Documentation: Camera Library and File Reader Module

Generated Documentation

February 4, 2026

Contents

1 Overview

This document provides comprehensive documentation for two Python modules used in the ePDF project data processing pipeline:

- `camera_library.py`: TEM detector specifications database
- `filereader.py`: Image file I/O and metadata extraction utilities

These modules facilitate loading DM4 files from electron microscopy experiments, extracting detector specifications, calculating electron wavelengths, and normalizing image data.

2 Module: camera_library.py

2.1 Overview

`camera_library.py` contains a database of TEM detector specifications with their technical parameters and aliases for flexible identification.

2.2 Data Structure: DETECTOR_LIBRARY

A dictionary containing detector specifications with the following structure:

```
DETECTOR_LIBRARY = {
    'detector_name': {
        'aliases': [regex_patterns],
        'binning': int,
        'description': str,
        'image_height': int,
        'image_width': int,
        'pixel_size': float
    },
    ...
}
```

2.2.1 Included Detectors

Camera	Width	Height	Pixel Size	Type	Binning
Gatan 80X Metro	2048	2048	5.0 µm	DED	1
K2 IS	3840	3712	5.0 µm	DED	1
OneView	4096	4096	15.0 µm	sCMOS	1
Orius SC200D	2048	2048	7.4 µm	CCD	1
Ultrascan	2048	2048	14.0 µm	CCD	1

Table 1: Detector specifications in DETECTOR_LIBRARY

3 Module: filereader.py

3.1 Overview

`filereader.py` provides utilities for loading electron microscopy data files, extracting metadata, calculating electron wavelengths, and organizing detector information.

3.2 Core Functions

3.2.1 extract_camera_type()

```
extract_camera_type(metadata, detector_lib=DETECTOR_LIBRARY)
```

Purpose: Identifies the camera type from HyperSpy metadata using flexible regex pattern matching.

Algorithm:

1. Extract the `General.title` field from metadata
2. Check for exact match in detector library
3. Search for matches against regex alias patterns (case-insensitive)
4. If regex fails, fall back to substring matching
5. Return `(camera_key, camera_title)` or `(None, None)`

Parameters:

- `metadata`: HyperSpy metadata object
- `detector_lib`: Dictionary mapping camera names to specifications (default: `DETECTOR_LIBRARY`)

Returns:

- Tuple: `(camera_key, title)` or `(None, title)` if not found

3.2.2 _search_metadata_recursive()

```
_search_metadata_recursive(obj, target_keys, depth=0, max_depth
    =10)
```

Purpose: Recursively traverses metadata objects to locate specific fields.

Algorithm:

1. Iteratively search object attributes and dictionary keys
2. Match keys case-insensitively against `target_keys`
3. Recursively descend into nested objects up to `max_depth`
4. Extract numeric and string values only

Parameters:

- `obj`: Metadata object to search

- `target_keys`: List of key names (case-insensitive)
- `depth`: Current recursion depth
- `max_depth`: Maximum depth to prevent infinite recursion (default: 10)

Returns:

- Dictionary: {`key: value`} of matching fields found

3.2.3 extract_wavelength()

```
extract_wavelength(metadata=None, voltage_kv=None)
```

Purpose: Calculates or extracts electron wavelength using de Broglie formula with relativistic correction.

Physical Foundation:

The de Broglie wavelength of relativistic electrons is given by:

$$\lambda = \frac{h}{\sqrt{2m_e eV\left(1 + \frac{eV}{2m_e c^2}\right)}}$$

where:

- $h = 6.62607015 \times 10^{-34}$ J·s (Planck constant)
- $m_e = 9.1093837015 \times 10^{-31}$ kg (electron mass)
- $e = 1.602176634 \times 10^{-19}$ C (elementary charge)
- $c = 299792458$ m/s (speed of light)
- V = accelerating voltage in volts

Algorithm:

1. Search metadata for wavelength field (priority: *wavelength* > *energy* > *voltage*)
2. If wavelength found ($0.001 < \lambda < 1$ Å), return directly
3. If voltage found, calculate wavelength using de Broglie formula
4. Apply relativistic correction factor: $1 + \frac{eV}{2m_e c^2}$
5. Convert from SI (meters) to Ångströms ($\times 10^{10}$)

Parameters:

- `metadata`: HyperSpy metadata object (optional)
- `voltage_kv`: Accelerating voltage in kV (optional)

Returns:

- Float: Wavelength in Ångströms, or `None` if not calculable

Priority Search Order:

1. Direct wavelength field in metadata
2. *beam_energy* or *energy* fields
3. *accelerating_voltage* or *acceleration_voltage* fields
4. Fallback to provided `voltage_kv` parameter

3.2.4 get_detector_params()

```
get_detector_params(camera_key, detector_lib=DETECTOR_LIBRARY)
```

Purpose: Retrieves detector parameters from library.
Algorithm:

1. Look up `camera_key` in detector library
2. Copy parameters dictionary
3. Remove `aliases` field from result
4. Return parameters or warning if not found

Parameters:

- `camera_key`: Name of detector in library
- `detector_lib`: Detector database (default: `DETECTOR_LIBRARY`)

Returns:

- Dictionary containing: `pixel_size`, `image_width`, `image_height`, `binning`, `description`
- `None` if camera not found

3.2.5 load_data()

```
load_data(file, normalize=True, verbose=True)
```

Purpose: Comprehensive file loading with automatic metadata extraction and image normalization.
Algorithm:

1. Load DM4 file using HyperSpy
2. Extract metadata object
3. Identify camera type using `extract_camera_type()`
4. Retrieve detector parameters
5. Calculate wavelength using `extract_wavelength()`
6. Search for exposure time in metadata

7. If `normalize=True`: divide image by exposure time
8. Return detector info dictionary and raw image array

Parameters:

- `file`: Path to DM4 file
- `normalize`: Normalize by exposure time (default: `True`)
- `verbose`: Print loading information (default: `True`)

Returns:

- Tuple: `(detector_info, raw_image)`
- `detector_info`: Dictionary with keys:
  - `camera_type`: Identified camera name
  - `camera_title`: Metadata title string
  - `wavelength`: Wavelength in Ångströms
  - `pixel_size`: Pixel size in micrometers
  - `image_width`, `image_height`: Image dimensions
  - `exposure_time`: Exposure time in seconds
  - `binning`: Binning factor
  - `description`: Detector description
- `raw_image`: NumPy 2D array of image data

Normalization:
If `exposure_time` is available and `normalize=True`:

$$I_{\text{normalized}} = \frac{I_{\text{raw}}}{t_{\text{exp}}}$$

This corrects for varying exposure times across measurements.

3.2.6 add_detector()

```
add_detector(camera_key, pixel_size, image_width, image_height,
             binning=1, description='', aliases=None)
```

Purpose: Adds a new detector to the library and persists to disk.
Algorithm:

1. Check if `camera_key` already exists (reject duplicates)
2. Create specification dictionary
3. Add to in-memory `DETECTOR_LIBRARY`
4. Serialize library to `camera_library.py`

5. Print confirmation message

Parameters:

- `camera_key`: Unique identifier for detector
- `pixel_size`: Pixel size in micrometers
- `image_width`: Image width in pixels
- `image_height`: Image height in pixels
- `binning`: Binning factor (default: 1)
- `description`: Optional description string
- `aliases`: List of regex patterns for identification (optional)

Returns:

- `True` if successful, `False` if key already exists

3.2.7 _save_detector_library()

```
_save_detector_library()
```

Purpose: Internal function to persist `DETECTOR_LIBRARY` to `camera_library.py`.
Algorithm:

1. Format `DETECTOR_LIBRARY` as formatted Python code (pretty-printed)
2. Generate file header comment
3. Write to `camera_library.py` in module directory
4. Handle write errors gracefully

Returns:

- None (prints success/error messages)

4 Usage Examples

4.1 Basic File Loading

```
from filereader import load_data

detector_info, raw_image = load_data('path/to/file.dm4')
print(f"Camera: {detector_info['camera_type']}")
print(f"Wavelength: {detector_info['wavelength']:.4f}  ")
print(f"Image shape: {raw_image.shape}")
```

4.2 Camera Type Extraction

```
from filereader import extract_camera_type
import hyperspy.api as hs

image = hs.load('file.dm4')
camera_key, title = extract_camera_type(image.metadata)
print(f"Identified camera: {camera_key}")
```

4.3 Wavelength Calculation

```
from filereader import extract_wavelength

# From metadata
wavelength = extract_wavelength(metadata=image.metadata)

# From known voltage
wavelength = extract_wavelength(voltage_kv=200)

print(f"Wavelength: {wavelength:.6f}  ")
```

4.4 Adding New Detector

```
from filereader import add_detector

add_detector(
    camera_key='NewCamera XL',
    pixel_size=6.5,
    image_width=4096,
    image_height=4096,
    binning=1,
    description='New high-resolution detector',
    aliases=['newcamera.*xl', 'xl.*new']
)
```

5 Key Implementation Details

5.1 Metadata Search Strategy

The `_search_metadata_recursive()` function uses a breadth-first approach with the following features:

- Handles both object attributes and dictionary-like access patterns
- Case-insensitive matching for robustness
- Depth limiting to prevent infinite loops in circular references
- Graceful error handling for non-standard metadata objects

5.2 Camera Identification

The `extract_camera_type()` function provides three levels of matching:

1. Exact string match
2. Regex pattern matching (regex errors fall back to substring matching)
3. Substring matching for simple patterns

5.3 Relativistic Correction

The wavelength calculation applies relativistic correction via the factor:

$$\text{relativistic\_factor} = 1 + \frac{eV}{2m_e c^2}$$

For typical TEM energies (100-400 keV), this correction is ∼5-15%, significantly improving accuracy.

5.4 Image Normalization

Exposure time normalization is crucial for quantitative analysis:

$$I_{\text{counts/s}} = \frac{I_{\text{raw}}}{t_{\text{exp}}}$$

This enables direct comparison between images acquired with different exposure times.

6 Error Handling

Both modules implement defensive programming practices:

- *Metadata extraction*: Graceful degradation with warning messages
- *Camera identification*: Falls back to `None` if not found
- *Wavelength calculation*: Returns `None` with warnings if voltage unavailable
- *File I/O*: Try-catch blocks prevent crashes on save/load failures

7 Performance Considerations

- Metadata recursion limited to depth 10 to prevent exponential search times
- Regex compilation occurs during library initialization
- Image data handled as NumPy arrays for efficient processing

8 Dependencies

- `hyperspy`: For DM4 file I/O and metadata handling
- `numpy`: For numerical operations and array handling
- `re`: For regex pattern matching
- `os`: For file path operations