey : str or [HipsSurveyProperties](#)

HiPS survey properties

tile_format : {'fits', 'jpg', 'png'}

Format of HiPS tile to use (some surveys are available in several formats, so this extra argument is needed)

precise : bool

Use the precise drawing algorithm

progress_bar : bool

Show a progress bar for tile fetching and drawing

fetch_opts : dict

Keyword arguments for fetching HiPS tiles. To see the list of passable arguments, refer to [fetch_tiles](#)

Returns

result : [HipsDrawResult](#)

Result object

15.1.3 test

`hips.test(package=None, test_path=None, args=None, plugins=None, verbose=False, pastebin=None, remote_data=False, pep8=False, pdb=False, coverage=False, open_files=False, **kwargs)`
Run the tests using [py.test](#). A proper set of arguments is constructed and passed to [pytest.main](#).

Parameters

package : str, optional

The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.

test_path : str, optional

Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

args : str, optional

Additional arguments to be passed to [pytest.main](#) in the args keyword argument.

plugins : list, optional

Plugins to be passed to [pytest.main](#) in the plugins keyword argument.

verbose : bool, optional

Convenience option to turn on verbose output from [py.test](#). Passing True is the same as specifying '-v' in args.

pastebin : {'failed', 'all', None}, optional

Convenience option for turning on [py.test](#) pastebin output. Set to 'failed' to upload info for failed tests, or 'all' to upload info for all tests.

remote_data : bool, optional

Controls whether to run tests marked with `@remote_data`. These tests use online data and are not run by default. Set to True to run these tests.

pep8 : bool, optional

Turn on PEP8 checking via the [pytest-pep8 plugin](#) and disable normal tests. Same as specifying '`--pep8 -k pep8`' in args.

pdb : bool, optional

Turn on PDB post-mortem analysis for failing tests. Same as specifying '`--pdb`' in args.

coverage : bool, optional

Generate a test coverage report. The result will be placed in the directory `htmlcov`.

open_files : bool, optional

Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the [psutil](#) package.

parallel : int, optional

When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the [pytest-xdist](#) plugin installed. Only available when using Astropy 0.3 or later.

kwargs

Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

15.2 Classes

HipsDrawResult ((image: <code>numpy.ndarray</code> , ...))	HiPS draw result object (sky image and more).
HipsPainter ((geometry: <code>typing.Union[dict, ...]</code>)	Paint a sky image from HiPS image tiles.
HipsSurveyProperties (...)	HiPS properties container.
HipsSurveyPropertiesList (...)	HiPS survey properties list.
HipsTile (...)	HiPS tile container.
HipsTileAllskyArray (...)	All-sky tile array container.
HipsTileMeta ((order: int, ipix: int, ...))	HiPS tile metadata.
WCSGeometry ((wcs: <code>astropy.wcs.wcs.WCS</code> , ...))	Sky image geometry: WCS and image shape.

15.2.1 HipsDrawResult

class `hips.HipsDrawResult`(image: `numpy.ndarray`, geometry: `hips.utils.wcs.WCSGeometry`, tile_format: `str`, tiles: `typing.List[hips.tiles.tile.HipsTile]`, stats: `dict`) → None

Bases: `object`

HiPS draw result object (sky image and more).

Parameters

image : `ndarray`

Sky image (the main result)

geometry : `WCSGeometry`

WCS geometry of the sky image

tile_format : {'fits', 'jpg', 'png'}

Format of HiPS tile

tiles : list

Python list of [HipsTile](#) objects that were used

stats : dict

Information including time for fetching and drawing HiPS tiles

Methods Summary

<code>fromPainter(...)</code>	Make a HipsDrawResult from a HipsTilePainter .
<code>plot()</code> -> None	Plot the all sky image and overlay HiPS tile outlines.
<code>report()</code> -> None	Print a brief report for the fetched data.
<code>write_image(filename: str) -> None</code>	Write image to file.

Methods Documentation

classmethod fromPainter(*painter*: [hips.draw.paint.HipsPainter](#)) → [hips.draw.ui.HipsDrawResult](#)
Make a [HipsDrawResult](#) from a [HipsTilePainter](#).

plot() → None
Plot the all sky image and overlay HiPS tile outlines.

Uses [astropy.visualization.wcsaxes](#).

report() → None
Print a brief report for the fetched data.

write_image(*filename*: str) → None
Write image to file.

Parameters

filename : str

Filename

15.2.2 HipsPainter

class hips.HipsPainter(*geometry*: [typing.Union\[dict, hips.utils.wcs.WCSGeometry\]](#), *hips_survey*: [typing.Union\[str, hips.tiles.survey.HipsSurveyProperties\]](#), *tile_format*: str, *precise*: bool = False, *progress_bar*: bool = True, *fetch_opts*: dict = None) → None

Bases: [object](#)

Paint a sky image from HiPS image tiles.

Paint HiPS tiles onto a sky image using a simple projective transformation method. The algorithm implemented is described here: [HiPS tile drawing](#).

Parameters

geometry : dict or [WCSGeometry](#)

An object of [WCSGeometry](#)

hips_survey : str or [HipsSurveyProperties](#)

HiPS survey properties

tile_format : {'fits', 'jpg', 'png'}

Format of HiPS tile

precise : bool

Use the precise drawing algorithm

progress_bar : bool

Show a progress bar for tile fetching and drawing

fetch_opts : dict

Keyword arguments for fetching HiPS tiles. To see the list of passable arguments, refer to [fetch_tiles](#)

Examples

```
>>> from astropy.coordinates import SkyCoord
>>> from hips import WCSGeometry
>>> from hips import HipsSurveyProperties
>>> from hips import HipsPainter
>>> geometry = WCSGeometry.create(
...     skydir=SkyCoord(0, 0, unit='deg', frame='icrs'),
...     width=2000, height=1000, fov='3 deg',
...     coordsys='icrs', projection='AIT',
... )
>>> url = 'http://alasky.unistra.fr/DSS/DSS2Merged/properties'
>>> hips_survey = HipsSurveyProperties.fetch(url)
>>> painter = HipsPainter(geometry, hips_survey, 'fits')
>>> painter.draw_hips_order
7
>>> painter.run()
>>> painter.image.shape
(1000, 2000)
```

Attributes Summary

<code>draw_hips_order</code>	Compute HiPS tile order matching a given image pixel size.
<code>image</code>	Computed sky image (ndarray).
<code>tile_indices</code>	Get list of index values for HiPS tiles.
<code>tiles</code>	List of HipsTile (cached on multiple access).

Methods Summary

<code>draw_all_tiles()</code>	Make an empty sky image and draw all the tiles.
<code>make_tile_list()</code>	
<code>plot_mpl_hips_tile_grid() -> None</code>	Plot output image and HiPS grid with matplotlib.
<code>projection(...)</code>	Estimate projective transformation on a HiPS tile.

Continued on next page

Table 15.5 – continued from previous page

<code>run()</code> → <code>numpy.ndarray</code>	Draw HiPS tiles onto an empty image.
<code>warp_image(...)</code>	Warp a HiPS tile and a sky image.

Attributes Documentation

`draw_hips_order`

Compute HiPS tile order matching a given image pixel size.

`image`

Computed sky image (`ndarray`).

- The dtype is always chosen to match the tile dtype. This is `uint8` for JPG or PNG tiles, and can be e.g. `int16` or `float32` for FITS tiles.
- The output shape is documented here: `shape`.

`tile_indices`

Get list of index values for HiPS tiles.

`tiles`

List of `HipsTile` (cached on multiple access).

Methods Documentation

`draw_all_tiles()`

Make an empty sky image and draw all the tiles.

`make_tile_list()`

`plot_mpl_hips_tile_grid()` → `None`

Plot output image and HiPS grid with matplotlib.

This is mainly useful for debugging the drawing algorithm, not something end-users will call or need to know about.

`projection(tile: hips.tiles.tile.HipsTile)` → `skimage.transform._geometric.ProjectiveTransform`

Estimate projective transformation on a HiPS tile.

`run()` → `numpy.ndarray`

Draw HiPS tiles onto an empty image.

`warp_image(tile: hips.tiles.tile.HipsTile)` → `numpy.ndarray`

Warp a HiPS tile and a sky image.

15.2.3 HipsSurveyProperties

class `hips.HipsSurveyProperties(data: collections.OrderedDict)` → `None`

Bases: `object`

HiPS properties container.

Parameters

data : `OrderedDict`

HiPS survey properties

Examples

```
>>> from hips import HipsSurveyProperties
>>> url = 'http://alasky.unistra.fr/DSS/DSS2Merged/properties'
>>> hips_survey_property = HipsSurveyProperties.fetch(url)
>>> hips_survey_property.base_url
'http://alasky.u-strasbg.fr/DSS/DSS2Merged'
```

Attributes Summary

<code>astropy_frame</code>	Astropy coordinate frame (str).
<code>base_url</code>	HiPS access URL
<code>hips_frame</code>	HiPS coordinate frame (str).
<code>hips_order</code>	HiPS order (int).
<code>hips_service_url</code>	HiPS service base URL (str).
<code>hips_to_astropy_frame_mapping</code>	
<code>hips_version</code>	HiPS version (str).
<code>tile_format</code>	HiPS tile format (str).
<code>tile_width</code>	HiPS tile width
<code>title</code>	HiPS title (str).

Methods Summary

<code>fetch(...)</code>	Read from HiPS survey description file from remote URL (HipsSurveyProperties).
<code>from_name(...)</code>	Create object from Survey ID (HipsSurveyProperties).
<code>make((hips_survey: typing.Union[str, ...])</code>	Convenience constructor for <code>from_string</code> classmethod or existing object (HipsSurveyProperties).
<code>parse((text: str, ...)</code>	Parse HiPS survey description text (HipsSurveyProperties).
<code>read(...)</code>	Read from HiPS survey description file (HipsSurveyProperties).
<code>tile_url(...)</code>	Tile URL on the server (str).

Attributes Documentation

`astropy_frame`
Astropy coordinate frame (str).

`base_url`
HiPS access URL

`hips_frame`
HiPS coordinate frame (str).

`hips_order`
HiPS order (int).

`hips_service_url`
HiPS service base URL (str).

```
hips_to_astropy_frame_mapping = OrderedDict([('equatorial', 'icrs'), ('galactic', 'galactic'), ('ecliptic', 'ecliptic')])
```

hips_version

HiPS version (str).

tile_format

HiPS tile format (str).

tile_width

HiPS tile width

title

HiPS title (str).

Methods Documentation

classmethod **fetch**(*url: str*) → hips.tiles.survey.HipsSurveyProperties

Read from HiPS survey description file from remote URL ([HipsSurveyProperties](#)).

Parameters

url : str

URL containing HiPS properties

classmethod **from_name**(*name: str*) → hips.tiles.survey.HipsSurveyProperties

Create object from Survey ID ([HipsSurveyProperties](#)).

classmethod **make**(*hips_survey: typing.Union[str, _ForwardRef('HipsSurveyProperties')]*) → hips.tiles.survey.HipsSurveyProperties

Convenience constructor for from_string classmethod or existing object ([HipsSurveyProperties](#)).

classmethod **parse**(*text: str, url: str = None*) → hips.tiles.survey.HipsSurveyProperties

Parse HiPS survey description text ([HipsSurveyProperties](#)).

Parameters

text : str

Text containing HiPS survey properties

url : str

Properties URL of HiPS

classmethod **read**(*filename: str*) → hips.tiles.survey.HipsSurveyProperties

Read from HiPS survey description file ([HipsSurveyProperties](#)).

Parameters

filename : str

HiPS properties filename

tile_url(*tile_meta: hips.tiles.tile.HipsTileMeta*) → str

Tile URL on the server (str).

15.2.4 HipsSurveyPropertiesList

class hips.**HipsSurveyPropertiesList**(*data: typing.List[hips.tiles.survey.HipsSurveyProperties]*) → None

Bases: [object](#)

HiPS survey properties list.

Parameters**data** : listPython list of [HipsSurveyProperties](#)**Examples**

Fetch the list of available HiPS surveys from CDS:

```
>>> from hips import HipsSurveyPropertiesList
>>> surveys = HipsSurveyPropertiesList.fetch()
```

Look at the results:

```
>>> len(surveys.data)
335
>>> survey = surveys.data[0]
>>> survey.title
'2MASS H (1.66 microns)'
>>> survey.hips_order
9
```

You can make a `astropy.table.Table` of available HiPS surveys:

```
>>> table = surveys.table
```

and then do all the operations that Astropy table supports, e.g.

```
>>> table[['ID', 'hips_order', 'hips_service_url']][1, 30, 42]
>>> table.show_in_browser(jsviewer=True)
>>> table.show_in_notebook()
>>> table.to_pandas()
```

Attributes Summary

DEFAULT_URL	
table	Table with HiPS survey infos (Table).

Methods Summary

fetch(...)	Fetch HiPS list text from remote location (HipsSurveyPropertiesList).
from_name(...)	Return a matching HiPS survey (HipsSurveyProperties).
parse(...)	Parse HiPS list text (HipsSurveyPropertiesList).
read(...)	Read HiPS list from file (HipsSurveyPropertiesList).

Attributes Documentation

`DEFAULT_URL = 'http://alaska.unistra.fr/MocServer/query?hips_service_url=*&dataprodut_type=!catalog&data'`

table

Table with HiPS survey infos ([Table](#)).

Methods Documentation

classmethod `fetch(url: str = None) → hips.tiles.survey.HipsSurveyPropertiesList`

Fetch HiPS list text from remote location ([HipsSurveyPropertiesList](#)).

Parameters

url : str

HiPS list URL

from_name(*name: str*) → hips.tiles.survey.HipsSurveyProperties

Return a matching HiPS survey ([HipsSurveyProperties](#)).

classmethod `parse(text: str) → hips.tiles.survey.HipsSurveyPropertiesList`

Parse HiPS list text ([HipsSurveyPropertiesList](#)).

Parameters

text : str

HiPS list text

classmethod `read(filename: str) → hips.tiles.survey.HipsSurveyPropertiesList`

Read HiPS list from file ([HipsSurveyPropertiesList](#)).

Parameters

filename : str

HiPS list filename

15.2.5 HipsTile

class `hips.HipsTile(meta: hips.tiles.tile.HipsTileMeta, raw_data: bytes) → None`

Bases: [object](#)

HiPS tile container.

This class provides methods for fetching, reading, and writing a HiPS tile.

Note: In HiPS, the pixel data is flipped in the y direction for jpg and png format with respect to FITS. In this package, we handle this by flipping jpg and png data to match the fits orientation, at the I/O boundary, i.e. in [from_numpy](#) and [to_numpy](#).

Parameters

meta : [HipsTileMeta](#)

Metadata of HiPS tile

raw_data : [bytes](#)

Raw data (copy of bytes from file)

Examples

Fetch a HiPS tile:

```
>>> from hips import HipsTile, HipsTileMeta
>>> meta = HipsTileMeta(order=6, ipix=30889, file_format='fits')
>>> url = 'http://alasky.unistra.fr/2MASS/H/Norder6/Dir30000/Npix30889.fits'
>>> tile = HipsTile.fetch(meta, url)
```

The tile pixel data is available as a Numpy array:

```
>>> type(tile.data)
numpy.ndarray
>>> tile.data.shape
(512, 512)
>>> tile.data.dtype.name
int16
```

Attributes Summary

<code>children</code>	Create four children tiles from parent tile.
<code>data</code>	Tile pixel data (<code>ndarray</code>).

Methods Summary

<code>fetch((meta: hips.tiles.tile.HipsTileMeta, ...))</code>	Fetch HiPS tile and load into memory (<code>HipsTile</code>).
<code>from_numpy(...)</code>	Create a tile from given pixel data.
<code>read((meta: hips.tiles.tile.HipsTileMeta, ...))</code>	Read HiPS tile data from a directory and load into memory (<code>HipsTile</code>).
<code>to_numpy((raw_data: bytes, ...))</code>	Convert raw image bytes to Numpy array pixel data.
<code>write((filename: str) -> None)</code>	Write to file.

Attributes Documentation

`children`

Create four children tiles from parent tile.

`data`

Tile pixel data (`ndarray`).

This is a cached property, it will only be computed once.

See the `to_numpy` function.

Methods Documentation

classmethod `fetch(meta: hips.tiles.tile.HipsTileMeta, url: str) → hips.tiles.tile.HipsTile`
Fetch HiPS tile and load into memory (`HipsTile`).

Parameters

`meta` : `HipsTileMeta`

Metadata of HiPS tile

url : str

URL containing HiPS tile

classmethod from_numpy(*meta*: *hips.tiles.tile.HipsTileMeta*, *data*: *numpy.ndarray*) → *hips.tiles.tile.HipsTile*

Create a tile from given pixel data.

Parameters

meta : *HipsTileMeta*

Metadata of HiPS tile

data : *ndarray*

Tile pixel data

Returns

tile : *HipsTile*

HiPS tile object in the format requested in meta.

classmethod read(*meta*: *hips.tiles.tile.HipsTileMeta*, *filename*: *str* = *None*) → *hips.tiles.tile.HipsTile*

Read HiPS tile data from a directory and load into memory (*HipsTile*).

Parameters

meta : *HipsTileMeta*

Metadata of HiPS tile

filename : str

Filename

static to_numpy(*raw_data*: *bytes*, *fmt*: *str*) → *numpy.ndarray*

Convert raw image bytes to Numpy array pixel data.

Parameters

raw_data : bytes

Raw image bytes (usually read from file or fetched from URL)

fmt : { 'fits', 'jpg', 'png' }

File format

Returns

data : *ndarray*

Pixel data as a numpy array

write(*filename*: *str*) → None

Write to file.

Parameters

filename : str

Filename

15.2.6 HipsTileAllskyArray

class *hips.HipsTileAllskyArray*(*meta*: *hips.tiles.tile.HipsTileMeta*, *raw_data*: *bytes*) → None

Bases: *hips.HipsTile*

All-sky tile array container.

To quote from section 4.3.2 “Allsky preview file” of the HiPS IVOA working draft: “The tiles at low orders (0 to 3) may be packaged together into a unique file called Allsky.”

This class implements that all-sky tile array format.

TODO: We’re sub-classing `HipsTile` here at the moment. This is weird! Probably the current `HipsTile` should be renamed `ImageIO` or be split up into functions that do image I/O in the three supported formats?

TODO: We re-use the `HipsTileMeta` class to store order as well as other info like `file_format` and `frame`. Note that `ipix` doesn’t make sense for an `AllSkyTileArray`. Maybe there’s a better way to handle this without code duplication?

Examples

Load an example existing HiPS all-sky image (unfortunately one has to pass a dummy `ipix` value here):

```
>>> from hips import HipsTileAllskyArray, HipsTileMeta
>>> meta = HipsTileMeta(order=3, ipix=-1, file_format='jpg', frame='icrs')
>>> url = 'http://alasky.unistra.fr/Fermi/Color/Norder3/Allsky.jpg'
>>> allsky = HipsTileAllskyArray.fetch(meta, url)
```

Now you can extract tiles (e.g. for drawing):

```
>>> tile = allsky.tile(ipix=42)
>>> tile.meta
HipsTileMeta(order=3, ipix=42, file_format='jpg', frame='icrs', width=64)
```

TODO: add an example how to go the other way: combine tiles into an allsky image.

Attributes Summary

<code>height</code>	Image pixel height (int)
<code>n_tiles</code>	Number of tiles in the image (int).
<code>n_tiles_in_row</code>	Number of tiles per tile row (int).
<code>tile_width</code>	Pixel width of a single tile (int).
<code>tiles</code>	Split into a list of <code>HipsTile</code> .
<code>width</code>	Image pixel width (int).

Methods Summary

<code>from_tiles(...)</code>	Create all-sky image from list of tiles.
<code>tile(ipix: int) -> hips.tiles.tile.HipsTile</code>	Extract one of the tiles (<code>HipsTile</code>)
<code>tiles_to_allsky_array(...)</code>	Combine tiles into an all-sky image.

Attributes Documentation

`height`

Image pixel height (int)

`n_tiles`

Number of tiles in the image (int).

n_tiles_in_row
Number of tiles per tile row (int).

tile_width
Pixel width of a single tile (int).

tiles
Split into a list of [HipsTile](#).
This is called when using the all-sky image for drawing.

width
Image pixel width (int).

Methods Documentation

classmethod from_tiles(*tiles:* *typing.List[hips.tiles.tile.HipsTile]*) → *hips.tiles.allsky.HipsTileAllskyArray*
Create all-sky image from list of tiles.

tile(*ipix: int*) → *hips.tiles.tile.HipsTile*
Extract one of the tiles ([HipsTile](#))
A copy of the data by default. For drawing we could avoid the copy by passing `copy=False` here.

static tiles_to_allsky_array(*tiles: typing.List[hips.tiles.tile.HipsTile]*) → *numpy.ndarray*
Combine tiles into an all-sky image.

15.2.7 HipsTileMeta

class `hips.HipsTileMeta`(*order: int, ipix: int, file_format: str, frame: str = 'icrs', width: int = 512*) → *None*

Bases: [object](#)

HiPS tile metadata.

Parameters

order : int
HEALPix order

ipix : int
HEALPix pixel number

file_format : {'fits', 'jpg', 'png'}
File format

frame : {'icrs', 'galactic', 'ecliptic'}
Sky coordinate frame

width : int
Tile width (tiles always have width = height)

Examples

```
>>> from hips import HipsTileMeta
>>> tile_meta = HipsTileMeta(order=3, ipix=450, file_format='fits')
>>> tile_meta
HipsTileMeta(order=3, ipix=450, file_format='fits', frame='icrs', width=512)
>>> tile_meta.skycoord_corners
<SkyCoord (ICRS): (ra, dec) in deg
[( 264.375, -24.62431835), ( 258.75 , -30.      ),
 ( 264.375, -35.68533471), ( 270.    , -30.      )]>
>>> tile_meta.tile_default_url
'Norder3/Dir0/Npix450.fits'
>>> tile_meta.tile_default_path.as_posix()
'Norder3/Dir0/Npix450.fits'
```

Attributes Summary

<code>skycoord_corners</code>	Tile corner sky coordinates (SkyCoord).
<code>tile_default_path</code>	Tile relative filename path (Path).
<code>tile_default_url</code>	Tile relative URL (str).

Methods Summary

<code>copy()</code>	An independent copy.
---------------------	----------------------

Attributes Documentation

`skycoord_corners`

Tile corner sky coordinates ([SkyCoord](#)).

`tile_default_path`

Tile relative filename path ([Path](#)).

`tile_default_url`

Tile relative URL (str).

Methods Documentation

`copy()`

An independent copy.

15.2.8 WCSGeometry

class `hips.WCSGeometry`(*wcs: astropy.wcs.wcs.WCS*, *width: int*, *height: int*) → None

Bases: [object](#)

Sky image geometry: WCS and image shape.

Parameters

wcs : [WCS](#)

WCS projection object

width, height : int

Width and height of the image in pixels

Examples

To create a `WCSGeometry`, you can create any `WCS` and choose an image shape (number of pixels):

```
from astropy.wcs import WCS
from hips import WCSGeometry
wcs = WCS(naxis=2)
wcs.wcs.ctype[0] = 'GLON-AIT'
wcs.wcs.ctype[1] = 'GLAT-AIT'
wcs.wcs.crval[0] = 0
wcs.wcs.crval[1] = 0
wcs.wcs.crpix[0] = 1000
wcs.wcs.crpix[1] = 500
wcs.wcs.cdelt[0] = -0.01
wcs.wcs.cdelt[1] = 0.01
geometry = WCSGeometry(wcs, width=2000, height=1000)
```

See also `WCSGeometry.create` as a simpler (but also not quite as flexible) way to generate `WCS` and `WCSGeometry` objects.

Attributes Summary

<code>WCS_ORIGIN_DEFAULT</code>	
<code>celestial_frame</code>	Celestial frame for the given <code>WCS</code> (str).
<code>center_pix</code>	Image center in pixel coordinates (tuple of x, y).
<code>center_skycoord</code>	Image center in sky coordinates (<code>SkyCoord</code>).
<code>fits_header</code>	FITS header for the given <code>WCS</code> (<code>Header</code>).
<code>pixel_skycoords</code>	Grid of sky coordinates of the image pixels (<code>SkyCoord</code>).

Methods Summary

<code>create(...)</code>	Create <code>WCS</code> object for given sky image parameters (<code>WCSGeometry</code>).
<code>create_from_dict(...)</code>	Create <code>WCS</code> object from a dictionary (<code>WCSGeometry</code>).
<code>make((geometry: typing.Union[dict, ...])</code>	Convenience constructor for <code>create_from_dict</code> class-method or existing object (<code>WCSGeometry</code>).
<code>pix_to_sky((x, ...)</code>	Helper function to convert pix to sky coordinates.

Attributes Documentation

`WCS_ORIGIN_DEFAULT` = 0

celestial_frame

Celestial frame for the given `WCS` (str).

Calls `wcs_to_celestial_frame`.

center_pix

Image center in pixel coordinates (tuple of x, y).

center_skycoord

Image center in sky coordinates (`SkyCoord`).

fits_header

FITS header for the given WCS (`Header`).

pixel_skycoords

Grid of sky coordinates of the image pixels (`SkyCoord`).

Methods Documentation

classmethod create(*skydir: astropy.coordinates.sky_coordinate.SkyCoord, width: int, height: int, fov: typing.Union[str, astropy.coordinates.angles.Angle], coordsys: str = 'icrs', projection: str = 'AIT'*) → `hips.utils.wcs.WCSGeometry`
Create WCS object for given sky image parameters (`WCSGeometry`).

Parameters

skydir : `SkyCoord`

Sky coordinate of the WCS reference point

width, height : int

Width and height of the image in pixels

fov : str or `Angle`

Field of view

coordsys : { 'icrs', 'galactic' }

Coordinate system

projection : str

Projection of the WCS object. To see list of supported projections visit: <http://docs.astropy.org/en/stable/wcs/#supported-projections>

Examples

```
>>> from astropy.coordinates import SkyCoord
>>> from hips import WCSGeometry
>>> skycoord = SkyCoord(10, 20, unit='deg')
>>> geometry = WCSGeometry.create(
...     skydir=SkyCoord(0, 0, unit='deg', frame='galactic'),
...     width=2000, height=1000, fov='3 deg',
...     coordsys='galactic', projection='AIT',
... )
>>> geometry.wcs
Number of WCS axes: 2
CTYPE : 'GLON-AIT' 'GLAT-AIT'
CRVAL : 0.0 0.0
CRPIX : 500.0 1000.0
PC1_1 PC1_2 : 1.0 0.0
PC2_1 PC2_2 : 0.0 1.0
```



```
CDELTA : -0.0015  0.0015
NAXIS : 0  0
>>> geometry.shape
Shape(width=2000, height=1000)
```

classmethod `create_from_dict(params: dict)` → hips.utils.wcs.WCSGeometry

Create WCS object from a dictionary ([WCSGeometry](#)).

The extra options are passed to the create class method, it can take the following parameters:

- target
- width
- height
- fov
- coordsys
- projection

For detailed description, see the create class method's docstring.

classmethod `make(geometry: typing.Union[dict, _ForwardRef('WCSGeometry')])` → hips.utils.wcs.WCSGeometry

Convenience constructor for `create_from_dict` classmethod or existing object ([WCSGeometry](#)).

pix_to_sky(x, y) → astropy.coordinates.sky_coordinate.SkyCoord

Helper function to convert pix to sky coordinates.

Part V

Changelog

CHAPTER 16

0.2

Version 0.2 of hips was released on October 28, 2017.

- Change from using healpy to astropy-healpix. This means hips now works on Windows! [#109]
- Introduce asynchronous fetching of HiPS tiles [#106]
- Add progress bar support for fetching and drawing HiPS tiles [#105]
- Add reporting functionality for HipsPainter [#104]
- Fix tile splitting criterion [#101]

CHAPTER 17

0.1

This first version of the hips package was released on July 28, 2017. It contains a first implementation to fetch and draw tiles.

This is a very early release, to get some users and feedback. Note that the API will change in the coming weeks, and you can also expect new features, fixes and performance and usability enhancements.

The hips package started as a project developed as part of Google summer of code 2017, i.e. planning in early 2017 and coding in June 2017.

Part VI

Develop

CHAPTER 18

Hello!

Want to contribute to the hips package?

Great! Talk to us by filing a Github issue any time (it doesn't have to be a concrete feature request or bug report).

This package was created using the Astropy affiliated package template, and everything works pretty much as in Astropy and most affiliated packages.

We didn't write any developer docs specifically for this package yet. For now, check out the Astropy core package developer docs, or just talk to us if you have any questions.

CHAPTER 19

Install development version

Install the latest development version from <https://github.com/hipspy/hips> :

```
git clone https://github.com/hipspy/hips
cd hips
pip install .
```

Then run the tests with either of these commands:

```
python -m pytest -v hips
python setup.py test -V
```

To run all tests and get a coverage report:

```
python setup.py test -V --remote-data --coverage
```

To build the documentation, do:

```
python setup.py build_docs
```


CHAPTER 20

Get the hips-extra test datasets

To run tests accessing files from [hips-extra](#) repository, users must have it cloned on their system, otherwise some test cases will be skipped. This contains tiles from different HiPS surveys which are used by the drawing module. After this, the HIPS_EXTRA environment variable must be set up on their system. On UNIX operating systems, this can be set using

```
export HIPS_EXTRA=path/to/hips-extra
```


CHAPTER 21

Why only Python 3?

This package requires Python 3.6 or later.

It will not work with Python 2.7 or 3.5!

This was a choice we made when starting this package in summer of 2017, at a time when Jupyter had just made their Python 3 only release and other packages we depend on (like Astropy) were about to drop support for legacy Python (Python 2).

Supporting only Python 3 means we e.g. get these benefits:

- `async / await` for asynchronous HiPS tile fetching (introduced in Python 3.5)
- Keyword-only arguments (introduced in Python 3.0)
- Type annotations (some only introduced in Python 3.6)
- f-strings (introduced in Python 3.6)

At the moment, the only Python 3.6 only feature we use are f-strings, so if several potential users that are on Python 3.5 and can't easily upgrade for some reason complain, we might consider supporting Python 3.5 in the next release.

Part VII

HiPS tile drawing

This section describes the HiPS tile drawing algorithm implemented in this package, to create a sky image for a given WCS.

The original description was for the algorithm implemented in Aladin Lite, written by Thomas Boch. In the meantime, the algorithm implemented in this Python package has deviated a bit, they are no longer the same.

The implementation is based on `numpy`, `astropy`, `healpy` and `scikit-image`.

Naive algorithm

This is a naive (one could also say: simple and fast) algorithm for drawing HiPS tiles using affine transformations, implemented in the `HipsPainter` class and usually executed by users via the high-level `make_sky_image` function.

First we compute and fetch the tiles that are needed for the given sky image:

1. The user specifies a `WCSGeometry`, which is a `astropy.wcs.WCS` as well as a width and height of the sky image to compute.
2. Compute HiPS order corresponding to the requested image size/resolution. The attributes of HiPS properties needed for this are `hips_order` (order at the tile level) and `hips_tile_width` (number of pixels for tile width and height). If `hips_tile_width` is missing, a default value of 512 is assumed.
3. Compute the list of tiles corresponding to the image FoV. This is done by computing the HiPS tile HEALPix pixel index for every pixel in the sky image and then computing the unique set.
4. Fetch (HTTP calls or from a local cache) all tiles in the list.

Then we draw the tiles one by one using these steps:

1. For each tile, compute the world coordinates of the tile corner vertices, using `healpy boundaries` function.
2. For each tile, project vertices in image coordinates according to the requested WCS (performing ICRS to Galactic frame transformation if the HiPS and sky image aren't in the same frame already).
3. We extend the tile by 1 pixel in all directions in order to hide “stitches” with other tiles drawing (TODO: not done yet. needed?)
4. The four corners uniquely define a projective transform between pixel coordinates on the tile and the output sky image. We use `scikit-image` to compute and apply that transform, which uses cubic spline interpolation under the hood. Thus the output is always float data, even if the input was integer RGB image data.

At the moment, we simply initialise the output sky image with pixel values of zero, and then sum the sky images we get from each projected tile. This is inefficient, and can result in incorrect pixel values at the lines corresponding to tile borders. We plan to implement a better (more efficient and more correct) way to handle that soon.

Note that any algorithm using interpolation is not fully conserving flux or counts. This might be a concern if you use the resulting sky images for data analysis. It's your responsibility to decide if using this method is appropriate for your application or not!

Tile distortion issue

While the algorithm previously described works fine for HiPS tiles not distorted, it brings some astrometry offsets for distorted tiles. This distortion is strongly visible in the HEALPix scheme for tiles at the boundary between the equatorial zone and the polar cap.

An example of such distortions is shown in the example below (uncheck *Activate deformations reduction algorithm* to view the astrometry offsets): <http://cds.unistra.fr/~boch/AL/test-reduce-deformations2.html>

To overcome this problem, Aladin Desktop and Aladin Lite use the following recursive strategy: for tiles either too large (one edge is >300 pixels when projected) or too distorted (ratio of smaller diagonal on larger diagonal is smaller than 0.7 and one of the diagonal is >150 pixels when projected):

- We consider 4 children tiles, dynamically generated from the pixels of their father. Each children tile has a width and height equal to half of its father's width/height.
- For each children tile, we compute the world coordinates of its vertices, project them and either draw it if not too distorted or repeat the process by splitting again into 4 children tiles.

The recursion is limited by a maximum number of recursive steps (for 512×512 tiles, you are limited to a maximum of 9 steps as $2^9=512$) and/or a maximum order (maximum order set arbitrarily at 19 in Aladin Desktop).

CHAPTER 24

Precise algorithm

Note: The precise algorithm isn't implemented yet.

Contrary the previous algorithm which used affine transformations, the idea here for the drawing step is to scan the result image pixels, and for each of them interpolate (Nearest neighbour or bilinear) the value, ie compute the indexes of nearest neighbour(s), retrieve the values of these pixels and merge them to determine the value of the target pixel. This is very similar to what [reproject](#) is doing.

One challenge is that one needs to know how to find the tile and pixel corresponding to a given HEALPix index. The correspondance between a HEALPix index and a pixel in a HiPS tile is given by a hpx2xy array (see method `createHpx2xy` in class `cds.tools.pixtools.Util` from [Aladin Desktop source code](#).)

WCS for FITS tiles

It seems that the astrometry of a HiPS tile can be accurately described using a WCS header like this one (example for HiPS in equatorial frame, Norder 3, Npix 448):

```
NAXIS      =                2 / number of data axes
NAXIS1     =               512 / length of data axis 1
NAXIS2     =               512 / length of data axis 1
CRPIX1     =          -2047.5 / Coordinate reference pixel
CRPIX2     =          -5631.5 / Coordinate reference pixel
CD1_1      = -1.0986328125000E-02 / Transformation matrix (rot + scale)
CD1_2      = -1.0986328125000E-02 / Transformation matrix (rot + scale)
CD2_1      =  1.0986328125000E-02 / Transformation matrix (rot + scale)
CD2_2      = -1.0986328125000E-02 / Transformation matrix (rot + scale)
CTYPE1     = 'RA---HPX'        / Longitude in an HPX projection
CTYPE2     = 'DEC--HPX'       / Latitude in an HPX projection
CRVAL1     =                0. / [deg] Longitude at the reference point
CRVAL2     =                0. / [deg] Latitude at the reference point
PV2_1      =                4 / HPX H parameter (longitude)
PV2_2      =                3 / HPX K parameter (latitude)
```

HPX projection is supported by WCSLib. It is understood by DS9. Support in other tools (reproject, Montage, etc) is unclear and has to be tested.

Note: It seems that we can define a WCS for each tile. If so, this would allow us to simply use the reproject package to draw the tiles, which would be an alternative “precise” algorithm.

h

hips, [43](#)

A

astropy_frame (hips.HipsSurveyProperties attribute), 50

B

base_url (hips.HipsSurveyProperties attribute), 50

C

celestial_frame (hips.WCSGeometry attribute), 59
center_pix (hips.WCSGeometry attribute), 60
center_skycoord (hips.WCSGeometry attribute), 60
children (hips.HipsTile attribute), 54
copy() (hips.HipsTileMeta method), 58
create() (hips.WCSGeometry class method), 60
create_from_dict() (hips.WCSGeometry class method), 61

D

data (hips.HipsTile attribute), 54
DEFAULT_URL (hips.HipsSurveyPropertiesList attribute), 53
draw_all_tiles() (hips.HipsPainter method), 49
draw_hips_order (hips.HipsPainter attribute), 49

F

fetch() (hips.HipsSurveyProperties class method), 51
fetch() (hips.HipsSurveyPropertiesList class method), 53
fetch() (hips.HipsTile class method), 54
fetch_tiles() (in module hips), 43
fits_header (hips.WCSGeometry attribute), 60
from_name() (hips.HipsSurveyProperties class method), 51
from_name() (hips.HipsSurveyPropertiesList method), 53
from_numpy() (hips.HipsTile class method), 55
from_painter() (hips.HipsDrawResult class method), 47
from_tiles() (hips.HipsTileAllskyArray class method), 57

H

height (hips.HipsTileAllskyArray attribute), 56

hips (module), 43

hips_frame (hips.HipsSurveyProperties attribute), 50
hips_order (hips.HipsSurveyProperties attribute), 50
hips_service_url (hips.HipsSurveyProperties attribute), 50

hips_to_astropy_frame_mapping
(hips.HipsSurveyProperties attribute), 50
hips_version (hips.HipsSurveyProperties attribute), 51
HipsDrawResult (class in hips), 46
HipsPainter (class in hips), 47
HipsSurveyProperties (class in hips), 49
HipsSurveyPropertiesList (class in hips), 51
HipsTile (class in hips), 53
HipsTileAllskyArray (class in hips), 55
HipsTileMeta (class in hips), 57

I

image (hips.HipsPainter attribute), 49

M

make() (hips.HipsSurveyProperties class method), 51
make() (hips.WCSGeometry class method), 61
make_sky_image() (in module hips), 44
make_tile_list() (hips.HipsPainter method), 49

N

n_tiles (hips.HipsTileAllskyArray attribute), 56
n_tiles_in_row (hips.HipsTileAllskyArray attribute), 57

P

parse() (hips.HipsSurveyProperties class method), 51
parse() (hips.HipsSurveyPropertiesList class method), 53
pix_to_sky() (hips.WCSGeometry method), 61
pixel_skycoords (hips.WCSGeometry attribute), 60
plot() (hips.HipsDrawResult method), 47
plot_mpl_hips_tile_grid() (hips.HipsPainter method), 49
projection() (hips.HipsPainter method), 49

R

read() (hips.HipsSurveyProperties class method), 51

`read()` (`hips.HipsSurveyPropertiesList` class method), 53
`read()` (`hips.HipsTile` class method), 55
`report()` (`hips.HipsDrawResult` method), 47
`run()` (`hips.HipsPainter` method), 49

S

`skycoord_corners` (`hips.HipsTileMeta` attribute), 58

T

`table` (`hips.HipsSurveyPropertiesList` attribute), 53
`test()` (in module `hips`), 45
`tile()` (`hips.HipsTileAllskyArray` method), 57
`tile_default_path` (`hips.HipsTileMeta` attribute), 58
`tile_default_url` (`hips.HipsTileMeta` attribute), 58
`tile_format` (`hips.HipsSurveyProperties` attribute), 51
`tile_indices` (`hips.HipsPainter` attribute), 49
`tile_url()` (`hips.HipsSurveyProperties` method), 51
`tile_width` (`hips.HipsSurveyProperties` attribute), 51
`tile_width` (`hips.HipsTileAllskyArray` attribute), 57
`tiles` (`hips.HipsPainter` attribute), 49
`tiles` (`hips.HipsTileAllskyArray` attribute), 57
`tiles_to_allsky_array()` (`hips.HipsTileAllskyArray` static method), 57
`title` (`hips.HipsSurveyProperties` attribute), 51
`to_numpy()` (`hips.HipsTile` static method), 55

W

`warp_image()` (`hips.HipsPainter` method), 49
`WCS_ORIGIN_DEFAULT` (`hips.WCSGeometry` attribute), 59
`WCSGeometry` (class in `hips`), 58
`width` (`hips.HipsTileAllskyArray` attribute), 57
`write()` (`hips.HipsTile` method), 55
`write_image()` (`hips.HipsDrawResult` method), 47