nhf_reader: An Overview

06.01.2025, Nanosurf Library v 1.11 and above

nhf_reader: Processing data of Studio files

Studio Software save measurements in files with extension NHF. They are based on the HDF-File Standard (**H**ierarchical **D**ata **F**ormat).

The Python library has a module **nhf_reader** which can read such files.

Starting with Studio Fluorine the NHF files have a new structure, but the new nhf_reader tries to hide this restructuring as much as possible.

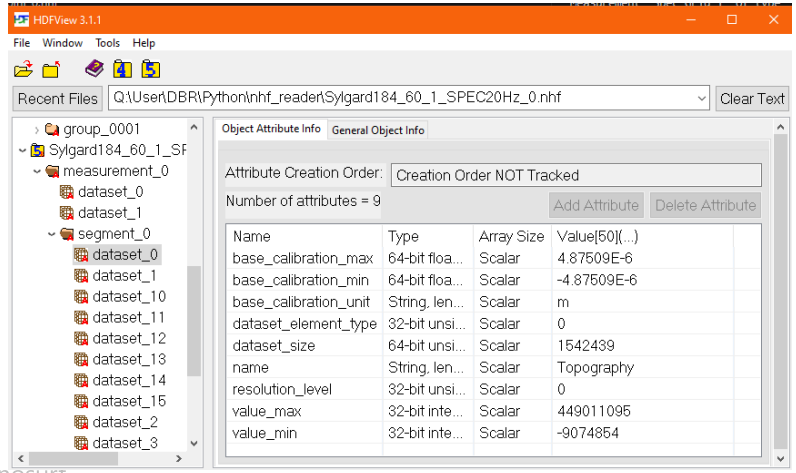
NHF-File Structure Overview

Studio saves data in ‘Measurement’ groups: (e.g Spectroscopy, Image,…)

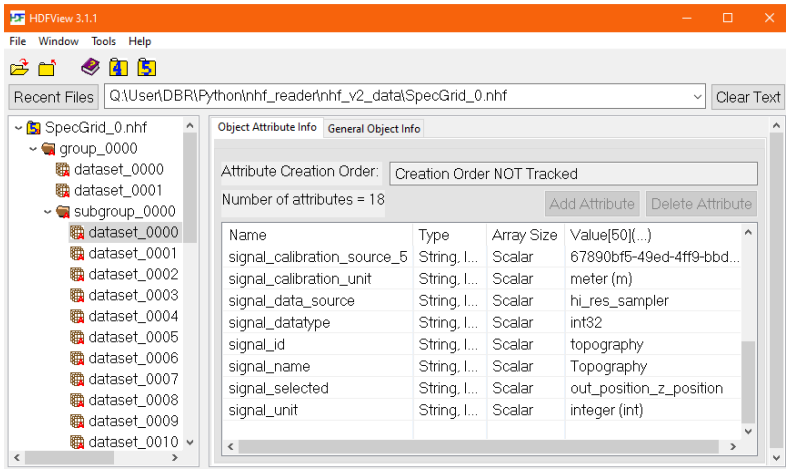
Each measurement have ‘Segment’ groups inside (e.g Forward Scan”, “Advanced to Setpoint”,…).

A Segment then contains the data in dataset called ‘Channels’: (e.g “Topography”, “Deflection”, …)

Version 1 of Data files



Version 2 of Data files



Open File and print structure

nhf_reader class reads files of version 1 and 2 and keep the naming convention with ‘measurement’ and ‘segment’.

Named measurements, segments, and data channels can be accessed by their name.

Unnamed groups, subgroups or dataset are provided as such unprocessed.

Here is an example of how to open a data file and print its structure

```
from nanosurf.lib.util import nhf_reader

source_file =r"Q:\User\DBR\Python\nhf_reader\nhf_v2_data\SpecGrid_0.nhf"

# open the file and show the data structure
try:
    with nhf_reader.NHFFileReader(source_file) as nhf_file:
        print(f"File version: {nhf_file.version()}")
        nhf_file.print_file_structure()

except Exception as e:
    print(f"Could not read file: {source_file}.\nReason: {e}")
```

Looking inside the file structure

nhf_reader.print_file_structure() shows the structure of the file in a hierarchical text form:

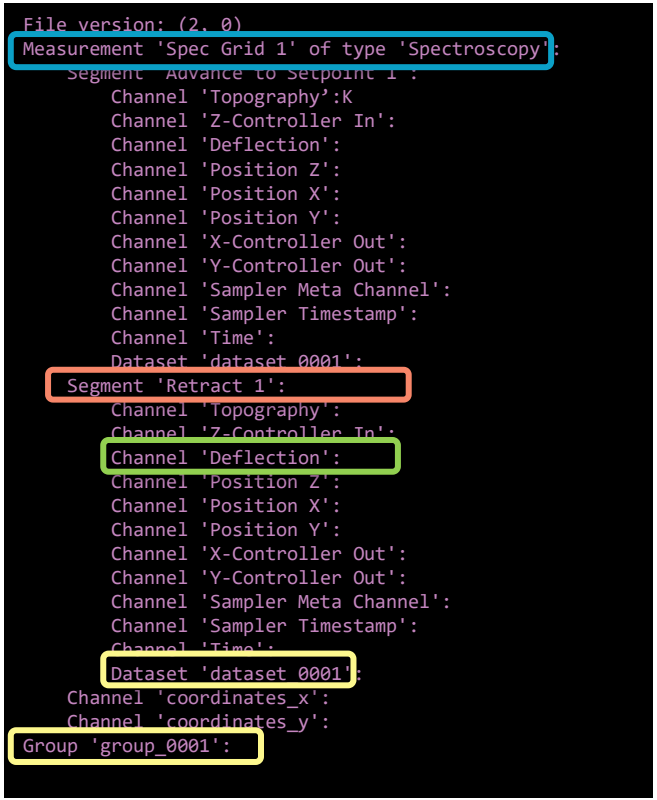
First level: Measurment with ‘name’ and type

Second level: Segment with ‘name’

Channel: Dataset with ‘name’

Dataset: Unknown Dataset

Group: Unknown Group



Nhf_reader's internal file model

Highest level is the **NHFFileReader** class:

It contains access to top level groups of measurements and attributes. (e.g., e.g., Image measurement, file version attribute)

Each measurement is stored by a **NHFMeasurement** class:

It contains access to multiple segments and attributes (e.g. A Backward Image segment, attribute containing x/y range information)

Each segment is stored by a **NHFSegement** class:

It contains access to multiple data channels and attributes (e.g., Topography channel, x/y data points information)

Each data channel is stored by a **NHFDataset** class.

It contains access to the channel data array (or matrix), name, unit and other attributes.

Each level has a member 'attribute' which stores a dictionary of each attribute by key='attribute name' and value='attribute value'.

NHFData channels are read from file by the segment's member function **read_channel().**

Unknown groups or dataset can be at each level of the file hierarchy. Such data is stored as member 'groups or 'datasets' in they original unmodified form. (e.g., spectroscopy point information grid)

In case a hierarchy level has an attribute called "**property**", its content is provided as individual attribute and as a SciVal class member in the member variable 'property' of a NHFSegment or NHFMeasurement class.

Reading data channels

Data channels are part of a segment and can be read by the function **read_channel()**

Accessing a known named data channel :

```
measurement = nhf_file.measurement['Spec Grid 1']
if measurement.measurement_type == nhf_reader.NHFMeasurementType.Spectroscopy:
    first_segment = measurement.segment['Advance to Setpoint 1']
    ch_topo = first_segment.read_channel("Topography")
```

Accessing a data channel by index or in a loop:

```
measurement = nhf_file.measurements[0]
first_segment = measurement.segments[0]
for ch in fwd_ segment.channels:
    print(f"{ch.name=}, {ch.signals()=}, {ch.units()=}")
```

Reading imaging data as matrix:

```
ch_topo = fwd_segment.read_channel("Topography", as_matrix=True)
print(f“X/Y-Shape of dataset={ch_topo.dataset.shape}")
```

Reading imaging data with NaN-Values and make sure its unit it ‘m’:

```
ch_topo = fwd_segment.read_channel("Topography", as_matrix=True, with_unit="m")
if ch_topo.has_nan_values():
    ch_masked_topo = ch_topo.get_masked_dataset()
```

read_channel() function

Full function description of read_channel()

```
    def read_channel(self, ch:typing.Union[str, int], as_matrix:bool=False, with_signal:str=None, with_unit:str=None) -> NHFDataset:
        """ Loads a specified data channel by name or index.
            The data provided by physical units as specified in the Otherwise, calibration information.

            For file version 2.0 and newer multiple calibrations per channel are possible.
            The default calibration and unit is defined by the selected signal at the time of file storage.

            Returns:
            --------
                NHFDataset:
                    A class holding the channel data itself and supplementary information (e.g. unit)
                    If the channels has invalid numbers (e.g., not measured data points) these numbers are converted to NHFDataset.get_nan_value().
                    One can check if such numbers are in the data set by NHFDataset.has_nan_values().
                    if there are NaN values a masked data array can be provided by NHFDataSet.get_masked_dataset() to process only valid data points.

            Parameters:
            -----------
            ch: str, int
                Define the channel to load by its name or its index.

            as_matrix: bool, optional
                If set to True, the dataset is converted to a 2D numpy array. Otherwise, the data is a 1D array

            with_signal: str, optional
                If a signal name is provided, the corresponding calibration is used for the channel.
                If not defined, the channels default signal is used.
                Possible signals can be accessed by self.channel_signals()

            with_unit: str
                If a unit name is provided, the corresponding calibration is used for the channel.
                If not defined, the channels default unit is used.
                Possible signals can be accessed by self.channel_units()
        """
```

Reading Attributes

Each element in the structure (file, measurement, segment, dataset) have its attributes.

These can be accessed by the member self.attribute[“name”], where “name” is the name of an attribute.

Interesting are the attributes of a measurement, because there are important values about the measurement stored:

```
measurement = nhf_file.measurement['Spec Grid 1’]
op_mode = measurement.attribut['measurement_mode']
speed = measurement.attribut[‘scan_line_rate']
```