pynrrd Documentation

Release 0.4.2

Maarten Everts

Apr 01, 2021

Contents:

CHAPTER 1

User Guide

1.1 About the NRRD file format

NRRD stands for **N**early **R**aw **R**aster **D**ata and is a file format designed for scientific visualization and image processing involving N-dimensional data. It is a simple and flexible file format with a header containing information about the data that can be read by simply opening the NRRD file in a text editor. Many other file formats such as PNG, JPG, DICOM cannot be read with a simple text editor! In addition, the raw data can be stored in a NRRD file in a number of commonly known formats such as ASCII, gzip, bzip or even raw. The header information and data itself can be stored in the same file or separately. Another feature of the NRRD file format is the support of custom key/value pairs in the header to allow storing of user information not defined in the NRRD specification.

The general structure of the NRRD file format is as follows:

```
NRRD<NRRD file version>
# Complete NRRD file format specification at:
# http://teem.sourceforge.net/nrrd/format.html
<field>: <value>
<field>: <value>
...
<field>: <value>
<custom field>:=<value>
<custom field>:=<value>
...
<custom field>:=<value>
# Comments are identified with a # symbol

<data here if stored in same file, otherwise stored in detached file>
```

As with most file formats, the file begins with a magic line that uniquely identifies the file as a NRRD file. Along with the magic line is a version that identifies what revision of the NRRD specification is used. As of today, the latest NRRD version is 5. Line comments can be specified by starting the line with a # symbol. Following the magic line is the header which is written in ASCII with the structure `<field>: <value>` or `<field>:=<value>` in the case of a custom field. The header is finished reading once a blank line has been read. If the data is stored in the same file (based on whether the header contains a datafile field), then the data will follow the header.

More information on the NRRD file format can be found here.

1.2 Header datatypes

There are 10 possible datatypes that a value can have in the header. Below are the valid datatypes along with the datatype they are parsed into in Python and examples of the datatypes.

1.2.1 int

NRRD Syntax <i>

NRRD Example 12

Python Datatype `int`

Python Example 12

1.2.2 double

NRRD Syntax <d>

NRRD Example 3.14

Python Datatype `float`

Python Example 3.14

1.2.3 string

NRRD Syntax <s>

NRRD Example test

Python Datatype `str`

Python Example 'test'

1.2.4 int list

NRRD Syntax <i> <i> . . . <i>

NRRD Example 1 2 3 4

Python Datatype `numpy.ndarray` (1D, dtype=int)

Python Example np.array([1, 2, 3, 4])

1.2.5 double list

NRRD Syntax <d> <d> . . . <d>

NRRD Example 1.2 2.3 3.4 4.5

Python Datatype `numpy.ndarray` (1D, dtype=float)

Python Example np.array([1.2, 2.3, 3.4, 4.5])

1.2.6 string list

NRRD Syntax <s> <s> … <s>

NRRD Example this is space separated

Python Datatype `list` of `str`

Python Example [‘this’, ‘is’, ‘string’, ‘separated’]

1.2.7 quoted string list

NRRD Syntax “<s>” “<s>” … “<s>”

NRRD Example “one item” “two items” “three” “four”

Python Datatype `list` of `str`

Python Example [‘one item’, ‘two items’, ‘three’, ‘four’]

1.2.8 int vector

NRRD Syntax (<i>,<i>,…,<i>)

NRRD Example (1,2,3,4)

Python Datatype (N,) `numpy.ndarray` of `int`

Python Example np.array([1, 2, 3, 4])

pynrrd will correctly handle vectors with or without spaces between the comma-delimiter. Saving the NRRD file back will remove all spaces between the comma-delimiters.

1.2.9 double vector

NRRD Syntax (<d>,<d>,…,<d>)

NRRD Example (1.2,2.3,3.4,4.5)

Python Datatype (N,) `numpy.ndarray` of `float`

Python Example np.array([1.2, 2.3, 3.4, 4.5])

pynrrd will correctly handle vectors with or without spaces between the comma-delimiter. Saving the NRRD file back will remove all spaces between the comma-delimiters.

1.2.10 int matrix

NRRD Syntax (<i>,<i>,…,<i>) (<i>,<i>,…,<i>) … (<i>,<i>,…,<i>)

NRRD Example (1,0,0) (0,1,0) (0,0,1)

Python Datatype (M,N) `numpy.ndarray` of `int`

Python Example np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])

All rows of the matrix are required, unlike that of the *double matrix*. If some of the rows need to be ‘none’, then use a *double matrix* instead. The reason is that empty rows (i.e. containing ‘none’) are represented as a row of NaN’s, and NaN’s are only available for floating point numbers.

1.2.11 double matrix

NRRD Syntax (<d>,<d>,. . . ,<d>) (<d>,<d>,. . . ,<d>) . . . (<d>,<d>,. . . ,<d>)

NRRD Example (2.54, 1.3, 0.0) (3.14, 0.3, 3.3) none (0.05, -12.3, -3.3)

Python Datatype (M,N) `numpy.ndarray` of `float`

Python Example np.array([[2.54, 1.3, 0.0], [3.14, 0.3, 3.3], [np.nan, np.nan, np.nan], [0.0, -12.3, -3.3]])

This datatype has the added feature where rows can be defined as empty by setting the vector as `none`. In the NRRD specification, instead of the row, the `none` keyword is used in it's place. This is represented in the Python NumPy array as a row of all NaN's. An example use case for this optional row matrix is for the 'space directions' field where one row may be empty because it is not a domain type.

1.3 Supported Fields

A list of supported fields, according to the NRRD specification, and its corresponding datatype can be seen below. Each field has a link to the NRRD specification providing a detailed description of the field.

Field	NRRD Datatype
dimension	*int*
type	*string*
sizes	*int list*
endian	*string*
encoding	*string*
datafile or data file	*string*
lineskip or line skip	*int*
byteskip or byte skip	*int*
content	*string*
kinds	*string list*
labels	*quoted string list*
units	*quoted string list*
min	*double*
max	*double*
spacings	*double list*
thicknesses	*double list*
centerings	*string list*
oldmin or old min	*double*
oldmax or old max	*double*
axismins or axis mins	*double list*
axismaxs or axis maxs	*double list*
sample units	*string*
space	*string*
space dimension	*int*
space units	*quoted string list*
space directions	*double matrix*
space origin	*double vector*
measurement frame	*int matrix*

1.4 Reading NRRD files

There are three functions that are used to read NRRD files: `read()`, `read_header()`, and `read_data()`. `read()` is a convenience function that opens the specified filepath and calls `read_header()` and `read_data()` in succession to return the NRRD header and data.

Reading NRRD files using `read()` or `read_data()` will by default return arrays being indexed in Fortran-order, i.e array elements are accessed by *data[x, y, z]*. This differs from the C-order where array elements are accessed by *data[z, y, x]*, which is the more common order in Python and libraries (e.g. NumPy, scikit-image, PIL, OpenCV). The `index_order` parameter can be used to specify which index ordering should be used on the returned array ('C' or 'F'). The `index_order` parameter needs to be consistent with the parameter of same name in `write()`.

The `read()` and `read_header()` methods accept an optional parameter `custom_field_map` for parsing custom field types not listed in *Supported Fields* of the header. It is a `dict` where the key is the custom field name and the value is a string identifying datatype for the custom field. See *Header datatypes* for a list of supported datatypes.

The `read_data()` will typically be called in conjunction with `read_header()` because header information is required in order to read the data. The function returns a `numpy.ndarray` of the data saved in the given NRRD file.

Some NRRD files, while prohibited by specification, may contain duplicated header fields causing an exception to be raised. Changing `nrrd.reader.ALLOW_DUPLICATE_FIELD` to `True` will show a warning instead of an error while trying to read the file.

1.5 Writing NRRD files

Writing to NRRD files can be done with the function `write()`. The `filename` parameter to the function specifies the absolute or relative filename to write the NRRD file. If the `filename` extension is .nhdr, then the `detached_header` parameter is set to true automatically. If the `detached_header` parameter is set to `True` and the `filename` ends in .nrrd, then the header file will have the same path and base name as the `filename` but with an extension of .nhdr. In all other cases, the header and data are saved in the same file.

The `data` parameter is a `numpy.ndarray` of data to be saved. `header` is an optional parameter of type `dict` containing the field/values to be saved to the NRRD file.

Writing NRRD files will by default index the `data` array in Fortran-order where array elements are accessed by *data[x, y, z]*. This differs from C-order where array elements are accessed by *data[z, y, x]*, which is the more common order in Python and libraries (e.g. NumPy, scikit-image, PIL, OpenCV). The `index_order` parameter can be used to specify which index ordering should be used for the given `data` array ('C' or 'F').

Note: The following fields are automatically generated based on the `data` parameter ignoring these values in the `header`: 'type', 'endian', 'dimension', 'sizes'.

Note: The default encoding field used if not specified in `header` is 'gzip'.

Note: The `index_order` parameter must be consistent with the index order specified in `read()`. Reading an NRRD file in C-order and then writing as Fortran-order or vice versa will result in the data being transposed in the NRRD file.

1.6 Index Ordering

NRRD stores the image elements in row-major ordering where the row can be seen as the fastest-varying axis. The header fields of NRRD that describes the axes are always specified in the order from fastest-varying to slowest-varying, i.e., *sizes* will be equal to *(# rows, # columns)*. This is also applicable to images of higher dimensions.

Both the reading and writing functions in pynrrd include an `index_order` parameter that is used to specify whether the returned data array should be in C-order ('C') or Fortran-order ('F').

Fortran-order is where the dimensions of the multi-dimensional data is ordered from fastest-varying to slowest-varying, i.e. in the same order as the header fields. So for a 3D set of data, the dimensions would be ordered *(x, y, z)*.

On the other hand, C-order is where the dimensions of the multi-dimensional data is ordered from slowest-varying to fastest-varying. So for a 3D set of data, the dimensions would be ordered *(z, y, x)*.

C-order is the index order used in Python and many Python libraries (e.g. NumPy, scikit-image, PIL, OpenCV). pynrrd recommends using C-order indexing to be consistent with the Python community. However, as of this time, the default indexing is Fortran-order to maintain backwards compatibility. In the future, the default index order will be switched to C-order.

Note: Converting from one index order to the other is done via transposing the data.

Note: All header fields are specified in Fortran order, per the NRRD specification, regardless of the index order. For example, a C-ordered array with shape (60, 800, 600) would have a sizes field of (600, 800, 60).

1.7 Example reading and writing NRRD files with C-order indexing

```python
import numpy as np
import nrrd

# Treat this data array as a 3D volume using C-order indexing
# This means we have a volume with a shape of 600x800x70 (x by y by z)
data = np.zeros((70, 800, 600))
# Save the NRRD object with the correct index order
nrrd.write('output.nrrd', data, index_order='C')

# Read the NRRD file with C-order indexing
# Note: We can specify either index order here, this is just a preference unlike in␣
↪nrrd.write where
# it MUST be set based on the data setup
data, header = nrrd.read('output.nrrd', index_order='C')

# The data shape is exactly the same shape as what we wrote
print(data.shape)
>>> (70, 800, 600)

# But the shape saved in the header is in Fortran-order
print(header['sizes'])
>>> [600 800  70]

# Read the NRRD file with Fortran-order indexing now
data, header = nrrd.read('output.nrrd', index_order='F')
```

(continues on next page)

(continued from previous page)

```
# The data shape is exactly the same shape as what we wrote
# The data shape is now in Fortran order, or transposed from what we wrote
print(data.shape)
>>> (600, 800, 70)
print(header['sizes'])
>>> [600 800  70]
```

CHAPTER 2

Examples

2.1 Basic Example

```
import numpy as np
import nrrd

data = np.linspace(1, 50, 50)
nrrd.write('output.nrrd', data)

data2, header = nrrd.read('output.nrrd')
print(np.all(data == data2))
>>> True

print(header)
>>> OrderedDict([('type', 'double'), ('dimension', 1), ('sizes', array([50])), (
↪'endian', 'little'), ('encoding', 'gzip')])
```

2.2 Example only reading header

```
import numpy as np
import nrrd

data = np.linspace(1, 50, 50)
nrrd.write('output.nrrd', data)

header = nrrd.read_header('output.nrrd')
print(header)
>>> OrderedDict([('type', 'double'), ('dimension', 1), ('sizes', array([50])), (
↪'endian', 'little'), ('encoding', 'gzip')])
```

2.3 Example with fields and custom fields

```
import numpy as np
import nrrd

data = np.linspace(1, 60, 60).reshape((3, 10, 2))
header = {'kinds': ['domain', 'domain', 'domain'], 'units': ['mm', 'mm', 'mm'],
↪'spacings': [1.0458, 1.0458, 2.5], 'space': 'right-anterior-superior', 'space␣
↪directions': np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), 'encoding': 'ASCII',
↪'custom_field_here1': 24.34, 'custom_field_here2': np.array([1, 2, 3, 4])}
custom_field_map = {'custom_field_here1': 'double', 'custom_field_here2': 'int list'}

nrrd.write('output.nrrd', data, header, custom_field_map=custom_field_map)

data2, header2 = nrrd.read('output.nrrd', custom_field_map)

print(np.all(data == data2))
>>> True

print(header)
>>> {'units': ['mm', 'mm', 'mm'], 'spacings': [1.0458, 1.0458, 2.5], 'custom_field_
↪here1': 24.34, 'space': 'right-anterior-superior', 'space directions': array([[1, 0,
↪ 0],
   [0, 1, 0],
   [0, 0, 1]]), 'type': 'double', 'encoding': 'ASCII', 'kinds': ['domain', 'domain',
↪'domain'], 'dimension': 3, 'custom_field_here2': array([1, 2, 3, 4]), 'sizes': [3,␣
↪10, 2]}

print(header2)
>>> OrderedDict([('type', 'double'), ('dimension', 3), ('space', 'right-anterior-
↪superior'), ('sizes', array([ 3, 10,  2])), ('space directions', array([[1., 0., 0.
↪],
   [0., 1., 0.],
   [0., 0., 1.]])), ('kinds', ['domain', 'domain', 'domain']), ('encoding', 'ASCII'),␣
↪('spacings', array([1.0458, 1.0458, 2.5   ])), ('units', ['mm', 'mm', 'mm']), (
↪'custom_field_here1', 24.34), ('custom_field_here2', array([1, 2, 3, 4]))])
```

2.4 Example reading NRRD file with duplicated header field

```
import nrrd

# Set this field to True to enable the reading of files with duplicated header fields
nrrd.reader.ALLOW_DUPLICATE_FIELD = True

# Name of the file you want to read with a duplicated header field
filename = "filename.nrrd"

# Read the file
# filedata = numpy array
# fileheader = header of the NRRD file
# A warning is now received about duplicate headers rather than an error being thrown
filedata, fileheader = nrrd.read(filename)
>>> UserWarning: Duplicate header field: 'space' warnings.warn(dup_message)
```

CHAPTER 3

Reference Guide

3.1 Table of Contents

3.1.1 Reading NRRD files

nrrd.read(filename[, custom_field_map, . . .])	Read a NRRD file and return the header and data
nrrd.read_header(file[, custom_field_map])	Read contents of header and parse values from `file`
nrrd.read_data(header[, fh, filename, . . .])	Read data from file into `numpy.ndarray`
nrrd.reader.ALLOW_DUPLICATE_FIELD	Allow duplicate header fields when reading NRRD files

3.1.2 Writing NRRD files

nrrd.write(filename, data[, header, . . .])	Write `numpy.ndarray` to NRRD file

3.1.3 Parsing NRRD fields

nrrd.parse_number_auto_dtype(x)	Parse number from string with automatic type detection.
nrrd.parse_number_list(x[, dtype])	Parse NRRD number list from string into (N,) `numpy.ndarray`.
nrrd.parse_vector(x[, dtype])	Parse NRRD vector from string into (N,) `numpy.ndarray`.
nrrd.parse_optional_vector(x[, dtype])	Parse optional NRRD vector from string into (N,) `numpy.ndarray` or `None`.
nrrd.parse_matrix(x[, dtype])	Parse NRRD matrix from string into (M,N) `numpy.ndarray`.
nrrd.parse_optional_matrix(x)	Parse optional NRRD matrix from string into (M,N) `numpy.ndarray` of `float`.

3.1.4 Formatting NRRD fields

nrrd.format_number(x)	Format number to string
nrrd.format_number_list(x)	Format a (N,) `numpy.ndarray` into a NRRD number list.
nrrd.format_vector(x)	Format a (N,) `numpy.ndarray` into a NRRD vector string
nrrd.format_optional_vector(x)	Format a (N,) `numpy.ndarray` into a NRRD optional vector string
nrrd.format_matrix(x)	Format a (M,N) `numpy.ndarray` into a NRRD matrix string
nrrd.format_optional_matrix(x)	Format a (M,N) `numpy.ndarray` of `float` into a NRRD optional matrix string

3.2 NRRD Module

`nrrd.`**`read`**(*filename*, *custom_field_map=None*, *index_order='F'*)

Read a NRRD file and return the header and data

See *Reading NRRD files* for more information on reading NRRD files.

Note: Users should be aware that the *index_order* argument needs to be consistent between *nrrd.read* and *nrrd.write*. I.e., reading an array with *index_order='F'* will result in a transposed version of the original data and hence the writer needs to be aware of this.

Parameters

filename [`str`] Filename of the NRRD file

custom_field_map [`dict` (`str`, `str`), optional] Dictionary used for parsing custom field types where the key is the custom field name and the value is a string identifying datatype for the custom field.

index_order [{'C', 'F'}, optional] Specifies the index order of the resulting data array. Either 'C' (C-order) where the dimensions are ordered from slowest-varying to fastest-varying (e.g. (z, y, x)), or 'F' (Fortran-order) where the dimensions are ordered from fastest-varying to slowest-varying (e.g. (x, y, z)).

Returns

data [`numpy.ndarray`] Data read from NRRD file

header [`dict` (`str`, `Object`)] Dictionary containing the header fields and their corresponding parsed value

See also:

`write()`, ***`read_header()`***, ***`read_data()`***

`nrrd.`**`read_data`**(*header*, *fh=None*, *filename=None*, *index_order='F'*)

Read data from file into `numpy.ndarray`

The two parameters `fh` and `filename` are optional depending on the parameters but it never hurts to specify both. The file handle (`fh`) is necessary if the header is attached with the NRRD data. However, if the NRRD data is detached from the header, then the `filename` parameter is required to obtain the absolute path to the data file.

See *Reading NRRD files* for more information on reading NRRD files.

Parameters

header [`dict` (`str`, `Object`)] Parsed fields/values obtained from *`read_header()`* function

fh [file-object, optional] File object pointing to first byte of data. Only necessary if data is attached to header.

filename [`str`, optional] Filename of the header file. Only necessary if data is detached from the header. This is used to get the absolute data path.

index_order [{'C', 'F'}, optional] Specifies the index order of the resulting data array. Either 'C' (C-order) where the dimensions are ordered from slowest-varying to fastest-varying (e.g. (z, y, x)), or 'F' (Fortran-order) where the dimensions are ordered from fastest-varying to slowest-varying (e.g. (x, y, z)).

Returns

data [`numpy.ndarray`] Data read from NRRD file

See also:

`read()`, ***`read_header()`***

`nrrd.`**`read_header`**(*file*, *custom_field_map=None*)

Read contents of header and parse values from `file`

`file` can be a filename indicating where the NRRD header is located or a string iterator object. If a filename is specified, then the file will be opened and closed after the header is read from it. If not specifying a filename, the `file` parameter can be any sort of iterator that returns a string each time `next()` is called. The two common objects that meet these requirements are file objects and a list of strings. When `file` is a file object, it must be opened with the binary flag ('b') on platforms where that makes a difference, such as Windows.

See *Reading NRRD files* for more information on reading NRRD files.

Parameters

file [`str` or string iterator] Filename, file object or string iterator object to read NRRD header from

custom_field_map [`dict` (`str`, `str`), optional] Dictionary used for parsing custom field types where the key is the custom field name and the value is a string identifying datatype for the custom field.

Returns

header [`dict` (`str`, `Object`)] Dictionary containing the header fields and their corresponding parsed value

See also:

`read()`, ***`read_data()`***

`nrrd.`**`write`**(*filename*, *data*, *header=None*, *detached_header=False*, *relative_data_path=True*, *custom_field_map=None*, *compression_level=9*, *index_order='F'*)

Write `numpy.ndarray` to NRRD file

The `filename` parameter specifies the absolute or relative filename to write the NRRD file to. If the `filename` extension is .nhdr, then the `detached_header` parameter is set to true automatically. If the `detached_header` parameter is set to `True` and the `filename` ends in .nrrd, then the header file will have the same path and base name as the `filename` but with an extension of .nhdr. In all other cases, the header and data are saved in the same file.

`header` is an optional parameter containing the fields and values to be added to the NRRD header.

Note: The following fields are automatically generated based on the `data` parameter ignoring these values in the `header`: 'type', 'endian', 'dimension', 'sizes', and 'data file'. In addition, the generated fields will be added to the given `header`. Thus, one can check the generated fields by viewing the passed `header`.

Note: The default encoding field used if not specified in `header` is 'gzip'.

Note: The `index_order` parameter must be consistent with the index order specified in `read()`. Reading an NRRD file in C-order and then writing as Fortran-order or vice versa will result in the data being transposed in the NRRD file.

See *Writing NRRD files* for more information on writing NRRD files.

Parameters

- **filename** [`str`] Filename of the NRRD file
- **data** [`numpy.ndarray`] Data to save to the NRRD file
- **detached_header** [`bool` or `str`, optional] Whether the header and data should be saved in separate files. Defaults to `False`. If a `str` is given this specifies the path to the datafile. This path will ONLY be used if the given filename ends with nhdr (i.e. the file is a header)
- **relative_data_path** [`bool`] Whether the data filename in detached header is saved with a relative path or absolute path. This parameter is ignored if there is no detached header. Defaults to `True`
- **custom_field_map** [`dict` (`str`, `str`), optional] Dictionary used for parsing custom field types where the key is the custom field name and the value is a string identifying datatype for the custom field.
- **compression_level** [`int`] Integer between 1 to 9 specifying the compression level when using a compressed encoding (gzip or bzip). A value of `1` compresses the data the least amount and is the fastest, while a value of `9` compresses the data the most and is the slowest.
- **index_order** [{'C', 'F'}, optional] Specifies the index order used for writing. Either 'C' (C-order) where the dimensions are ordered from slowest-varying to fastest-varying (e.g. (z, y, x)), or 'F' (Fortran-order) where the dimensions are ordered from fastest-varying to slowest-varying (e.g. (x, y, z)).

See also:

`read()`, ***`read_header()`***, ***`read_data()`***

`nrrd.`**`format_number_list`**(*x*)

Format a (N,) `numpy.ndarray` into a NRRD number list.

See *int list* and *double list* for more information on the format.

Parameters

x [(N,) `numpy.ndarray`] Vector to convert to NRRD number list string

Returns

list [`str`] String containing NRRD list

`nrrd.`**`format_number`**`(x)`

Format number to string

Function converts a number to string. For numbers of class `float`, up to 17 digits will be used to print the entire floating point number. Any padding zeros will be removed at the end of the number.

See *int* and *double* for more information on the format.

Note: IEEE754-1985 standard says that 17 significant decimal digits are required to adequately represent a 64-bit floating point number. Not all fractional numbers can be exactly represented in floating point. An example is 0.1 which will be approximated as 0.10000000000000001.

Parameters

x [`int` or `float`] Number to convert to string

Returns

vector [`str`] String of number `x`

`nrrd.`**`format_matrix`**`(x)`

Format a (M,N) `numpy.ndarray` into a NRRD matrix string

See *int matrix* and *double matrix* for more information on the format.

Parameters

x [(M,N) `numpy.ndarray`] Matrix to convert to NRRD vector string

Returns

matrix [`str`] String containing NRRD matrix

`nrrd.`**`format_optional_matrix`**`(x)`

Format a (M,N) `numpy.ndarray` of `float` into a NRRD optional matrix string

Function converts a (M,N) `numpy.ndarray` of `float` into a string using the NRRD matrix format. For any rows of the matrix that contain all NaNs for each element, the row will be replaced with a 'none' indicating the row has no vector.

See *double matrix* for more information on the format.

Note: `x` must have a datatype of float because NaN's are only defined for floating point numbers.

Parameters

x [(M,N) `numpy.ndarray` of `float`] Matrix to convert to NRRD vector string

Returns

matrix [`str`] String containing NRRD matrix `nrrd.`**`format_optional_vector`**(*x*)

Format a (N,) `numpy.ndarray` into a NRRD optional vector string

Function converts a (N,) `numpy.ndarray` or `None` into a string using NRRD vector format. If the input `x` is `None`, then `vector` will be 'none'

See *int vector* and *double vector* for more information on the format.

Parameters

x [(N,) `numpy.ndarray` or `None`] Vector to convert to NRRD vector string

Returns

vector [`str`] String containing NRRD vector

`nrrd.`**`format_vector`**(*x*)

Format a (N,) `numpy.ndarray` into a NRRD vector string

See *int vector* and *double vector* for more information on the format.

Parameters

x [(N,) `numpy.ndarray`] Vector to convert to NRRD vector string

Returns

vector [`str`] String containing NRRD vector

`nrrd.`**`parse_matrix`**(*x*, *dtype=None*)

Parse NRRD matrix from string into (M,N) `numpy.ndarray`.

See *int matrix* and *double matrix* for more information on the format.

Parameters

x [`str`] String containing NRRD matrix

dtype [data-type, optional] Datatype to use for the resulting Numpy array. Datatype can be `float`, `int` or `None`. If `dtype` is `None`, then it will be automatically determined by checking any of the elements for fractional numbers. If found, then the matrix will be converted to `float`, otherwise `int`. Default is to automatically determine datatype.

Returns

matrix [(M,N) `numpy.ndarray`] Matrix that is parsed from the `x` string

`nrrd.`**`parse_number_auto_dtype`**(*x*)

Parse number from string with automatic type detection.

Parses input string and converts to a number using automatic type detection. If the number contains any fractional parts, then the number will be converted to float, otherwise the number will be converted to an int.

See *int* and *double* for more information on the format.

Parameters

x [`str`] String representation of number

Returns

result [`int` or `float`] Number parsed from `x` string

`nrrd.`**`parse_number_list`**(*x*, *dtype=None*)

Parse NRRD number list from string into (N,) `numpy.ndarray`.

See *int list* and *double list* for more information on the format.

Parameters

x [`str`] String containing NRRD number list

dtype [data-type, optional] Datatype to use for the resulting Numpy array. Datatype can be `float`, `int` or `None`. If `dtype` is `None`, then it will be automatically determined by checking for fractional numbers. If found, then the string will be converted to `float`, otherwise `int`. Default is to automatically determine datatype.

Returns

vector [(N,) `numpy.ndarray`] Vector that is parsed from the `x` string

`nrrd.`**`parse_optional_matrix`**(*x*)

Parse optional NRRD matrix from string into (M,N) `numpy.ndarray` of `float`.

Function parses optional NRRD matrix from string into an (M,N) `numpy.ndarray` of `float`. This function works the same as *`parse_matrix()`* except if a row vector in the matrix is none, the resulting row in the returned matrix will be all NaNs.

See *double matrix* for more information on the format.

Parameters

x [`str`] String containing NRRD matrix

Returns

matrix [(M,N) `numpy.ndarray` of `float`] Matrix that is parsed from the `x` string

`nrrd.`**`parse_optional_vector`**(*x*, *dtype=None*)

Parse optional NRRD vector from string into (N,) `numpy.ndarray` or `None`.

Function parses optional NRRD vector from string into an (N,) `numpy.ndarray`. This function works the same as *`parse_vector()`* except if `x` is ‘none’, `vector` will be `None`

See *int vector* and *double vector* for more information on the format.

Parameters

x [`str`] String containing NRRD vector or ‘none’

dtype [data-type, optional] Datatype to use for the resulting Numpy array. Datatype can be `float`, `int` or `None`. If `dtype` is `None`, then it will be automatically determined by checking any of the vector elements for fractional numbers. If found, then the vector will be converted to `float`, otherwise `int`. Default is to automatically determine datatype.

Returns

vector [(N,) `numpy.ndarray` or `None`] Vector that is parsed from the `x` string or `None` if `x` is ‘none’

`nrrd.`**`parse_vector`**(*x*, *dtype=None*)

Parse NRRD vector from string into (N,) `numpy.ndarray`.

See *int vector* and *double vector* for more information on the format.

Parameters

x [`str`] String containing NRRD vector

dtype [data-type, optional] Datatype to use for the resulting Numpy array. Datatype can be `float`, `int` or `None`. If `dtype` is `None`, then it will be automatically determined by checking any of the vector elements for fractional numbers. If found, then the vector will be converted to `float`, otherwise `int`. Default is to automatically determine datatype.

Returns

vector [(N,) `numpy.ndarray`] Vector that is parsed from the `x` string

`nrrd.reader.`**`ALLOW_DUPLICATE_FIELD = False`**

Allow duplicate header fields when reading NRRD files

When there are duplicated fields in a NRRD file header, pynrrd throws an error by default. Setting this field as `True` will instead show a warning.

Example: Reading a NRRD file with duplicated header field 'space' with field set to `False`.

```
>>> filedata, fileheader = nrrd.read('filename_duplicatedheader.nrrd')
nrrd.errors.NRRDError: Duplicate header field: 'space'
```

Set the field as `True` to receive a warning instead.

```
>>> nrrd.reader.ALLOW_DUPLICATE_FIELD = True
>>> filedata, fileheader = nrrd.read('filename_duplicatedheader.nrrd')
UserWarning: Duplicate header field: 'space' warnings.warn(dup_message)
```

Note: Duplicated fields are prohibited by the NRRD file specification.

CHAPTER 4

Indices and tables

- genindex
- modindex

Python Module Index

Index