Astropysics Documentation

Release 0.1.dev-r1039

Erik Tollerud

December 07, 2012

Contents

Author Erik Tollerud

Astropysics is a library containing a variety of utilities and algorithms for reducing, analyzing, and visualizing astronomical data. Best of all, it encourages the user to leverage the existing capabilities of Python to make this quick, easy, and as painless as cutting-edge science can even actually be. There do exist other Python packages with some of the capabilities of this project, but the goal of this project is to integrate all these tools together and make them interact in the most straightforward ways possible.

(And to that end, if you are running one of those other projects, I'd love to help integrate our projects into a common framework!)

CHAPTER 1

Contents

Astropysics is divided into two major subcomponents - the core modules that contain functions and classes to the calculations and organize data, and the gui module that contains a number of useful small-scale astronomy applications.

1.1 Installing Astropysics

1.1.1 Requirements

Before you do anything with astropysics, you'll need:

- Python 2.5 or higher (2.6 highly recommended), although 3.x is not yet supported.
- numpy
- scipy

Follow the instructions on those sites, or, far easier, install them as packages from your operating system (e.g. apt-get or the synaptic GUI on Ubuntu, Macports on OS X, etc.).

1.1.2 Install

Once you have the requirements satisfied, you have a few options for installing astropysics.

Note: On most unix-like systems, you will need to either execute these commands as the root user, or preface them with `sudo`.

If you have pip (the new, better easy installer) or easy_install/setuptools (you should probably install pip...), just run either:

```
pip install astropysics
```

or:

```
easy_install astropysics
```

If you are installing from source code, instead, just do:

```
python setup.py install
```

If you plan on using the most up-to-date version of astropysics or if you wish to alter the source code (see the *Developer Guidelines for Astropysics*), a useful way to immediately see changes without having to re-install every time is to use the command:

```
python setup.py develop
```

1.1.3 Setup

After the package has been installed, at the command line, run:

```
astpys-setup
```

This script does two things:

- Prompts you to select which optional packages you want to download and install.
- Configures IPython to support the `ipyastpys` script (described in *Getting Started/Tutorial*).

The first of these involves an interactive process that downloads and installed requested packages. To install all of them, type `a` (and hit enter), otherwise choose a number to install that package. If you want to quit before installing all of the package (for example, if some don't install correctly), choose `q`. For information on a package, type `i#` (where # is the number for the package).

Note that if you can't get any packages to install, you might try running the script as:

```
astpys-setup -s
```

Depending on your operating system, you may want to use your package management system to install the recommended packages, before running the setup (although you may need the more up-to-date versions given here).

1.1.4 Recommended Packages

A number of other packages are necessary for added functionality in astropysics or to provide functionality that has no need to be duplicated. These packages can be installed with the `astpys-setup` script as described in *Setup*, but if available from your system's package management system, it may be better to try installing that way, instead.

- **Matplotlib** *highly recommended*, as it is necessary for all plotting (aside from the GUI applications).
- **IPython** *highly recommended*, as it is a far better interactive shell than the python default and has many wonderful features. Necessary for the `ipyastpys` script.
- **NetworkX** *highly recommended*, as it is used for a variety of internal purposes as well as any place where a network/graph is plotted.
- **PyGraphviz** It might also be useful to have a closely related package for generating graphviz graphs from networkx.
- **pyfits** *highly recommended*, necessary for reading FITS files (the most common astronomy data format).
- ***asciitable <http://cxc.cfa.harvard.edu/contrib/asciitable/>*** A valuable tool for loading and writing ASCII tables.
- ***ATpy <http://atpy.github.com/>*** Astronomical Tables in Python - a general tool for dealing with tabular data, both ASCII (uses asciitable) and other formats.
- ***pidly <http://astronomy.sussex.ac.uk/~anthonys/pidly/>*** IDL within Python. For those times when someone sends you an IDL code that you don't have the time to convert to python, but want to be able to call from inside python. Requires IDL to be installed.

- **Traits, TraitsGUI, Chaco, and Mayavi. Alternatively, ETS is all bundled in one.** Necessary for the interfaces in the `gui` modules:

  ```
  pip install ETS
  ```

  or:

  ```
  pip install traits
  pip install traitsGUI
  pip install chaco
  pip install mayavi
  ```

Astropysics also includes pythonic wrappers around some astronomy-related tools that need to be installed seperately if their functionality is desired:

- SExtractor
- Kcorrect

1.2 Getting Started/Tutorial

If you have not done so, install astropysics as described in *Installing Astropysics*, and be sure to run the setup command `astpys-setup` as described in *Setup*. Be sure you have IPython installed for the rest of this section to function correctly.

1.2.1 Interactive Environment

Astropysics uses IPython to provide an interactive environment to run python. To start using astropysics in ipython, just run the helper script `ipyastpys` - that will run ipython with a customized profile that automatically imports commonly-used parts of of astropysics (and `numpy`).

1.2.2 Projects

The `ipyastpys` environment also supports "projects", allowing the interactive environment to be started such that it automatically moves to a given directory and runs a given script. This directory can then hold all necessary data files and the script can load data and store functions to generate plots for the project/paper. A project can be created using the command:

```
ipyastpys -c projectname
```

This will create a project named "projectname", with a directory "/path/to/currentdir/projectname" and script "projectname/projectname.py". If the directory or script don't exist, they will be created. A different directory or script name can be used by calling the script as:

```
ipyastpys -c projectname /path/to/projectdir projectscript.py
```

Note that the script must be inside the project directory (in the above example, the script's full path is "/path/to/projectdir/projectscript.py").

The interactive environment should then be started using:

```
ipyastpys -p projectname
```

And it will start ipython in /path/to/projectdir, with the projectscript.py script automatically run inside the interactive environment (the `-s` option can be used to give projectscript.py any necessary command line arguments).

If a project script does not already exist, it will be generated from `a template` that allows for easy interactive work to create plots. In the script, add as many functions as necessary with the function decorator `plotfunc()` (see the template itself for detailed syntax), and then load any necessary data variables at the end of the script. The `make_plots()` function can then be used to generate the plots by passing it the name of the plot function. The script can also be used at the command line to auto-generate all plots.

1.2.3 Module Tutorials

Most of the modlues in astropysics have their own tutorials and examples that are relevant to their specific functionality. This is the best place to go to find examples of how to actually use astropysics. See *Core Modules* and *GUI applications* for a list of these modules.

Todo

Add links to the relevant tutorials in some automated way

1.3 Core Modules

The modules forming the core of astropysics' functionality are documented below. These are primarily useful in a script or interpreter setting, and hence are non-GUI. All plotting functions in these modules use matplotlib, allowing them to be used in ipython without blocking the interactive interpreter.

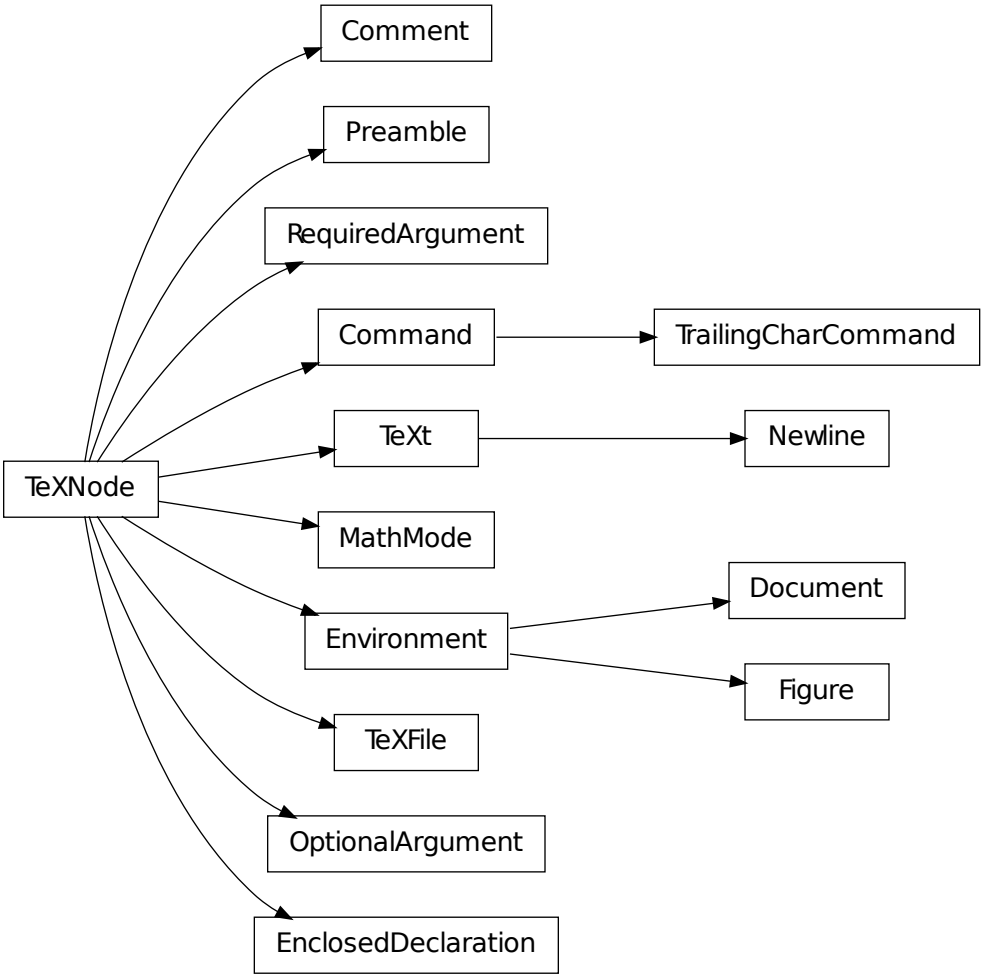
1.3.1 publication – tools for preparing astronomy papers in LaTeX

The `publication` module contains classes and funtions to assist in preparing papers or other publications with an emphasis on common astronomy journals. The tools here are for use with LaTeX as the actual authoring tools.

Todo

examples

Classes and Inheritance Structure



Module API

class astropysics.publication.**Command**(*parent*, *content*)
Bases: astropysics.publication.TeXNode

A LaTeX command (anything with leading backslash and possible arguments that isn't an environment)

Parameters

- **parent** – The parent node
- **content** – Either a string with the command text, or a (name,args) tuple where args is a sequence of strings or a string '{arg1}[oarg1]{arg2}...'

getSelfText()

optargs
A list of strings with the text of the optional arguments (arguments enclosed in square brackets)

reqargs
A list of strings with the text of the required arguments (arguments enclosed in curly braces).

class astropysics.publication.**Comment**(*parent*, *ctext*, *endswithnewline=False*)
Bases: astropysics.publication.TeXNode

A single-line comment of a TeX File. Note that unlike most

Parameters

- **parent** – The parent node
- **ctext** – The comment text (with or without an initial %)

children = ()

getSelfText()

text = ''
The text of this comment (not including the initial %)

class astropysics.publication.**Document**(*parent*, *content*, *envname=None*)
Bases: astropysics.publication.Environment

abstract = None
The abstract environment for this document or None if one does not exist

name = 'document'

sections = {}
A dictionary mapping section names to the associated index into

class astropysics.publication.**EnclosedDeclaration**(*parent*, *content*)
Bases: astropysics.publication.TeXNode

A TeX construct of the form {name{op}[op] content}. Note that declarations terminated by the end command will not be treated as this kind of object.

Parameters

- **parent** – The parent node
- **content** – A (padstr,commandnode,innercontent) tuple where *padstr* is the string before the command, *commandnode* is a Command object with the command portion of the declaration, and *innercontent* is the content after the command either as a string or a list of nodes (possibly mixed with strings). Alternatively, it can be the full text string including the outermost braces.

cmd = None
A Command object with the command in the declaration

getSelfText()

padstr = ''
Whitespace string between the opening brace and the command

class astropysics.publication.**Environment**(*parent*, *content*, *envname=None*)
Bases: astropysics.publication.TeXNode

A LaTex environment.

Subclassing Subclasses can implement the postParse() method - see the method for syntax. They should also be registered with the registerEnvironment() static method to have them be parsed with the default

TeXFile parser. Generally, they should also have a class attribute named *name* that gives the name of the environment (this name will be automatically used to determine which environments the subclass represents)

Parameters

- **parent** – The parent node
- **content** – The string between 'egin{...}' and 'end{...}'
- **envname** – If a string, it will be taken as the environment name. If None, it will be taken from the class-level name

static getEnvironments()
Returns a tuple of the registered environments.

getSelfText()

name = ''
The name of this environment.

postParse(*nodes*)

static registerEnvironment(*envclass*)
Registers the provided *envclass* in the environment registry. Also returns the class to allow use as a decorator.

Parameters **envclass** – The Environment object to be registered.

Raises

- **TypeError** – If the provided class is not a Environment subclass.
- **ValueError** – If the name attribute of *envclass* matches one already in the registry.

static unregisterEnvironment(*envclass*)
Removes the *envclass* Environment object from the registered environment list

Parameters **envclass** – The Environment object to be removed, or its associated name.

class astropysics.publication.**Figure**(*parent*, *content*, *envname=None*)
Bases: astropysics.publication.Environment

filenames
The names of the files (usually .eps) in this figure.

name = ['figure', 'figure*']

class astropysics.publication.**MathMode**(*parent*, *content*)
Bases: astropysics.publication.TeXNode

A math environment surrounded by $ symbols or $$ (for display mode)

Parameters

- **parent** – The parent node
- **content** – The full string (including $) or a tuple(displaymode,content) where *displaymode* is a bool and *content* is a string

displaymode = False
determines if the MathMode is in display mode ($$) or not ($)

getSelfText()

name = ''

class astropysics.publication.**Newline**(*parent*)
Bases: astropysics.publication.TeXt

A node that stores just a new line. This is always a leaf.

text = '\n'

class astropysics.publication.**OptionalArgument**(*parent*, *text*)
Bases: astropysics.publication.TeXNode

An argument to a macro that is required (i.e. enclosed in square brackets)

children = ()

getSelfText()

class astropysics.publication.**Preamble**(*parent*, *content*)
Bases: astropysics.publication.TeXNode

The preamble of a TeX File (i.e. everything before egin{document})

Parameters

- **parent** – The parent node
- **content** (*string*) – The text of the preamble

docclass
The document class of the tex file as a string

docclassopts
The document class options for the tex file as a comma-seperated string.

getSelfText()

class astropysics.publication.**RequiredArgument**(*parent*, *text*)
Bases: astropysics.publication.TeXNode

An argument to a macro that is required (i.e. enclosed in curly braces)

children = ()

getSelfText()

text = ''
The text of this argument object

class astropysics.publication.**TeXFile**(*fn=None*)
Bases: astropysics.publication.TeXNode

A TeX Document loaded from a file.

document = None
The first Document environment in this file or None if there isn't one

getSelfText()

preamble = None
The Preamble object for this file

save(*fn*)
Save the content of this object to a new file.

Parameters **fn** (*str*) – The name of the file to save.

class astropysics.publication.**TeXNode**(*parent*)
Bases: object

An element in the TeX parsing tree. The main shared characteristic is that calling the node will return a string with the combined text.

Subclassing

Subclasses must implement getSelfText() (see docstring for details)

children = ()
A list of child nodes of this node.

getSelfText()
Subclass implementations must return a 2-tuple of strings such that the child text goes in between the tuple elements. Alternatively, it can return a 3-tuple (before,between,after), and the resulting text will be "<beforetext><childtext1><between><childtext2>...<after>". It can also be None, in which case just the strings from the children will be returned. Otherwise, it can return a string, which will be returned as the full text.

isLeaf()
Returns True if this node is a leaf (has no children)

isRoot()
Returns True if this node is a root (has no parent)

parent = None
The parent of this node in the node tree, or None if this is a root.

prune(*prunechildren=True*)
Removes this node from the tree.

Parameters **prunechildren** – If True, all the children of this node will be pruned (recursively). This is not strictly necessary, but will speed up garbage collection and probably prevent memory leaks.

visit(*func*)
Visits all the nodes in the tree (depth-first) and calls the supplied function on those nodes.

Parameters **func** – The function to call on the nodes - should only accept the node as an argument.

Returns A sequence of the return values of the function. If the *func* returns None, it is not included in the returned list.

class astropysics.publication.**TeXt**(*parent*, *text*)
Bases: astropysics.publication.TeXNode

A node that stores generic text. This is always a leaf.

children

countWords(*sep=None*)
Returns the number of words in this object.

Parameters **sep** – The seperator between words. If None, use any whitespace.

Returns The number of words in this TeXt object.

getSelfText()

text = ''
The text in this object

class astropysics.publication.**TrailingCharCommand**(*parent*, *content*)
Bases: astropysics.publication.Command

A special command that allows a single trailing character of any type - used for 'left{' 'right]' and similar.

Parameters

- **parent** – The parent node
- **conent** – A (name,char) tuple, or a command string

children = ()

getSelfText()

astropysics.publication.**environment_factory**(*parent*, *texstr*)
This function takes a string from a TeX document starting with 'egin' and ending in 'end{...}' and uses it to construct the appropriate Environment object.

astropysics.publication.**prep_for_apj_pub**(*texfn*, *newdir='pubApJ'*, *overwritedir=False*, *figexts=('eps', 'pdf')*, *verbose=True*)
Takes a LaTeX file and prepares it for submission to The Astrophysical Journal. This involves the following actions:

1. Removes all text after end{document} from the .tex file
2. Removes all comments from .tex file.
3. Checks that the abstract is within the ApJ word limit and issues a warning if it is not.
4. Sets the document class to aastex.
5. Converts deluxetable* environments to deluxetable.
6. Removes epsscale{?} from all figures
7. Makes the directory for the files.
8. Renames the figures to the 'f1.eps','f2a.eps', etc. convention for ApJ, and copies the appropriate files over.
9. Copies .bib (or .bbl if no .bib) file if bibliography is present.
10. Saves the .tex file as "ms.tex"
11. Creates ms.tar.gz file containing the files and places it in the *newdir* directory.

Parameters

- **texfn** (*str*) – The filename of the .tex file to be submitted.
- **newdir** – The directory to save the publication files to.
- **overwritedir** – If True the directory specified by *newdir* will be overwritten if it is present. Otherwise, if *newdir* is present, the directory name will be `newdir_#` where # is the first number (starting from 2) that is not already present as a directory.
- **figexts** – A sequence of strings with the file name extensions that should be copied over for each figure, if present.
- **verbose** – If True, information will be printed when the each action is taken. Otherwise, only warnings will be issued when there is a problem.

Returns (file,dir) where *file* is the altered `TexFile` object and *dir* is the directory used for the publication materials.

astropysics.publication.**prep_for_arxiv_pub**(*texfn*, *newdir='pubArXiv'*, *overwritedir=False*, *figexts=('eps', 'pdf')*, *verbose=True*)

Takes a LaTeX file and prepares it for posting to arXiv. This includes the following actions:

1. Removes all text after end{document} from the .tex file
2. Removes all comments from .tex file.
3. Checks that the abstract is within the ArXiv line limit and issues a warning if it is not (will require abridging during submission).
4. Makes the directory for the files.
5. Copies over all necessary .eps and/or .pdf files.
6. Copies .bbl (or .bib if no .bbl) file if bibliography is present.
7. Creates the modified .tex file.
8. Creates a .tar.gz file containing the files and places it in the *newdir* directory.

Parameters

- **texfn** (*str*) – The filename of the .tex file to be submitted.
- **newdir** – The directory to save the publication files to.
- **overwritedir** – If True the directory specified by *newdir* will be overwritten if it is present. Otherwise, if *newdir* is present, the directory name will be `newdir_#` where # is the first number (starting from 2) that is not already present as a directory.
- **figexts** – A sequence of strings with the file name extensions that should be copied over for each figure, if present.
- **verbose** – If True, information will be printed when the each action is taken. Otherwise, only warnings will be issued when there is a problem.

Returns (file,dir) where *file* is the altered `TexFile` object and *dir* is the directory used for the publication materials.

astropysics.publication.**print_warnings = True**

Can be False to hide, True to print, or 'builtin' to use the python warnings mechanism

astropysics.publication.**text_to_nodes**(*parent*, *txt*)

Converts a string into a list of corresponding `TeXNode` objects.

Parameters

- **parent** – The parent node
- **txt** – The text to parse

Returns A list of `TeXNode` objects

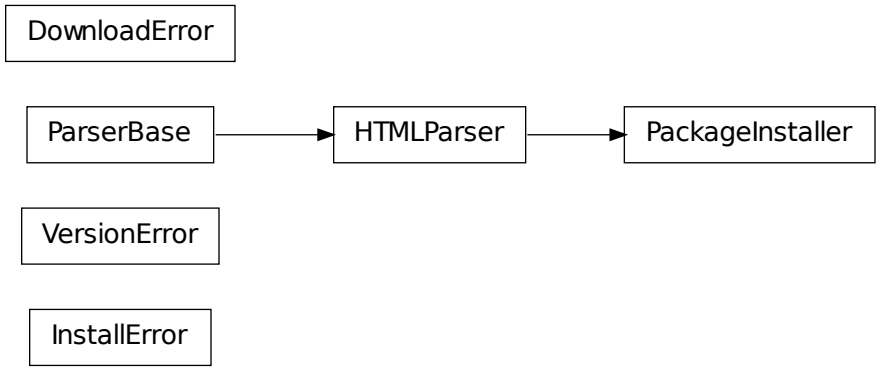
1.3.2 config – configuration and setup

The `config` module contains functions to manage and access the persistent astropysics configuration. It also includes utilities to install recommended packages and set up the ipython environment.

Configuration files can be found in the directory returned by `get_config_dir()`, typically a subdirectory '.astropysics' of the user's home directory. The format for the files is that of the ' configobj<http://www.voidspace.org.uk/python/configobj.html>'_ package, although for most files this is as simple as:

```
name1=value
#maybe a comment
name2 = anothervalue
```

Classes and Inheritance Structure



Module API

exception astropysics.config.**DownloadError**
Bases: exceptions.Exception

exception astropysics.config.**InstallError**
Bases: exceptions.Exception

class astropysics.config.**PackageInstaller**(*name*, *importmod=None*, *version=None*, *buildargs=''*, *instargs=''*, *extrainfo=None*, *verbose=True*)
Bases: HTMLParser.HTMLParser

Represents a python package to be downloaded and installed.

Parameters

- **name** – The name of the package.
- **importmod** – The module name to import to test if the pacakge is installed. If None, will be assumed to match *name*
- **version** (*str*) – A version request for finding the package on PyPI, such as '0.2','>0.1' (greater than 0.1), or '<0.3'. Can also be None to get the most recent version. If the PyPI entry only has a download link, this is ignored.
- **buildargs** – Arguments to be given to the "python setup.py build [args]" stage. Can be either a string or a sequence of strings.
- **instargs** – Arguments to be given to the "python setup.py install [args]" stage. Can be either a string or a sequence of strings.

- **extrainfo** (*str*) – A string with additional information about the pacakge (to be shown if the user requests it in the install tool). Can be None to indicate no extra info.
- **verbose** (*bool*) – If True, information will be printed to standard out about steps in the install process.

Subclassing

If a package needs some additional install steps, the `postInstall()` and `preInstall()` methods can be overridden (default does nothing). If the package isn't in PyPI, the `getURL()` method should be overridden to return the necessary URL.

download(*dldir=None*, *overwrite=False*)
Attempt to download this package

Parameters

- **dldir** (*str*) – The directory to download to. If None, the standard configuration directory will be used.
- **overwrite** (*bool*) – If True, downloaded package archives will be overwritten instead of being re-used.

Returns The full path to the downloaded file.

Raises DownloadError If the pacakge could not be located

getUrl()
Returns the URL to download to get this package. Override this to get a URL from somewhere other than PyPI.

Returns (url,fn) where *url* is the URL to the source distribution, and *fn* is the filename to use locally if None, the end of the URL will be used)

handle_starttag(*tag*, *attrs*)

install(*dldir=None*, *overwrite=False*, *instprefix=''*)
Download the package, if necessary, and install it.

Parameters

- **dldir** (*str*) – The directory to download to. If None, the standard configuration directory will be used.
- **overwrite** (*bool*) – If True, downloaded package archives will be overwritten instead of being re-used.
- **instprefix** (*str*) – A command line prefix (before "python") to be used in the install step. Most often this is 'sudo' on certain oses.

Raises InstallError If the install fails (`postInstall()` will be called immediately before).

isInstalled()
Test if the package is installed.

Returns True if the module is installed, otherwise False.

postInstall(*idir*, *success*)
Subclasses can override this method to do something after building and installing occurs. Only called if install succeeds.

Parameters

- **idir** (*str*) – The path to the directory in which the package is built.
- **success** (*bool*) – If True, the install was sucessful. Otherwise, it failed.

preInstall(*idir*)
Subclasses can override this method to do something before building and installing occurs.

Parameters **idir** (*str*) – The path to the directory in which the package is built.

exception astropysics.config.**VersionError**
Bases: exceptions.Exception

astropysics.config.**add_project**(*name*, *dir=None*, *scriptfile=None*)
Add a new project to the project registry.

Parameters

- **name** (*str*) – The name of the project.
- **dir** (*str*) – The name of the directory associated with the project or None to use *name*, relative to the current directory. If the directory dos not exist, it will be created.
- **scriptfile** (*str*) – The name of the file with the main runnable python code for the project, relative to the *dir* directory, or None to use the name. If the script does not exist, it will be created with an analysis template file (see project_template.py).

Returns A 2-tuple (projectdir,projectscriptfilename)

Raises IOError If something is wrong with the file or directory.

astropysics.config.**del_project**(*name*)
Unregisters the project with the specified name.

astropysics.config.**get_config**(*name*)
Returns a dictionary-like object that has the configuration information for the specified configuration file. To save the configuration data if it is modified, call write() on the object.

Parameters **name** (*str*) – The name of the configuration file (without any path). The file will be searched for in/written to the config directory.

Returns A ConfigObj object with the configuration information.

Raises

- **ValueError** – If the *name* is invalid.
- **astropysics.external.configobj.ConfigObjError** – If the file exists and it is an invalid format.

astropysics.config.**get_config_dir**(*create=True*)
Returns the astropysics configuration directory name.

Parameters **create** (*bool*) – If True, the directory will be created if it doesn't exist.

Returns The absolute path to the config directory as a string

astropysics.config.**get_data_dir**(*create=True*)
Returns the directory name for data that astropysics downloads. See :func‘astropysics.utils.io.get_data‘ to work with data in this directory.

Parameters **create** (*bool*) – If True, the directory will be created if it doesn't exist.

Returns The absolute path to the data directory as a string.

astropysics.config.**get_projects**()
Returns all registered projects and their assoiciated directories and script files.

Returns A dictionary where the keys are the project names and the values are 2-tuples (projectdir,projectscriptfilename)

astropysics.config.**run_install_tool**(*sudo='auto'*)
Starts the console-based interactive installation tool.

Parameters **sudo** – Determines whether or not the install step is prefixed by 'sudo'. If True, sudo will be prefixed, if False, no prefix will be used. If 'auto', the platform will be examined to try to determine if sudo is necessary. If 'toggleauto', the opposite of whatever 'auto' gives will be used.

Note: The 'auto' option for *sudo* is nowhere close to perfect - it's pretty much just a list of common platforms that require it... if you're on an uncommon platform, you will probably have to set it manually.

---

astropysics.config.**run_ipython_setup**()
Runs the console-based ipython setup tool.

1.4 GUI applications

Astropysics includes a variety of graphical applications for various data analysis tasks. Their full power is acheived when used interactively or as part of scripts, but some operate as stand-alone command-line tools when the situation warrants.

Note that these applications make heavy use of Enthought Traits and the associated Enthought Tools Suite if they are not installed, most of these will not function.

Another important related GUI used in astropysics is the Fit GUI from `pymodelfit`. This GUI is used wherever interactive curve-fitting is needed (and was, in fact, originally made for astropysics).

1.4.1 Spylot – Spectrum Plotter

This application is a spectrum plotting/interactive analysis tool based on Traits. It is essentially a GUI wrapper and interface around a collection of `astropysics.spec.Spectrum` objects.

It can be run as part of a python script or interactively in ipython by generating `astropysics.gui.spylot.Spylot` objects or calling the `astropysics.gui.spylot.spylot_specs()` function as detailed below. A command-line script 'spylot' is also installed when you install astropysics. The command-line tool takes the name of the spectrum file to display and the following options:

-h, --help show help message and exit

-t TYPE, --type=TYPE file format of spectra: "wcs", "deimos", or "astropysics"(default)

-e EXT, --extension=EXT Fits extension number

-c CONFIGFILE, --config=CONFIGFILE File to save Spylot configuration to

---

Todo

Write a Tutorial/examples for both code and command-line script

---

1.4.2 Spectarget – MOS Targeting

This application is a tool based on Traits for interactively identifying spectroscopic targets for use with a multi-object spectograph. Note that this module is a bit rough around the edges and hasn't been too tested with anything other than

Keck/DEIMOS.

the `gui` module also imports the convinience functions and primary application classes for these GUIs. These include:

- `fit_data` (imported from `pymodelfit`)
- `Spylot`
- `spylot_specs`
- `SpecTarget`
- `spec_target`

1.5 Developer Guidelines for Astropysics

Astropysics welcomes contributions from anyone interested, from simple bugfixes to contributing new functionality. If you want to check out the most recent version to contribute (or just want the latest and greatest), you can pull the current development version from the google code page.

1.5.1 Getting the Code

You will need to have mercurial installed. Once you do, simply execute:

```
hg clone https://astropysics.googlecode.com/hg/ astropysics-dev
```

This will create a directory with the name `astropysics-dev` containing the latest and greatest version of astropysics. If at any time you want to update this version, go into the directory and do:

```
hg pull
hg update
```

then re-install following the directions above. If you plan on editing the astropysics source code (please do so, and submit patches/new features!), a useful way to immediately see changes without having to re-install every time is to use the command:

```
python setup.py develop
```

possibly prefixed with `sudo` depending on your OS. This will install links to the source code directory instead of copying over the source code, so any changes you make to a module can be seen just be doing `reload(module)`.

1.5.2 Cloning the Repository to Submit Code

If you intend to regularly contribute changes or patches to astropysics, a more convinient way to submit changes is with a public clone of the main astropysics repository. Go to the source tab of the google code project, and click on the `create clone` button. Fill in the necessary information, and clone *your* repository to your computer instead of the main astropysics repository. Make your changes, using `hg commit -m "a message"` to describe changes as you make them. You can then use `hg push` to send changes back to your repository on google code, and those can easily be merged with the main repository with a pull request.

1.5.3 Coding Style Guidelines

Naming Conventions

For general coding style, PEP8 provides the main coding style for astropysics, with an important exception: PEP8 calls for methods (i.e. functions that are in the scope of a class) to use the `lower_case_with_underscores` convention. Astropysics instead uses `camelCase` (e.g. first letter of each word upper case, first letter of the method lower case) for method names. Functions that are not methods (i.e. directly in a module scope) remain `lowercase_with_underscores`. This allows methods and functions to be easily distinguished (but keeps method names distinct from class names, for which the first letter is always upper case). This could change in the future to be fully PEP8 compliant, but for now, given the already existing codebase, `camelCase` it is.

To summarize, the naming conventions are:

- Module names are always `lowercase`.
- Class names are always `CamelCaseFirstLetterUppercase`.
- Methods (including static methods and class methods) are always `camelCaseFirstLetterLowercase`.
- Functions are always `lowercase_with_underscore`.
- Variables and class attributes should be `lowercase` and kept to a single word if possible.
- Private/internal functions, methods, or variables should be `_leadingUnderscore` with the appropriate convention for the type. Python and sphinx both know to hide these unless you specifically request them. Python also supports `__doubleLeadingUnderscore` for private class methods (the double-underscore is mangled), but this generally just leads to confusion if you're not careful, so it should be avoided unless there's some very good reason.

Documentation Standards

Documentation should be provided with every object, using sphinx REStructuredText markup. Functions and methods should use info field lists To specify input parameters, return values, and exceptions (where relevant). Below is an example of the standard format:

```
def a_function(arg1,arg2=None,flag=False):
    """
    A short description of the function.

    Multiple paragraphs if necessary, i.e. background is needed.

    :param arg1:
        This argument is important input data, although I'm not specifying
        exactly what it's type is (maybe it'd duck-typed?)  Also, the
        description is more than one line, so it has to start underneath
        and indented.
    :param arg2: This argument is an optional input.
    :type arg2: You can specify a type here if you want.
    :param bool flag: You can also give the type in param if it fits.

    :except ValueError:
        If you raise an exception, specify here what type it is and why.

    :returns: A description of the return value, if there is one.

    **Examples**

    If an examples are needed, they should go here, ideally in doctest
```

```
    format so that they can be used as tests:

    >>> inpt = 'something for a_function'
    >>> a_function(inpt,flag=True)
    'whatever a_function should output'

    """
```

Classes with public attributes can document using the sphinx construct for documenting class attributes:

```
class MyClass(object):

    #: Documentation for :attr:`value` attribute.
    value = None

    def __init__(self,value):
        self.value = value
```

1.5.4 Testing Astropysics

There is a test suite that should be periodically run to ensure everything that has tests is still working correctly. It requires nose. It can be run from the astropysics source directory (where setup.cfg lives) with the command:

```
nosetests
```

Note that this is also set up to easily debug in the event that some of the tests fail. Simply do:

```
nosetest --failed
```

And nose will only run those tests that failed the last time around. If you want to run a particular test, do:

```
nostest --with-id 3
```

Where the '3' can be replaced by whatever number test you want.

When writing functionality in astropysics, it's a good idea to add tests. These should go in the 'tests' directory, and should have module names with the word 'test' in them, along with the function names themselves. This naming is necessary to allow nose to find all the tests. Alternatively, snippets of code as they would appear on the python interpreter (*with* output) can be placed directly in the docstrings, and they will be automatically included in the tests.

CHAPTER 2

Quick Install

See *Installing Astropysics* for full install instructions, including prerequisite packages.

To install a current release of astropysics, the simplest approach is:

```
pip install astropysics
```

(on unix-like systems or OS X, add "sudo " before this command)

If you want the must up-to-date (possible unstable) version, do:

```
hg clone https://astropysics.googlecode.com/hg/ astropysics-dev
cd astropysics-dev
python setup.py develop
```

(note that mercurial must be installed, and on some systems the last command may need to have "sudo " at the beginning)

You can also alter the source code if you use this approach (see *Developer Guidelines for Astropysics* for guidelines of working contributing source code).

In either case, afterwords you can run:

```
astpys-setup
```

to install optional packages and setup the environment.

CHAPTER 3

Bug Reports

The best place to report bugs is via the google code bug tracker. That way they won't be forgotten unless an asteroid impact destroys all of google's servers.

CHAPTER 4

Logo Image Credit

The multiwavelength image of M81 was put together by the folks at the Chandra X-Ray Observatory (http://chandra.harvard.edu/photo/2008/m81/), and they credit: "X-ray: NASA/CXC/Wisconsin/D.Pooley & CfA/A.Zezas; Optical: NASA/ESA/CfA/A.Zezas; UV: NASA/JPL-Caltech/CfA/J.Huchra et al.; IR: NASA/JPL-Caltech/CfA". The Python logo can be found at http://www.python.org/community/logos/.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

a