ahds Documentation

Release 0.1

Paul K. Korir, PhD

May 17, 2019

Contents

CHAPTER 1

ahds

Table of Contents

· *Conversion classes*

1.1 Overview

ahds is a Python package to parse and handle Amira (R) files. It was developed to facilitate reading of Amira (R) files as part of the EMDB-SFF toolkit.

Note: Amira (R) is a trademark of Thermo Fisher Scientific. This package is in no way affiliated with with Thermo Fisher Scientific.

1.1.1 Use Cases

- Detect and parse Amira (R) headers and return structured data
- Decode data (HxRLEByte, HxZip)
- Easy extensibility to handle previously unencountered data streams

ahds was written and is maintained by Paul K. Korir.

1.2 Installation

Presently, ahds only works with Python 2.7 but will soon work on Python 3. Please begin by installing numpy<1.16 using

```
pip install numpy<1.16
```

because it is needed to run setup.py. Afterwards you may run

```
pip install ahds
```

1.3 License

```
Copyright 2017 EMBL - European Bioinformatics Institute

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing,
software distributed under the License is distributed on an
"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
either express or implied. See the License for the specific
language governing permissions and limitations under the License.
```

1.4 Future Plans

- Write out valid Amira (R) files

1.5 Background and Definitions

`ahds` presently handles two types of Amira (R) files:

- *AmiraMesh* files, which typically but not necessarily have a `.am` extension, and
- *HyperSurface* files, which have `.surf` and represent an older filetype.

Both file types consist of two parts:

- a *header*, and
- one or more *data streams*.

Headers are structured in a modified VRML-like syntax and differ between AmiraMesh and HyperSurface files in some of the keywords used.

A data stream is a sequence of encoded bytes either referred to in the header by some delimiter (usually `@<data_stream_index>`, where `<data_stream_index>` is an integer) or a set of structural keywords (e.g. `Vertices`, `Patches`) expected in a predefined sequence.

1.5.1 Headers in Detail

AmiraMesh and HyperSurface headers can be divided into four main sections:

- **designation**
- **definitions**
- **parameters**, and
- **data pointers**.

The *designation* is the first line and conveys several important details about the format and structure of the file such as:

- filetype (either `AmiraMesh` or `HyperSurface`)
- dimensionality (`3D`)
- format (`BINARY-LITTLE-ENDIAN`, `BINARY` or `ASCII`)
- version (a decimal number e.g. `2.1`
- extra format data e.g. `<hxsurface>` specifying that an AmiraMesh file will contain HyperSurface data

A series of *definitions* follow that refer to data found in the data pointer sections that either begin with the word â€~defineâ€™ or have â€~nâ€™ prepended to a variable. For example:

```
define Lattice 862 971 200
```

or

```
nVertices 85120
```

This is followed by grouped *parameters* enclosed in a series of braces beginning with the word â€~Parametersâ€™. Various parameters are then enclosed each beginning with the name of that group of parameters e.g. â€~Materialsâ€™

```
Parameters {
    # grouped parameters
    Material {
        # the names of various materials with attributes
        Exterior {
            id 0
        }
        Inside {
            id 1,
            Color 0 1 1,
            Transparency 0.5
        }
    }
    Patches {
    # patch attributes
        InnerRegion â€œInsideâ€,
        OuterRegion â€œExteriorâ€,
        BoundaryID 0,
        BranchingPoints 0
    }
    # inline parameters
    GridSize <value>,
    â€¦
}
```

The most important set of parameters are materials as these specify colours and identities of distinct segments/datasets within the file.

Finally, AmiraMesh files list a set of *data pointers* that point to data labels within the file together with additional information to decode the data. We refer to these as data streams because they consist of continuous streams of raw byte data that need to be decoded. Here is an example of data pointers that refer to the location of 3D surface primitives:

```
Vertices { float[3] Vertices } @1
TriangleData { int[7] Triangles } @2
Patches-0 { int Patches-0 } @3
```

These refer to three raw data streams each found beginning with the delimiter `@<number>`. Data stream one (`@1`) is called `Vertices` and consists of float triples, two is called `TriangleData` and has integer 7-tuples and three called `Patches-` is a single integer (the number of patches). In some cases the data pointer contains the data encoding for the corresponding data pointer.

```
Lattice { byte Labels } @1(HxByteRLE,234575740)
```

which is a run-length encoded data stream of the specified length, while

```
Lattice { byte Data } @1(HxZip,919215)
```

contains zipped data of the specified length.

1.5.2 Data Streams in Detail

AmiraMesh data streams are very simple. They always have a start delimiter made of `@` with an index that identifies the data stream. A newline character separates the delimiter with the data stream proper which is either plain ASCII or a binary stream (raw, zipped or encoded).

HyperSurface data streams structured to have the following sections:

```
# Header
Vertices <nvertices>
# vertices data stream

NBranchingPoints <nbranching_points>
NVerticesOnCurves <nvertices_on_curves>
BoundaryCurves <nboundary_curves>
Patches <npatches>
{
InnerRegion <inner_region_name>
OuterRegion <outer_region_name>
BoundaryID <boundary_id>
BranchingPoints <nbranching_points>
Triangles <ntriangles>
# triangles data stream
} # repeats for as <npatches> times
```

HyperSurface data streams can be either plain ASCII or binary.

1.6 ahds Modules

`ahds` has three main modules:

- *ahds.grammar* specifies an EBNF grammar
- *ahds.header*
- *ahds.data_stream*

These modules are tied into a user-level class called *ahds.AmiraFile* that does all the work for you.

```
>>> from ahds import AmiraFile
>>> # read an AmiraMesh file
>>> af = AmiraFile('am/test7.am')
>>> af.header
<AmiraHeader with 4 bytes>
>>> # empty data streams
>>> af.data_streams
>>> print af.data_streams
None
>>> # we have to explicitly read to get the data streams
>>> af.read()
>>> af.data_streams
<class 'ahds.data_stream.DataStreams'> object with 13 stream(s): 1, 2, 3, 4, 5, 6, 7,␣
↪8, 9, 10, 11, 12, 13
>>> for ds in af.data_streams:
...   print ds
...
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
```

(continues on next page)

(continued from previous page)

```
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 2,608 bytes
# we get the n-th data stream using the index/key notation
>>> af.data_streams[1].encoded_data
'1 \n2 \n3 \n'
>>> af.data_streams[1].decoded_data
[1, 2, 3]
>>> af.data_streams[2].encoded_data
'69 \n120 \n116 \n101 \n114 \n105 \n111 \n114 \n0 \n73 \n110 \n115 \n105 \n100 \n101␣
↪\n0 \n109 \n111 \n108 \n101 \n99 \n117 \n108 \n101 \n0 \n'
>>> af.data_streams[2].decoded_data
[69, 120, 116, 101, 114, 105, 111, 114, 0, 73, 110, 115, 105, 100, 101, 0, 109, 111,␣
↪108, 101, 99, 117, 108, 101, 0]
```

```
>>> # read an HyperSurface file
>>> af = AmiraFile('surf/test4.surf')
>>> af.read()
>>> af.data_streams
<class 'ahds.data_stream.DataStreams'> object with 5 stream(s): Patches,␣
↪NBranchingPoints, BoundaryCurves, Vertices, NVerticesOnCurves
# HyperSurface files have pre-set data streams
>>> af.data_streams['Vertices'].decoded_data[:10]
[(560.0, 243.0, 60.96875), (560.0, 242.9166717529297, 61.0), (559.5, 243.0, 61.0),␣
↪(561.0, 243.0, 60.95833206176758), (561.0, 242.5, 61.0), (561.0384521484375, 243.0,␣
↪61.0), (559.0, 244.0, 60.94444274902344), (559.0, 243.5, 61.0), (558.9722290039062,␣
↪244.0, 61.0), (560.0, 244.0, 60.459999084472656)]
```

1.6.1 `ahds.grammar`

This module describes the header grammar for Amira (R) (AmiraMesh and HyperSurface) files and so depends on `simpleparse` Python package. It defines a single class (*`ahds.grammar.AmiraDispatchProcessor`*) and four functions.

`ahds.grammar.AmiraDispatchProcessor` is a subclass of `simpleparse.dispatchprocessor` which implements the core functionality required to use the grammar. Each grammar token has a corresponding method defined on this class which determines how the data associated with that token will be rendered. Data can be rendered as a single or multimap, string, number, or in custom format.

- `ahds.grammar.get_parsed_data(fn, *args, **kwargs)()` is the user-level function that takes a filename and returns structured parsed data. It depends on the other three functions defined:
- `ahds.grammar.detect_format(fn, format_bytes=50, verbose=False)()` returns either `AmiraMesh` or `HyperSurface` given a file name and arguments,
- `ahds.grammar.get_header(fn, file_format, header_bytes=20000, verbose=False)()` returns the header portion based on the file format determined by detect_format(. . .),

and

- `ahds.grammar.parse_header(data, verbose=False)()` converts the raw header data returned by `ahds.grammar.get_header(...)()` into a structured header based on AmiraDispatchProcessor.

1.6.2 ahds.header

This module converts the structured header from the *ahds.grammar* module into an object with the sections of the header (`designation`, `definitions`, `parameters ``and ``data pointers`) and corresponding structured data available as attributes. That is, it converts the header:

```
# AmiraMesh BINARY-LITTLE-ENDIAN 2.1


define Lattice 862 971 200

Parameters {
    Materials {
        Exterior {
            Id 1
        }
        Inside {
            Color 0.64 0 0.8,
            Id 2
        }
        Mitochondria {
            Id 3,
            Color 0 1 0
        }
        Mitochondria_ {
            Id 4,
            Color 1 1 0
        }
        mitochondria__ {
            Id 5,
            Color 0 0.125 1
        }
        NE {
            Id 6,
            Color 1 0 0
        }
    }
    Content "862x971x200 byte, uniform coordinates",
    BoundingBox 0 13410.7 0 15108.4 1121.45 4221.01,
    CoordType "uniform"
}

Lattice { byte Labels } @1(HxByteRLE,4014522)
```

into an *ahds.header.AmiraHeader* object.

```
>>> from ahds.header import AmiraHeader
>>> amira_header = AmiraHeader.from_file('am/test2.am')
>>> amira_header.designation.attrs
['filetype', 'dimension', 'format', 'version', 'extra_format']
>>> amira_header.designation.filetype
'AmiraMesh'
```

(continues on next page)

(continued from previous page)

```
>>> amira_header.designation.dimension
>>> amira_header.designation.format
'BINARY-LITTLE-ENDIAN'
>>> amira_header.definitions.attrs
['Lattice']
>>> amira_header.definitions.Lattice
[862, 971, 200]
>>> amira_header.parameters.attrs
['Materials', 'Content', 'BoundingBox', 'CoordType']
>>> amira_header.parameters.Materials.attrs
['Exterior', 'Inside', 'Mitochondria', 'Mitochondria_', 'mitochondria__', 'NE']
>>> amira_header.parameters.Materials.Exterior.attrs
['Id']
>>> amira_header.parameters.Materials.Exterior.Id
1
>>> amira_header.parameters.Content
'"862x971x200 byte, uniform coordinates",'
>>> amira_header.parameters.BoundingBox
[0, 13410.7, 0, 15108.4, 1121.45, 4221.01]
>>> amira_header.parameters.CoordType
'"uniform"'
>>> amira_header.data_pointers.attrs
['data_pointer_1']
>>> amira_header.data_pointers.data_pointer_1.attrs
['pointer_name', 'data_format', 'data_dimension', 'data_type', 'data_name', 'data_
↪index', 'data_length']
>>> amira_header.data_pointers.data_pointer_1.pointer_name
'Lattice'
>>> amira_header.data_pointers.data_pointer_1.data_format
'HxByteRLE'
>>> amira_header.data_pointers.data_pointer_1.data_dimension
>>> amira_header.data_pointers.data_pointer_1.data_type
'byte'
>>> amira_header.data_pointers.data_pointer_1.data_name
'Labels'
>>> amira_header.data_pointers.data_pointer_1.data_index
1
>>> amira_header.data_pointers.data_pointer_1.data_length
4014522
```

This module consists of two main classes: *ahds.header.AmiraHeader* is the user-level class and *ahds.header.Block* which is a container class for a block of structured data from an Amira (R) header.

AmiraHeader has one constructor: `ahds.header.AmiraHeader.from_file(fn, *args, **kwargs)()` which takes an Amira (R) file by name and arguments and returns an *ahds.header.AmiraHeader* object with all attributes set as described above. Alternatively, one can use the initiator form to pass structured data directly: `ahds.header.AmiraHeader(parsed_data)` which returns an *ahds.header.AmiraHeader* object configured appropriately.

- The raw data structured data is available as read-only property: *ahds.header.AmiraHeader.raw_header*
- Internally the *ahds.header.AmiraHeader* class implements a set of private methods which individually load the four data sections (`designation`, `definitions`, `parameters`, and `data pointers`).

The *ahds.header.Block* class is a container class which converts structured groups to attributes and has two main attributes:

- `ahds.header.Block.name` provides the name of the current block

```
>>> amira_header.designation.name
'designation'
>>> amira_header.parameters.Materials.name
'Materials'
>>> amira_header.parameters.Materials.Exterior.name
'Exterior'
```

- `ahds.header.Block.attrs` provides the attributes available on this *ahds.header.Block*

```
>>> amira_header.designation.attrs
['filetype', 'dimension', 'format', 'version', 'extra_format']
>>> amira_header.designation.format
'BINARY-LITTLE-ENDIAN'
A given Materials block has two special features:
Block.ids returns the list of ids for all materials. This is important when decoding␣
↪HxByteRLE compressed data
Block[id] returns the material for the given id using index notation.
>>> amira_header.parameters.Materials.ids
[1, 2, 3, 4, 5, 6]
>>> amira_header.parameters.attrs
['Materials', 'Content', 'BoundingBox', 'CoordType']
# ids attribute is only available for â€~Materialâ€™ blocks within â€~parametersâ€™␣
↪section
>>> amira_header.parameters.Content.ids
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'str' object has no attribute 'ids'
# we can get the name of a material of the given id
>>> amira_header.parameters.Materials[4].name
'Mitochondria_'
```

1.6.3 ahds.data_stream

This is most complex module implementing a hierarchy of classes describing various data streams within Amira (R) files. It has 22 classes and five functions

Classes

There are three categories of classes:

- A user-level class that encapsulates (2) below.
- Classes describing Amira (R) data streams
- Classes describing AmiraMesh data streams
- Classes describing HyperSurface data streams
- Data conversion classes (AmiraMesh only)
- Classes abstracting images
- Classes abstracting contours

The user-level *ahds.data_stream.DataStreams* class is the preferred way to use the module. It takes the name of an Amira (R) file and encapsulates an iterator of data streams.

```
>>> from ahds import data_stream
>>> data_streams = data_stream.DataStreams('am/test6.am')
>>> data_streams
<class 'ahds.data_stream.DataStreams'> object with 2 stream(s): 1, 2
>>> for ds in data_streams:
...   print ds
...
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 968,909 bytes
<class 'ahds.data_stream.AmiraMeshDataStream'> object of 968,909 bytes
```

Functions

The functions implemented in this module decode data streams.

- *ahds.data_stream.hxbyterle_decode()* decodes HxByteRLE data streams
- ahds.data_stream.hxzip_decode(data_size, data)() unzips zlib-compressed data streams
- ahds.data_stream.unpack_binary(data_pointer, definitions, data)() unpacks the structured data stream according to the attributes specified in the dataâ€™s data pointer
- ahds.data_stream.unpack_ascii(data)() converts rows of ASCII data into numerical data

Classes in Detail

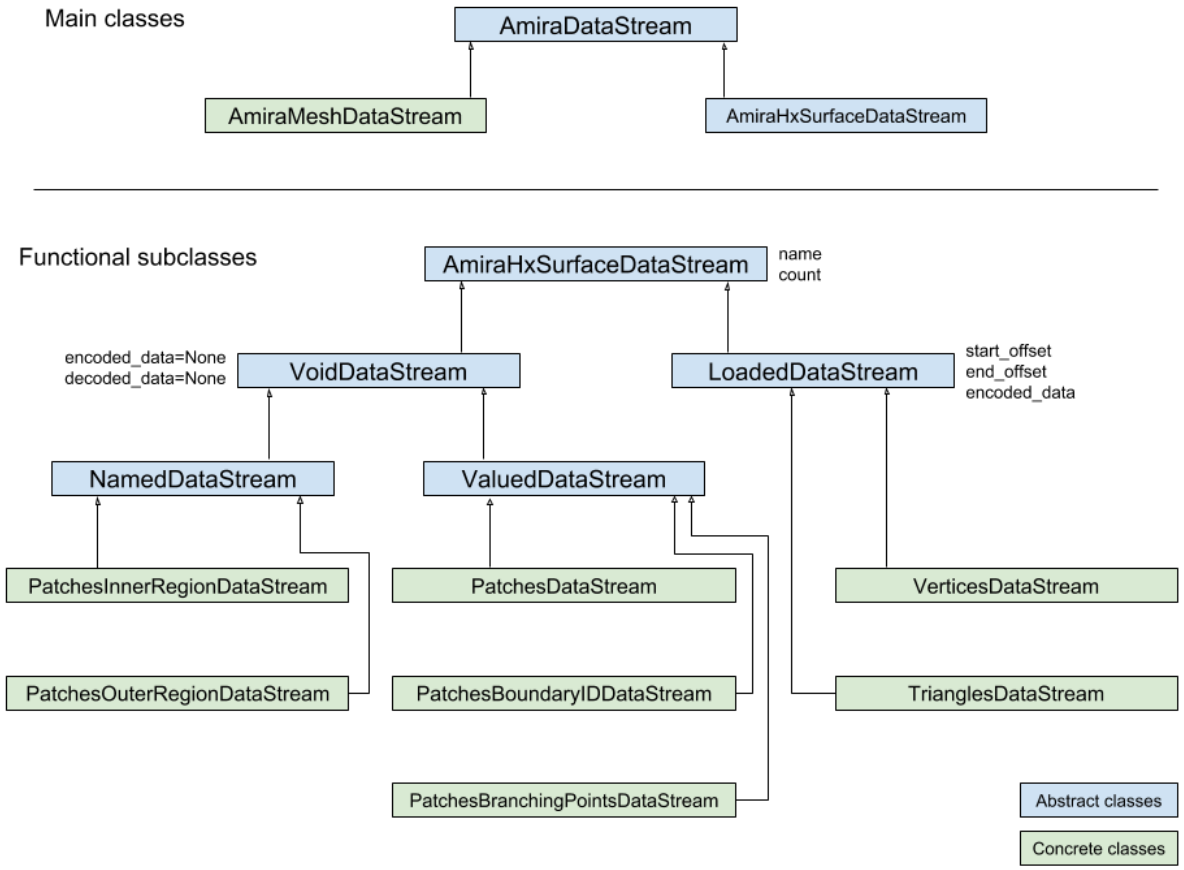
DataStreams class

The following attributes are available on objects of this class:

- ahds.data_stream.DataStreams.file - filename of Amira (R) file
- ahds.data_stream.DataStreams.header - an object of class *ahds.header.AmiraHeader* encapsulating the header data in four sections (designation, definitions, parameters, and data pointers)
- ahds.data_stream.DataStreams.filetype - the filetype as specified in (ii) above.
- ahds.data_stream.DataStreams.stream_data - all raw data from the file (including the header)
- len(DataStreams) - the number of data streams contained
- ahds.data_stream.DataStreams[<index>] - returns the data stream of the index specified (as defined in the data_pointers section of the header object

Classes describing Amira (R) data streams

The following diagrams illustrates the hierarchy of classes:



Classes describing Amira (R) data streams

- *ahds.data_stream.AmiraDataStream* is the base class for all data stream classes and defines the following attributes:
- *ahds.data_stream.AmiraDataStream.header* - an *ahds.header.AmiraHeader* object
- *ahds.data_stream.AmiraDataStream.data_pointer* - the `ahds.header.AmiraHeader.data_pointers.data_pointer_X` for this data stream
- *ahds.data_stream.AmiraDataStream.stream_data* - the raw file data
- *ahds.data_stream.AmiraDataStream.encoded_data* - the encoded data for this stream; `None` for `VoidDataStream` subclasses
- *ahds.data_stream.AmiraDataStream.decoded_data* - the decoded data for this stream; `None` for `VoidDataStream` subclasses
- *ahds.data_stream.AmiraDataStream.decoded_length* - the number of items (tuples, integers) in decoded data

The two main subclasses of *ahds.data_stream.AmiraDataStream* are *ahds.data_stream.AmiraMeshDataStream*, which is a concrete class representing all AmiraMesh data streams, and *ahds.data_stream.AmiraHxSurfaceDataStream*, which abstractly defines HyperSurface data streams.

There are two main AmiraHxSurfaceDataStream subclasses:

- *ahds.data_stream.VoidDataStream* represents *ahds.data_stream.AmiraHxSurfaceDataStream* data streams that only have a name and value but no actual encoded

data (on the following line). There are two subclasses:

- *ahds.data_stream.NamedDataStream* subclasses have a strings after data stream name. The two concrete subclasses are:
- *ahds.data_stream.PatchesInnerRegionDataStream* for the name of an inner region of a patch (see PatchesDataStream), and
- *ahds.data_stream.PatchesOuterRegionDataStream* for corresponding name of the outer region of a patch.
- *ahds.data_stream.ValuedDataStream* have an integer value after the data stream name. The three concrete subclasses are:
- *ahds.data_stream.PatchesBoundaryIDDataStream* hold the boundary ID of a patch,
- *ahds.data_stream.PatchesBranchingPointsDataStream* stores the number of branching points, and
- *ahds.data_stream.PatchesDataStream* with the number of patches, which is a special ahds.data_stream.ValueDataStream that contains an iterable of patches each containing a Patches<X>DataStream objects.
- *ahds.data_stream.LoadedDataStream* represent *ahds.data_stream.AmiraHxSurfaceDataStream* data streams that have a name, a value and encoded data. The two main concrete subclasses are:
- *ahds.data_stream.VerticesDataStream* represents data streams with float-triples, and
- *ahds.data_stream.PatchesTrianglesDataStream* represents data streams within a patch with triples of 1-based indices (triangles) of vertices specified in the *ahds.data_stream.VerticesDataStream*.

Conversion classes

There are two groups of conversion classes which only apply to (some) AmiraMesh data streams: Conversion classes

- Image conversion classes consist of a image container class *ahds.data_stream.ImageSet* and an *ahds.data_stream.Image* class. ImageSet objects that can be iterated to give *ahds.data_stream.Image* objects are returned from the ahds.data_stream.AmiraMeshDataStream.to_images() method call.

```
>>> # decode the data stream to images
>>> images = ds[1].to_images()
>>> images
<ImageSet with 200 images>
>>> for image in images:
...     print image
...
<Image with dimensions (971, 862)>
<Image with dimensions (971, 862)>
<Image with dimensions (971, 862)>
...
<Image with dimensions (971, 862)>
<Image with dimensions (971, 862)>
```

- Contour conversion classes convert individual images into sets of contours (*ahds.data_stream.ContourSet*) iterable as individual ahds.data_stream.Contours objects. They are obtained

from calls to the *ahds.data_stream.Image.as_contours* property. Furthermore, the `ahds.data_stream.Image.as_segments` property call returns a dictionary of the corresponding *ahds.data_stream.ContourSet* object indexed by the z plane.

```
>>> # contours per image
>>> # the dictionary key is the Amira Id for the segment (the Id of the Material)
>>> # a segment can have several non-overlapping contours (or polylines)
>>> for image in images:
...     print image.as_contours
...
{2: <class 'ahds.data_stream.ContourSet'> with 15 contours, 3: <class 'ahds.data_
↪stream.ContourSet'> with 3 contours, 5: <class 'ahds.data_stream.ContourSet'> with␣
↪2 contours}
{2: <class 'ahds.data_stream.ContourSet'> with 18 contours, 3: <class 'ahds.data_
↪stream.ContourSet'> with 3 contours, 5: <class 'ahds.data_stream.ContourSet'> with␣
↪2 contours}
...
{2: <class 'ahds.data_stream.ContourSet'> with 15 contours, 3: <class 'ahds.data_
↪stream.ContourSet'> with 1 contours, 5: <class 'ahds.data_stream.ContourSet'> with␣
↪3 contours}
{2: <class 'ahds.data_stream.ContourSet'> with 15 contours, 3: <class 'ahds.data_
↪stream.ContourSet'> with 1 contours, 5: <class 'ahds.data_stream.ContourSet'> with␣
↪3 contours}


>>> # separate individual segments
>>> images.segments
{1: {110: <class 'ahds.data_stream.ContourSet'> with 1 contours}, 2: {0: <class 'ahds.
↪data_stream.ContourSet'> with 15 contours, 1: <class 'ahds.data_stream.ContourSet'>␣
↪with 18 contours, ..., 198: <class 'ahds.data_stream.ContourSet'> with 3 contours,␣
↪199: <class 'ahds.data_stream.ContourSet'> with 3 contours}}
```

CHAPTER 2

ahds package

class ahds.**AmiraFile**(*fn*, **args*, ***kwargs*)
Bases: object

Convenience class to handle Amira (R) files

This class aggregates user-level classes from the *ahds.header* and *ahds.data_stream* modules into a single class with a simple interface AmiraFile.header() for the header and AmiraFile.data_streams data streams attribute.

2.1 ahds.grammar module

Grammar to parse headers in Amira (R) files

class ahds.grammar.**AmiraDispatchProcessor**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Class defining methods to handle each token specified in the grammar

ahds.grammar.**detect_format**(*fn*, *format_bytes=50*, *verbose=False*, **args*, ***kwargs*)
Detect Amira (R) file format (AmiraMesh or HyperSurface)

Parameters

- **fn** (*str*) – file name
- **format_bytes** (*int*) – number of bytes in which to search for the format [default: 50]
- **verbose** (*bool*) – verbose (default) or not

Return str file_format either AmiraMesh or HyperSurface

ahds.grammar.**get_header**(*fn*, *file_format*, *header_bytes=20000*, *verbose=False*, **args*, ***kwargs*)
Apply rules for detecting the boundary of the header

Parameters

- **fn** (*str*) – file name

- **file_format** (*str*) – either AmiraMesh or HyperSurface
- **header_bytes** (*int*) – number of bytes in which to search for the header [default: 20000]

Return str data the header as per the file_format

ahds.grammar.**get_parsed_data**(*fn*, **args*, ***kwargs*)
All above functions as a single function

Parameters **fn** (*str*) – file name

Return list parsed_data structured metadata

ahds.grammar.**parse_header**(*data*, *verbose=False*, **args*, ***kwargs*)
Parse the data using the grammar specified in this module

Parameters **data** (*str*) – delimited data to be parsed for metadata

Return list parsed_data structured metadata

2.2 ahds.header module

Module to convert parsed data from an Amira (R) header into a set of nested objects. The key class is *ahds.header.AmiraHeader*.

Usage:

```
>>> from ahds.header import AmiraHeader
>>> ah = AmiraHeader.from_file('file.am')
>>> print ah
```

Each nested object is constructed from the *ahds.header.Block* class defined.

There are four top-level attributes that every *ahds.header.AmiraHeader* will have:

- designation
- definitions
- parameters
- data_pointers

Each attribute can be queried using the attrs attribute.

```
>>> print ah.data_pointers.attrs
['data_pointer_1', 'data_pointer_2', 'data_pointer_3', 'data_pointer_4', 'data_
↪pointer_5', 'data_pointer_6']
>>> print ah.data_pointers.data_pointer_1
data_pointer_1
pointer_name: VERTEX
data_format: None
data_dimension: 3
data_type: float
data_name: VertexCoordinates
data_index: 1
data_length: None
```

Data pointers are identified by the name data_pointer_<n>.

class ahds.header.**AmiraHeader**(*parsed_data*)

Bases: object

Class to encapsulate Amira (R) metadata

data_pointers
The list of data pointers together with a name, data type, dimension, index, format and length

definitions
Definitions consist of a key-value pair specified just after the designation preceded by the key-word 'define'

designation
Designation of the Amira (R) file defined in the first row

Designations consist of some or all of the following data:

- filetype e.g. AmiraMesh or HyperSurface
- dimensions e.g. 3D
- format e.g. BINARY-LITTLE-ENDIAN
- version e.g. 2.1
- extra format e.g. <hxsurface>

classmethod from_file(*fn*, **args*, ***kwargs*)
Constructor to build an *ahds.header.AmiraHeader* object from a file

Parameters **fn** (*str*) – Amira (R) file

Return ah object of class *ahds.header.AmiraHeader* containing header metadata

Return type ah: *ahds.header.AmiraHeader*

parameters
The set of parameters for each of the segments specified e.g. colour, data pointer etc.

raw_header
Show the raw header data

class ahds.header.**Block**(*name*)

Bases: object

Generic block to be loaded with attributes

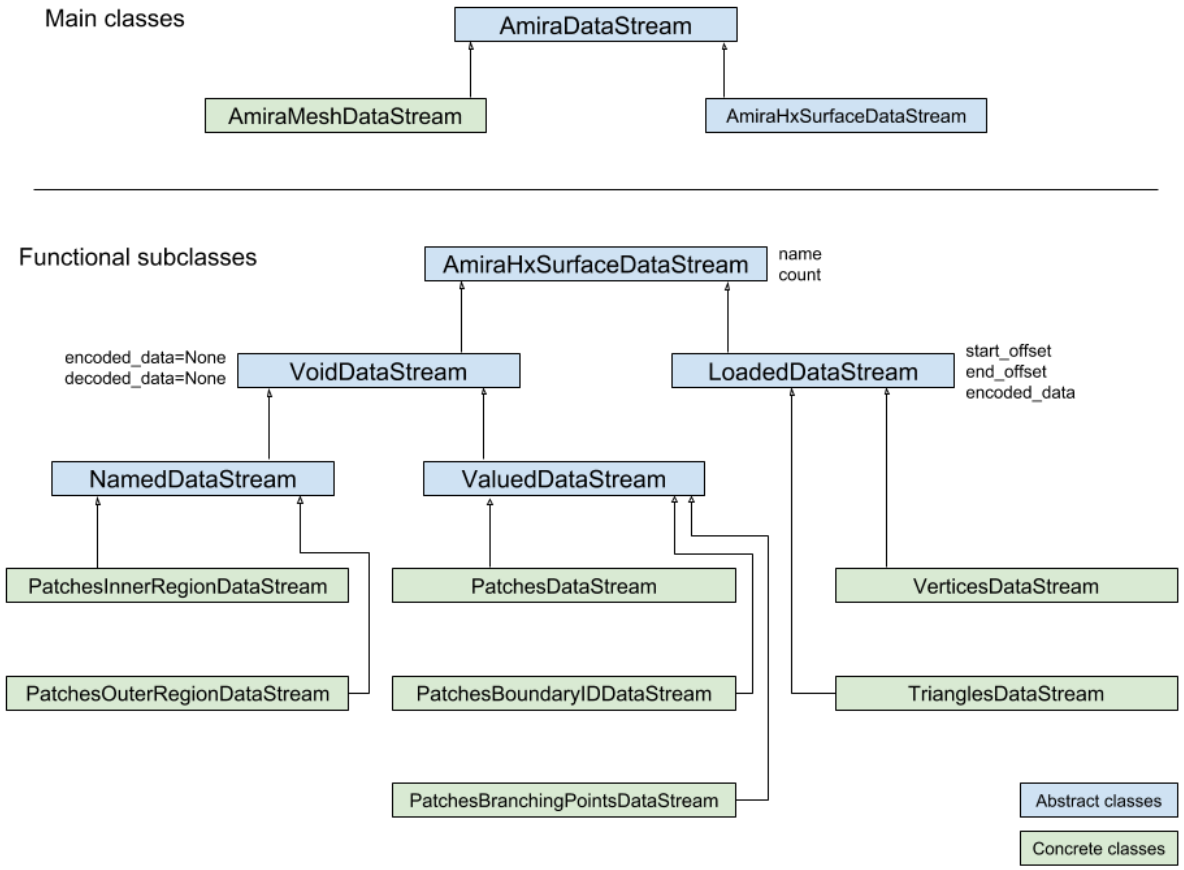
add_attr(*name*, *value*)
Add an attribute to an *ahds.header.Block* object

ids
Convenience method to get the IDs for Materials present

2.3 ahds.data_stream module

data_stream

The following image shows the class hierarchy for data streams.



class ahds.data_stream.**AmiraDataStream**(*amira_header*, *data_pointer*, *stream_data*)

Bases: object

Base class for all Amira DataStreams

data_pointer
The data pointer for this data stream

decoded_data
Decoded data for this stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

encoded_data
Encoded raw data in this stream

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**AmiraHxSurfaceDataStream**(**args*, ***kwargs*)

Bases: *ahds.data_stream.AmiraDataStream*

Base class for all HyperSurface data streams that inherits from *ahds.data_stream.AmiraDataStream*

data_pointer
The data pointer for this data stream

decoded_data
Decoded data for this stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

encoded_data
Encoded raw data in this stream

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**AmiraMeshDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.AmiraDataStream*

Class encapsulating an AmiraMesh data stream

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

to_volume()
Return a 3D volume of the data

class ahds.data_stream.**BoundaryCurvesDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.ValuedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**Contour**(*z*, *array*)
Bases: object

Encapsulates the array representing a contour

class ahds.data_stream.**ContourSet**(*initlist=None*)
Bases: UserList.UserList

Encapsulation for a set of *ahds.data_stream.Contour* objects

append(*item*)
S.append(object) – append object to the end of the sequence

count(*value*) → integer – return number of occurrences of value

extend(*other*)
S.extend(iterable) – extend sequence by appending elements from the iterable

index(*value*) → integer – return first index of value.
Raises ValueError if the value is not present.

insert(*i*, *item*)
S.insert(index, object) – insert object before index

pop([*index*]) → item – remove and return item at index (default last).
Raise IndexError if list is empty or index is out of range.

remove(*item*)
S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

reverse()
S.reverse() – reverse *IN PLACE*

class ahds.data_stream.**DataStreams**(*fn*, **args*, ***kwargs*)
Bases: object

Class to encapsulate all the above functionality

class ahds.data_stream.**Image**(*z*, *array*)
Bases: object

Encapsulates individual images

array
Accessor to underlying array data

as_contours
A dictionary of lists of contours keyed by byte_value

equalise()
Increase the dynamic range of the image

show()
Display the image

class ahds.data_stream.**ImageSet**(*initlist=None*)
Bases: UserList.UserList

Encapsulation for set of *ahds.data_stream.Image* objects

append(*item*)
S.append(object) – append object to the end of the sequence

count(*value*) → integer – return number of occurrences of value

extend(*other*)
S.extend(iterable) – extend sequence by appending elements from the iterable

index(*value*) → integer – return first index of value.
Raises ValueError if the value is not present.

insert(*i*, *item*)
S.insert(index, object) – insert object before index

pop([*index*]) → item – remove and return item at index (default last).
Raise IndexError if list is empty or index is out of range.

remove(*item*)
S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

reverse()
S.reverse() – reverse *IN PLACE*

segments
A dictionary of lists of contours keyed by z-index

class ahds.data_stream.**LoadedDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.AmiraHxSurfaceDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**NBranchingPointsDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.ValuedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**NVerticesOnCurvesDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.ValuedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**NamedDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.VoidDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**PatchesBoundaryIDDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.ValuedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**PatchesBranchingPointsDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.ValuedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**PatchesDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.LoadedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**PatchesInnerRegionDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.NamedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**PatchesOuterRegionDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.NamedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**PatchesTrianglesDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.LoadedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**ValuedDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.VoidDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**VerticesDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.LoadedDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

class ahds.data_stream.**VoidDataStream**(**args*, ***kwargs*)
Bases: *ahds.data_stream.AmiraHxSurfaceDataStream*

data_pointer
The data pointer for this data stream

decoded_length
The length of the decoded stream data in relevant units e.g. tuples, integers (not bytes)

header
An *ahds.header.AmiraHeader* object

stream_data
All the raw data from the file

ahds.data_stream.**byterle_decoder**(*data*, *output_size*)
Python drop-in replacement for compiled equivalent

Parameters

- **output_size** (*int*) – the number of items when `data` is uncompressed
- **data** (*str*) – a raw stream of data to be unpacked

Return numpy.array output an array of `numpy.uint8`

ahds.data_stream.**hxbyterle_decode**(*output_size*, *data*)
Decode HxRLE data stream

If C-extension is not compiled it will use a (slower) Python equivalent

Parameters

- **output_size** (*int*) – the number of items when `data` is uncompressed
- **data** (*str*) – a raw stream of data to be unpacked

Return numpy.array output an array of `numpy.uint8`

ahds.data_stream.**hxzip_decode**(*data_size*, *data*)
Decode HxZip data stream

Parameters

- **data_size** (*int*) – the number of items when `data` is uncompressed
- **data** (*str*) – a raw stream of data to be unpacked

Return numpy.array output an array of `numpy.uint8`

ahds.data_stream.**unpack_ascii**(*data*)
Unpack ASCII data using string methods‘‘

Parameters

- **data_pointer** (*ahds.header.Block*) – metadata for the `data_pointer` attribute for this data stream
- **definitions** (*ahds.header.Block*) – definitions specified in the header
- **data** (*bytes*) – raw binary data to be unpacked

Return list output unpacked data

ahds.data_stream.**unpack_binary**(*data_pointer*, *definitions*, *data*)
Unpack binary data using `struct.unpack`

Parameters

- **data_pointer** (*ahds.header.Block*) – metadata for the `data_pointer` attribute for this data stream
- **definitions** (*ahds.header.Block*) – definitions specified in the header
- **data** (*bytes*) – raw binary data to be unpacked

Return tuple output unpacked data

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

P

R

S

T

U

V