channelpack Documentation

Release 0.7.0

Tomas Nordin

Mar 06, 2021

Contents

A small Python library providing an object to hold a number of Numpy arrays. The object is callable like a function and calls are made to get at data.

A few factory functions to get a ChannelPack object from data files are also provided.

Site contents:

CHAPTER 1

Overview and examples

The ChannelPack class is a basic wrapper class for a dict of data and a dict of field names. Those dict attributes, *data* and *names*, are a little special – they both require integer keys and the *data* dict will convert sequence values to Numpy arrays if not arrays already. And the *data* dict will raise an exception if any resulting array is not 1-dimensional.

The 1-dimensional requirement reflects a view of the ChannelPack object as a holder of flat file data columns.

The integer keys in respective dict are supposed to align to be able to refer to arrays by name.

ChannelPack objects are callable (like functions) and the idea is to get at data by making calls to the object, like *pack(ch)*, where *ch* is the key for data, either a string name or an integer key.

1.1 Make an object

ChannelPack takes zero or one dict for data and zero or one dict for names to initialize. *data* and *names* can also be assigned after initialization.

1.1.1 Produce some data and make a pack

```
>>> import channelpack as cp
>>> pack = cp.ChannelPack()
>>> pack.data = {0: range(5), 1: ('A', 'B', 'C', 'D', 'E')}
>>> pack.names = {0: 'seq', 1: 'abc'}
>>> pack
ChannelPack(
data={0: array([0, 1, 2, 3, 4]),
      1: array(['A', 'B', 'C', 'D', 'E'], dtype='<U1')},
names={0: 'seq',
       1: 'abc'})
>>> # make calls to object to get at data
>>> pack(0)
array([0, 1, 2, 3, 4])
```

(continues on next page)

(continued from previous page)

```
>>> pack(0) is pack('seq')
True
```

The pack is meant to be called to get at data, (*__call__()*), but it is not against the law to operate on the the *data* and *names* attributes directly:

```
>>> pack.data[2] = [letter.lower() for letter in pack('abc')]
>>> pack.names[2] = 'abclower'
>>> pack
ChannelPack(
data={0: array([0, 1, 2, 3, 4]),
      1: array(['A', 'B', 'C', 'D', 'E'], dtype='<U1'),
      2: array(['a', 'b', 'c', 'd', 'e'], dtype='<U1')},
names={0: 'seq',
       1: 'abc',
       2: 'abclower'})
```

1.2 Slicing out parts of data

Support for slicing and filtering is provided by a Boolean array *mask* in the pack and the *parts* or *nof* arguments in calls. In calls to get at data, the mask is consulted to return parts of the data with corresponding True parts in the mask, depending on arguments. A True entry in the mask represents *valid* data.

1.2.1 The mask attribute

The mask in the pack is set by performing comparisons on arrays, possibly combined with Numpy bitwise operators like *&* and | (bitwise *AND* and *OR*). The goal is to set the mask to a Boolean array of the same size as the data arrays:

```
>>> pack.mask = (pack('seq') < 2) | (pack('abc') == 'D')
>>> pack('seq', part=0)
array([0, 1])
>>> pack('seq', part=1)
array([3])
>>> pack('abc', nof='filter')
array(['A', 'B', 'D'], dtype='<U1')
>>> pack('abc', nof='nan')
array(['A', 'B', None, 'D', None], dtype=object)
>>> pack('seq', nof='nan')
array([ 0.,  1., nan,  3., nan])
```

- The *part* argument refer to a contiguous True part of the mask, enumerated from 0. With all elements or only one part True in the mask there is one part == 0. This argument overrides the *nof* argument.
- With *nof='filter'*, a possibly shorter version of data is returned depending on the mask.
- With *nof='nan'*, the data length is the same as original array but with corresponding non-true elements in mask replaced with np.nan or None depending on the type.

See also:

mask_reset()

1.2.2 Start, stop and duration

Sometimes it's easier to think of a part as starting at some event or condition and stopping at some other. A method *startstop()* is supporting something like a "start and stop trigger".

Imagine some alternating movement over time that is slowing down:

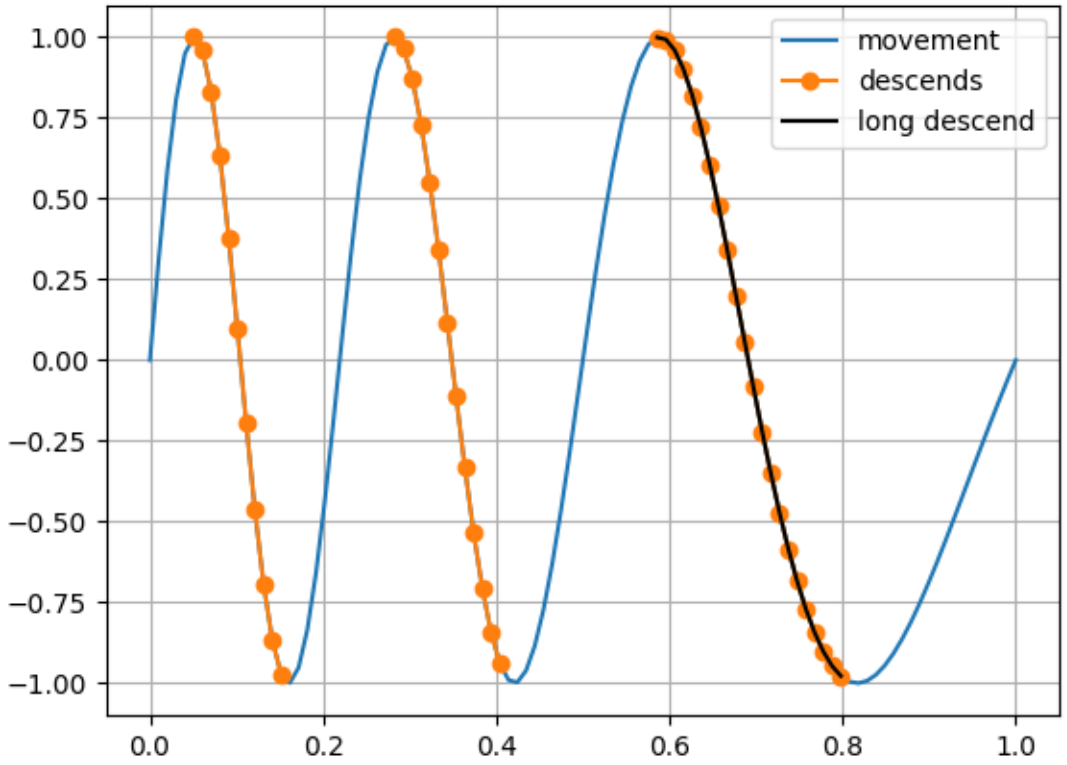
```
>>> import numpy as np
>>> import matplotlib.pyplot as pp
>>> t = np.linspace(0, 1, 100)  # (samplerate 100)
>>> f = 5.0
>>> movement = np.sin(2 * np.pi * (f - 2 * t) * t)
>>> pack = cp.ChannelPack({0: t, 1: movement}, {0: 'time', 1: 'movement'})
>>> # Plot the whole movement
>>> _ = pp.plot(pack('time'), pack('movement'), label='movement');
```

Say that the descending slopes are of particular interest:

```
>>> startb = pack('movement') > 0.98
>>> stopb = pack('movement') < -0.98
>>> _ = pack.startstop(startb, stopb)
>>> # plot only the descends
>>> _ = pp.plot(pack('time'), pack('movement', nof='nan'),
...             label='descends', marker='o')
```

A method *duration()* can be used to make false any true parts that is not long enough. Filter out the shorter slopes:

```
>>> _ = pack.duration(0.15, samplerate=100)
>>> # plot only the remaining descend
>>> _ = pp.plot(pack('time'), pack('movement', nof='nan'),
...             label='long descend', color='black')
>>> # show it
>>> pp.grid()
>>> _ = pp.legend(loc='upper right'); pp.show()
```



1.3 Factory functions to get a pack

A few factory functions are provided to create a pack from data files.

1.3.1 Text

Data stored in readable text files in the form of delimited data fields, (csv, txt). Fields might be numbers or text.

textpack()

If data is numeric only, a lazy variant is available

lazy_textpack()

1.3.2 Spread sheet

Code from the library *xlrd* is used, xls and xlsx types of spread sheets are supported.

sheetpack()

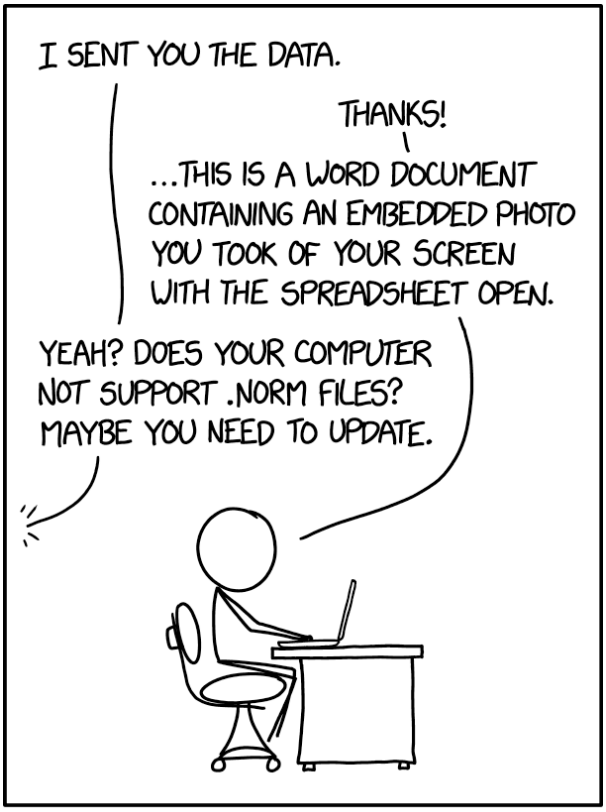
1.3.3 Xbase DBF format

Legacy kind of data base format.

dbfpack()

1.3.4 Normal File Format (.NORM)

Currently not supported.



CHAPTER 2

channelpack API Reference

All objects and functions documented below are available by:

```
import channelpack
```

in the *channelpack* namespace.

2.1 ChannelPack object

class channelpack.**ChannelPack**(*data=None*, *names=None*)

Callable collection of data.

Hold a dict of data (numpy 1d arrays) and make possible to refer to them by calls of this object, (*pack(ch)*). A boolean mask is kept with the pack, used to optionally filter out sections of data in calls.

data

The dict is not supposed to be consulted directly, call the ChannelPack object to refer to arrays. Keys are integers representing column numbers. Setting this attribute to a new dict of data will convert values to numpy arrays and call mask_reset() automatically.

Type dict

mask

A boolean array of the same size as the data arrays. Initially all True.

Type numpy.ndarray

nof

'nan', 'filter' or None. In calls to the object, this attribute is consulted to determine how to return data arrays. If None, arrays are returned as is (the default). If 'nan', elements in the returned array with corresponding False element in *mask* are replaced with numpy.nan or None, equivalent to *np.where(array, mask, np.full(len(array), np.nan))*. 'filter' yeilds the equivalent to *array[mask]* – the array is stripped down to elements with corresponding True elements in *mask*. The effect of this attribute can be overridden in calls of the object.

Type str or None

names
Keys are integers representing column numbers (like in *data*), values are strings, the field names. Keys in *names* aligned with keys in *data* makes it possible to refer to arrays by field names. This alignment is not enforced.

Type dict

mindur
Like the method *duration* (which see) but with a persistent effect. Any time the mask is updated, this attribute is consulted to falsify any true part in the mask that is not long enough. The value refer to the required number of elements in a true section.

Setting this attribute to a value (not None) updates the mask without first resetting it.

Type int or None

FALLBACK_PREFIX
Defaults to 'ch'. This can be used in calls of the pack in place of a "proper" name. If 4 is a key in the data dict, pack('ch4') can be used to get at that data. This is also used as requested in calls to the *records* method. Everything after this prefix is assumed to be a number. The prefix should be a valid python variable name.

Type str

fn
File name of a possible source data file. After initialization it is up to the caller to set this attribute, else it is the empty string.

Type str

filenames
Maintained by the pack when setting *fn*. Extended with *other.filenames* in calls to *append_pack(other)*. A list of one or more empty strings if *fn* is not set.

Type list of str

__init__(*data=None*, *names=None*)
Initiate a ChannelPack

Convert given sequences in *data* to numpy arrays if necessary.

Parameters

- **data** (*dict*) – Keys are integers representing column numbers, values are sequences representing column data.
- **names** (*dict*) – Keys are integers representing column numbers (like in data), values are strings, the field names.

__call__(*ch*, *part=None*, *nof=None*)
Return data from "channel" ch.

If *part* is not given, return the array for *ch* respecting the setting of attribute *nof*. See the class attributes description in *ChannelPack* for the meaning of *nof*.

Parameters

- **ch** (*str or int*) – The channel key, name or fallback string. The lookup order is keys in the data dict, names in the names dict and finally if *ch* matches a fallback string.
- **part** (*int*) – The 0-based enumeration of a True part to return. Overrides the effect of attribute or argument *nof*.

- **nof** (*str*) – One of 'nan', 'filter' or 'ignore'. Providing this argument overrides any setting of the corresponding attribute *nof*, and have the same effect on the returned data as the attribute *nof*. The value 'ignore' can be used to get the full array despite a setting of the attribute *nof*.

append_pack(*other*)
Append data from other into this pack.

If this pack has data (attribute data is non-empty), it has to have the same set of keys as other.data (if that is non-empty). Same is true for the attribute names.

Array dtypes in respective pack.data are at the mercy of numpy append function.

Extend *filenames* with *other.filenames*.

mask_reset is called after the append.

Parameters **other** (*ChannelPack instance*) – The other pack.

Raises ValueError – If non-empty dicts in packs do not align.

mask_reset()
Set the mask attribute to the length of data and all True.

If this pack's data dict is empty, set mask to an empty array. Size of the mask is based on the array with the lowest key in data.

duration(*duration*, *samplerate=1*, *mindur=True*)
Require each true part to be at least duration long.

Make false any true part in the mask attribute that is not at least *duration* long.

Parameters

- **duration** (*int or float*) –
- **samplerate** (*int or float*) – If samplerate is 10 and duration is 1, a True part of minimum 10 elements is required.
- **mindur** (*bool*) – If False, require parts to be at most duration long instead.

Returns The possibly altered mask.

Return type array

startstop(*startb*, *stopb*, *apply=True*)
Start and stop trigger masking.

Elements in startb and stopb are start and stop triggers for masking. A true stop dominates a true start.

Parameters

- **startb** (*sequence*) –
- **stopb** (*sequence*) – Elements are tested with *if el...*
- **apply** (*bool*) – If True, apply the result of this method to the mask attribute by anding it, (mask &= result).

Returns A bool ndarray, the result of this method.

Return type array

Example

One descend:

```
height: 1 2 3 4 5 4 3 2 1
startb: F F F F T F F F F (height == 5)
stobb:  T F F F F F F F T (height == 1)
result: F F F F T T T T F
-> height:      5 4 3 2
```

parts()
Return the enumeration of the True parts.

The list is always consecutive or empty. Each index in the returned list can be used to refer to a True part in the mask attribute.

records(*part=None*, *nof=None*, *fallback=False*)
Return a generator producing records of the pack.

Each record is provided as a collections.namedtuple with the packs names as field names. This is useful if each record make a meaningful data set on its own.

Parameters

- **part** (*int*) – The 0-based enumeration of a True part to return. Overrides the effect of attribute or argument *nof*.
- **nof** (*str*) – One of 'nan', 'filter' or 'ignore'. Providing this argument overrides any setting of the corresponding attribute *nof*, and have the same effect on the returned data as the attribute *nof*. The value 'ignore' can be used to get all the records despite a setting of the attribute *nof*.
- **fallback** (*bool*) – The named tuple requires python-valid naming. If fallback is False, ValueError is raised if any of the names in *names* is an invalid identifier. fallback=True will use FALLBACK_PREFIX to produce names.

Raises ValueError – In iteration of the generator if any of the names used for the namedtuple is invalid python identifiers.

Note: Either there must be names defined in the pack or argument *fallback* must be True, else there will be no records.

---

name(*ch*, *firstwordonly=False*, *fallback=False*)
Return a name string for channel *ch* in names.

A helper method to get a name string, possibly modified according to arguments. Succeeds only if *ch* corresponds to a key in data.

Parameters

- **ch** (*int or str.*) – The channel key or name. An integer key has precedence.
- **firstwordonly** (*bool or str*) – If True, return only the first space-stripped word in the name. If a string, use as a regex pattern with re.findall on the name string and return the first element found.
- **fallback** (*bool*) – If True, return the fallback string <FALLBACK_PREFIX><N>, where N corresponds to the data key. Ignore the firstwordonly argument.

2.2 Functions to get a pack from data files

2.2.1 Text

Data stored in readable text files in the form of delimited data fields, (csv, txt). Fields might be numbers or text:

channelpack.**textpack**(*fname*, *names=None*, *delimiter=None*, *skiprows=0*, *usecols=None*, *hasnames=False*, *encoding=None*, *converters=None*, *stripstrings=False*, *debug=False*)

Make a ChannelPack from delimited text data.

First line of data is the line following skiprows.

First line of data determines what fields (splitted by delimiter) can be converted to a float. Fields that can't be converted to float will be treated as strings. Converters in converters are used if given.

Numeric fields with decimal comma are understood as numeric (besides numerics with decimal point). If delimiter is a comma it is therefore important to specify that.

Parameters

- **fname** (*str, file or io stream*) –
- **names** (*dict*) – Keys are integers (0-based column numbers) and values are field names. If provided it will be set in the pack and is mutually exclusive with the usecols argument.
- **delimiter** (*str or bytes*) – If not given, any white space is assumed. If fname is a stream of bytes, delimiter must be bytes if not None.
- **skiprows** (*int*) – The number of lines to ignore in the top of fname. First line following skiprows is data.
- **usecols** (*sequence or int*) – The columns to read. A single integer means read that one column. Ignore if names is given.
- **hasnames** (*bool*) – If True, the last line of skiprows is assumed to be field names and will be used to set names in the pack. Ignored if names is given.
- **encoding** (*str*) – Use encoding to open fname. If None, use default encoding with io.open. Valid when fname is as string. If fname is a stream of bytes and encoding is given, use encoding to decode bytes in text fields.
- **converters** (*dict*) – A mapping of column numbers and functions. Each function take one string argument and return a value.
- **stripstrings** (*bool*) – For string fields, strip off leading and trailing whitespace resulting from whitespace around the delimiter.
- **debug** (*bool*) – If true, output the functions used on fields and the last successful line number read, before an exception is raised.

If data is numeric only, a lazy variant is available:

channelpack.**lazy_textpack**(*fname*, *parselines=25*, ***textkwargs*)

Return a ChannelPack instance using textpack function.

Try to automatically derive values for the textpack keyword arguments 'delimiter', 'skiprows' and 'converters'. Also try to parse out the field names.

Works with numerical data files, which might have a header with extra information to ignore. Converters derived is either float or one that converts numbers with decimal comma to a float.

Keyword arguments provided to this function overrides any derived equivalents.

Parameters

- **fname** (*file, str*) – Encoding given in textkwargs is respected.
- **parselines** (*int*) – The number of lines to preparse. For a successful preparse it must include at least one line of numeric data.
- ****textkwargs** – Other keyword arguments accepted by textpack. Overrides derived keyword arguments if duplicated.

2.2.2 Spread sheet

Code from the library *xlrd* is used, xls and xlsx types of spread sheets are supported:

channelpack.**sheetpack**(*fname*, *sheet=0*, *header=True*, *startcell=None*, *stopcell=None*, *usecols=None*)

Return a ChannelPack instance loaded from spread sheet file.

Parameters

- **fname** (*str*) – The file name to read from.
- **sheet** (*int or str*) – Sheet enumeration or name string.
- **header** (*bool or str*) – True means the data range include field names (top record). False means the whole range is data. A string can be used to specify the startcell of the header row, like "C1".
- **startcell** (*str*) – Spread sheet style notation of the upper left cell of the data range, like "C3".
- **stopcell** (*str*) – Spread sheet style notation of the lower right cell of the data range, like "H10".
- **usecols** (*str or seqence of ints*) – The columns to use, 0-based. 0 is the spread sheet column "A". Can be given as a string also - 'C:E, H' for columns C, D, E and H.

About code from the xlrd project

channelpack include code from the xlrd project copied from a checkout of commit d470bc9374ee3a1cf149c2bab0684e63c1dcc575 and is thereby not dependent on the xlrd project.

With the release of version 2.0.0 of xlrd, support for the xlsx format was removed. A main reason it seems was nobody was willing to maintain it (the xlrd project do not discourage using xlrd for xls files). Concerns about possible vulnerabilities with the xml parsing was also raised and since channelpack now include the code that was removed from xlrd, some sort of re-iteration of those concerns is given here so a potential user of channelpack can make an informed choice.

The announcement about xlrd 2.x series and the deprecation of xlsx support can be read here

https://groups.google.com/g/python-excel/c/IRa8IWq_4zk/m/Af8-hrRnAgAJ

One issue alleged was that defusedxml and xlrd as a combination don't work well with python 3.9. The linked defusedxml project readme discuss the vulnerabilities with xml files it addresses. Those vulnerabilities are also discussed in the Python docs here and in a thread on the python bug tracker, "XML vulnerabilities in Python", discussing if it should be addressed by Python xml libraries.

In short, it is possible to craft xml files so they might cause harm or disturbance when parsing them with a parser not taking precautions for the risk. The code from the xlrd project included in channelpack uses defusedxml if available.

Early xlrd includes software developed by David Giffin <david@giffin.org>.

2.2.3 Xbase DBF format

Legacy kind of data base format:

channelpack.**dbfpack**(*dbf*, *names=None*)
Make a ChannelPack from dbf data file.

Parameters

- **dbf** (*str or file*) – If a file it should be opened for binary reads.
- **names** (*list of str*) – A sequence of names to read. If not provided read all.

CHAPTER 3

Changes

0.7.0 (2021-03-06)

- Inclusion of files from the xlrd project from a checkout before version 2.0.0 to continue support for xlsx files. Not dependent on xlrd project any more.
- A 'mindur' attribute in the ChannelPack object with the same effect as the 'duration' method but with a persistent effect, applied automatically when the mask is renewed.
- Documentation updates.
- Note about the xlrd code under the Spread sheet section in API reference docs.

0.6.2 (2020-10-10)

- Bugfix of function _slicelist() in pack.py:Channelpack being called every time the pack was called on a part. A chached slicelist is used now instead and updated only when the mask is set anew.
- A Makefile fix in linter rule.

0.6.1 (2020-06-26)

- Better handling of missing values in text files.
- Bugfix to respect encoding using the contextopen function in readtext, d1e26a.
- readtext.py: Don't blindly strip all white space from lines (made converters argument have no effect).

0.6.0 (2020-06-10)

- The project rewritten completely.
- Python 3 and 2 supported.
- History prior to this release might be erased.

0.4.0 (2017-08-19)

- Allow open file objects to be supplied to the `txtpack` function, not only the file name as a string.
- Some pep8 work done in the core python files.

0.3.2 (2016-10-29)

- Bugfix in pullxl.py. An update in xlrd version 1.0.0 on empty cell values made this bug evident. Empty cells was reported as empty bytestrings before, now it is reported as empty unicode strings as documented. The channelpack bug resulted in all values in a "channel" being unicode strings when some should be numpy.nan and the rest numbers.

0.3.1 (2015-05-10)

- Added a records method to ChannelPack.

0.3.0 (2015-04-06)

- The persistent condition strings are written whith python syntax operating on numpy arrays. Identifiers use replacement syntax like `%(<id>)`.
- Removed method `ch` from the ChannelPack, superfluous.
- Added `counter` method to the ChannelPack.
- Added `parts` method to the ChannelPack.
- Change name on method set_samplerate, was set_sample_rate.
- A number of bugfixes.
- Two xldate conversion functions in pullxl.

0.2.2 (2014-11-04)

- Important bugfix in pulldbf module. Was forcing types on numpy, biggest float was 'f4'. In case of excel kind of dates for example, this is not enough, and numbers were lost. The result was repeating equal numbers that should not be equal.
- Since forcing types on numpy was deprecated, an issue with missing values in a dbf file (nulls) was easy to fix. Now, null values are replaced with numpy nan, was (0). Also very important.

0.2.1 (2014-10-20)

- xlrd added to install_requires list in setup.py
- Docstring fix in slicelist method. Was wrong.

0.2.0 (2014-10-19)

- Support for reading spreadsheet data (xlrd as backend).
- Add method slicelist to channelpack.

0.1.5 (2014-10-15)

- Bugfix in function rebase in the ChannelPack class. Was a KeyError.

0.1.4 (2014-09-22)

- Corrections in the changes file, (this file). Some changes was stated as coming but already implemented, (assumingly with 0.1.2).
- Add some coming changes in this file.
- This version has no changes in code and no distribution. But docs are updated.

0.1.3 (2014-09-21)

- Docs made available. Readme updated.
- Update docstring in ChannelPack.load
- Comments in the pulltxt module on possible alternative regex.

0.1.2 (2014-09-21)

- Bugfix (hopefully) with the start and stop conditions. Addition in function _startstop_bool in datautils module. Start conditions could be "ignored".
- Update append_load method in pack.ChannelPack. Keep data on all files loaded in metamulti dict, including the new file.
- Add method rebase in ChannelPack, rebase and align a channel.

0.1.1 (2014-08-16)

- Editorials on the README file. Some updates in the setup.py file.

0.1.0 (2014-08-16)

- Initial release

CHAPTER 4

Install

To install channelpack, do the usual pip tango:

```
$ pip install channelpack
```

A numpy (https://numpy.org/) installation is assumed and not installed by channelpack.

CHAPTER 5

Getting the source code

Get the source for the package:

```
$ git clone https://github.com/tomnor/channelpack.git
```

CHAPTER 6

License

GPL

Index

Symbols

A

C

D

F

L

M

N

P

R

S

T