

parseUV v1.0.0

Spectrometer Binary Data Converter and Analysis Toolkit
Technical Documentation, Format Specifications, and User Manual

Dr. Ricardo J. Fernández-Terán

10 August 2026

Contents

1	Introduction	2
2	Detailed Installation Instructions	2
2.1	Step 1 — System Requirements and Python Setup	2
2.2	Step 2 — Installing uv (Recommended Package Manager)	2
2.3	Step 3 — Obtaining the Source Code	3
2.4	Step 4 — Virtual Environment Creation and Installation	3
2.5	Step 5 — Verifying the Installation	3
2.6	Dependencies	3
3	Technical Format Specifications and Metadata Extraction	3
3.1	Varian / Agilent Cary Binary Formats (.DSW and .BSW)	4
3.1.1	Header Structure and Magic Byte Signature	4
3.1.2	Spectral Data Block Scanning Algorithm	4
3.1.3	Wavelength Encodings: Fixed-Step vs. WinUV v3.00+ Empirical Encoder Readings	5
3.1.4	Acquisition Date, Software, and Scan Settings Extraction for Cary Files	5
3.2	Shimadzu Binary Format (.SPC)	6
3.2.1	Primary Strategy: Shimadzu OLE2 Compound Storage Parser	6
3.2.2	Instrumental, Software, and Method Metadata Extraction Engine for SPC Files	6
4	Command-Line Interface (CLI Mode)	7
4.1	CLI Syntax and Command Options	7
4.2	Single-File Conversion Examples	7
4.3	Batch Directory Processing	7
5	PyQt6 Graphical User Interface (GUI Mode)	8
5.1	Launching the Interface	8
5.2	GUI Architecture and Components	8
5.3	Interactive Metadata Display Text Block	8
5.4	Automatic Spectrum Categorization and Subplot Layout	8
5.5	Export Controls and Data Reset	9
6	Practical Code Examples	10
6.1	Example 1 — Reading, Inspecting Acquisition Metadata, and Exporting (.DSW)	10
6.2	Example 2 — Reading Batch Spectra Files (.BSW)	10
6.3	Example 3 — Parsing Shimadzu Files (.SPC) and Accessing Instrumental Parameters	11
6.4	Example 4 — Programmatic Directory Batch Processing	11

1 Introduction

parseUV is a Python package and PyQt6 desktop application designed for reading, processing, visualizing, and exporting proprietary binary files generated by optical UV-Vis-NIR spectrophotometers.

Optical UV-Vis-NIR spectrophotometers commonly export experimental data into proprietary binary file structures—most notably Varian / Agilent Cary single spectrum (.DSW) and batch spectrum (.BSW) files, as well as Shimadzu UVProbe OLE2 Compound Storage (.SPC) files. Inspecting or converting these datasets traditionally depends on proprietary vendor Windows applications.

parseUV provides a pure Python binary parser requiring zero vendor software runtime libraries. It encapsulates individual spectral recordings into structured Spectrum objects, extracts comprehensive acquisition dates, software versions, instrumental models, scan rates, and parameters, provides CaryFile multi-spectrum container abstractions, integrates directly with pandas.DataFrame objects, and includes both a command-line interface (CLI) and an interactive PyQt6 drag-and-drop desktop GUI.

This manual provides detailed installation instructions, low-level technical specifications for the .DSW, .BSW, and .SPC formats (including metadata, date/timestamp, and software extraction engines), a walkthrough of the CLI and PyQt6 GUI modes, and code examples.

Acknowledgements

The author gratefully acknowledges **SpectraGryph** (optical spectroscopy software developed by Dr. Friedrich Menges, <https://www.effmm.de/spectragryph/>) for serving as an explicit inspiration for this project, particularly in spectrometer binary data format discovery, data conversion workflows, and interactive spectroscopy software design.

2 Detailed Installation Instructions

parseUV requires **Python 3.9 or later** and is managed using **uv**, a fast Python project and package manager.

2.1 Step 1 — System Requirements and Python Setup

Confirm that Python 3.9 or higher is installed on your system:

```
python --version # should report Python 3.9.x or later
```

2.2 Step 2 — Installing uv (Recommended Package Manager)

uv provides fast, reproducible environment setup and package resolution. Install uv globally on your system:

Windows (PowerShell).

```
powershell -ExecutionPolicy Bypass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

macOS / Linux.

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Via pip:

```
pip install uv
```

Verify the installation:

```
uv --version
```

2.3 Step 3 — Obtaining the Source Code

Clone the repository or extract the source archive into your working directory:

```
git clone https://github.com/username/parseUV.git
cd parseUV
```

2.4 Step 4 — Virtual Environment Creation and Installation

Create an isolated virtual environment and install parseUV in editable mode:

Core Installation (User Mode).

```
uv venv
uv pip install -e .
```

Full Development Installation.

To install development extras (ruff linter and pytest runner):

```
uv venv
uv pip install -e ".[dev]"
```

Standard pip Installation.

If you prefer standard pip without uv:

```
python -m venv .venv
# Linux / macOS
source .venv/bin/activate
# Windows (PowerShell)
.venv\Scripts\Activate.ps1
pip install -e .
```

2.5 Step 5 — Verifying the Installation

Confirm that parseUV imports correctly and reports version 1.0.0:

```
uv run python -c "import parseuv; print(parseuv.__version__)"
```

Run the automated unit test suite:

```
uv run pytest
```

Launch the desktop GUI:

```
uv run parseuv-gui
```

2.6 Dependencies

[Table 1](#) and [Table 2](#) document the core runtime dependencies and development extras.

3 Technical Format Specifications and Metadata Extraction

This section describes the low-level binary format specifications, scan settings, acquisition dates, and software version extraction engines implemented in `parseuv.parser`.

Table 1: Core runtime dependencies.

Package	Minimum version	Purpose
numpy	1.24.0	Numerical array operations
pandas	2.0.0	Tabular data structures and CSV export
matplotlib	3.8.0	Plotting canvas backend
PyQt6	6.5.0	Graphical desktop application interface
PyQt6-sip	13.5.0	Qt C++ bindings layer
olefile	0.46	OLE2 Compound document stream parser

Table 2: Development extras (`uv pip install -e ".[dev]"`).

Package	Minimum version	Purpose
ruff	0.16.2	Fast Python linter and code formatter
pytest	7.4.0	Automated unit testing framework

3.1 Varian / Agilent Cary Binary Formats (.DSW and .BSW)

Varian / Agilent Cary spectrophotometers (e.g., Cary 50, Cary 5000) save single spectrum measurements as .DSW files and multi-spectrum batch measurements as .BSW files.

3.1.1 Header Structure and Magic Byte Signature

The binary stream begins with a Pascal-style byte-length-prefixed ASCII header at offset 0x00:

1. **Length Byte:** Byte 0x00 defines header string byte length $L = \text{data}[0]$.
2. **Magic ASCII String:** Bytes 0x01 through $1 + L$ contain the magic string. A valid Cary file must start with "Varian UV-VIS".

Global acquisition parameters are decoded at fixed byte locations:

- **End Wavelength:** 32-bit IEEE 754 single-precision float (<f) at offset 0x5D:0x61.
- **Start Wavelength:** 32-bit IEEE 754 single-precision float (<f) at offset 0x61:0x65.
- **Point Count:** 32-bit signed integer (<i) at offset 0x6D:0x71.

3.1.2 Spectral Data Block Scanning Algorithm

Batch .BSW files contain multiple spectral data blocks separated by instrumental header metadata. `parseUV` scans the binary byte payload sequentially using a 16-byte sliding window.

Each 16-byte window represents two adjacent (w, y) pairs packed as 32-bit IEEE 754 little-endian floats:

$$(w_0, y_0) = \text{unpack}("<ff", \text{data}[i : i + 8]) \quad (1)$$

$$(w_1, y_1) = \text{unpack}("<ff", \text{data}[i + 8 : i + 16]) \quad (2)$$

A candidate spectral payload block is identified when an initial pair satisfies:

1. **Wavelength Range:** $190.0 \text{ nm} \leq w_0, w_1 \leq 1100.0 \text{ nm}$.
2. **Step Size:** $0.01 \text{ nm} \leq |\Delta w| \leq 25.0 \text{ nm}$ where $\Delta w = w_1 - w_0$.
3. **Absorbance Signal:** $-10.0 \leq y_0, y_1 \leq 100.0$.

The scan direction is determined by the sign of Δw :

- **Monotonic Decreasing** ($\Delta w < 0$): Enforces step constraint $-30.0 \text{ nm} \leq (w_{k+1} - w_k) \leq -0.001 \text{ nm}$.
- **Monotonic Increasing** ($\Delta w > 0$): Enforces step constraint $0.001 \text{ nm} \leq (w_{k+1} - w_k) \leq 30.0 \text{ nm}$.

The scanner continues reading 8-byte float pairs until a data point violates physical limits or step constraints. Candidate blocks containing fewer than 30 points ($N < 30$) are discarded.

3.1.3 Wavelength Encodings: Fixed-Step vs. WinUV v3.00+ Empirical Encoder Readings

Varian / Agilent Cary binary files store wavelength values directly alongside absorbance measurements as sequential 32-bit IEEE 754 little-endian float pairs (w_k, y_k) . `parseUV` supports both major wavelength encoding paradigms encountered in Cary binary datasets:

1. **Legacy Fixed Arithmetic Steps**: Older Cary software versions output exact theoretical wavelength values calculated at constant step increments (e.g., 800.0, 799.0, 798.0 nm for a 1.0 nm step).
2. **WinUV v3.00+ Empirical Monochromator Encoder Readings**: Newer Cary WinUV software versions [v3.00 and above, e.g., WinUV 3.00(182)] encode direct physical monochromator hardware encoder positions into the 8-byte point records. Rather than rounded integer values, the w_k float array contains empirical encoder readings (e.g., 799.998, 799.026, 798.053, ..., 199.968 nm) that account for real-time motor positioning tolerances.

Furthermore, `parseUV` seamlessly handles variable data sampling intervals across both single (.DSW) and batch (.BSW) files:

- **Standard Fine Sampling**: 0.05, 0.10, 0.50, 1.0, 2.0 nm steps.
- **Fast Coarse Sampling**: Custom step sizes up to 25.0 nm (e.g., 5.0 nm sampling intervals yielding 121 data points across 800 $\rightarrow$ 200 nm).

3.1.4 Acquisition Date, Software, and Scan Settings Extraction for Cary Files

In addition to wavelength and absorbance vectors, Varian Cary binary streams store acquisition timestamps, software identifiers, instrument model designations, and scan speed parameters within a 256-byte ASCII header block positioned immediately prior to each spectral data run (`data_offset - 256`). `parseUV` extracts the following parameters into `CaryFile.header` and `Spectrum.metadata`:

- **Acquisition Date/Time** (`date_time`): Extracted from ASCII timestamp headers or trailer blocks (e.g., "17/05/2021 16:46:35", "6/22/2022 1:54:42 PM").
- **Software Application** (`software`): Identifies the operating vendor software ("Varian Cary WinUV").
- **Instrument Model** (`instrument`): Identifies the physical instrument designation (e.g., "Cary 5000", "Cary 50").
- **Scan Rate / Speed** (`scan_rate`): Extracted scan speed in nm/min (e.g., "600.00 nm/min", "4800.00 nm/min", "24000.00 nm/min").
- **Averaging Time** (`ave_time`): Integration time per point (e.g., "0.1000 s", "0.0125 s").
- **Spectral Bandwidth / Slit Width** (`sbw / slit_width`): Monochromator slit bandwidth in nm (e.g., "2.00 nm").
- **Scan Mode** (`scan_mode`): Identifies continuous sweeps ("Continuous Wavelength Sweep (TContuumStore)").

3.2 Shimadzu Binary Format (.SPC)

Shimadzu UVProbe software outputs spectrum files with the .SPC extension. `parseUV` implements a dual-parser architecture to support both modern Shimadzu OLE2 Compound Storage containers and legacy Galactic GRAMS binary files.

3.2.1 Primary Strategy: Shimadzu OLE2 Compound Storage Parser

Modern Shimadzu .SPC files are OLE2 Compound Storage binary containers identified by the magic signature:

$$\text{Header Magic} = \text{0xD0CF11E0A1B11AE1} \quad (3)$$

When present, `parseUV` opens the document using `olefile` and enumerates internal object streams:

1. **Stream Path Matching:** Locates stream pairs matching the naming convention:

$$\text{DataSpectrumStorage/Data/X Data.N} \quad \text{and} \quad \text{DataSpectrumStorage/Data/Y Data.N} \quad (4)$$

where N is the dataset index number.

2. **Binary Unpacking:** The binary payload of X Data.N (wavelength axis in nm) and Y Data.N (absorbance signal) are decoded as 64-bit IEEE 754 double-precision float arrays (<d):

$$x = \text{unpack}(' < N_{\text{pts}} d', \text{raw_bytes}) \quad (5)$$

3.2.2 Instrumental, Software, and Method Metadata Extraction Engine for SPC Files

Shimadzu OLE2 containers embed structured instrumental method pages, dataset creation histories, and software metadata. `parseUV` implements an automated metadata extraction engine (`_extract_spc_ole_metadata`):

1. **DataSetHeaderInfo Stream:** Extracts institution name (e.g., "Institution Name"), software version ("UVProbe 2.43"), instrument model series ("UV-2400PC Series"), instrument designation ("UV2450"), and instrument serial ID string.
2. **MethodStorage Pages** (PageTexts0 through PageTexts4): Parses key-value method parameters into structured dictionaries:
 - **Measurement Properties:** Scan speed ("Slow"), sampling interval ("1.0"), scan mode ("Single"), wavelength boundaries.
 - **Sample Preparation:** Cell path length ("2 mm"), weight, volume, dilution.
 - **Instrument Properties:** Measuring mode ("Absorbance"), slit width ("1.0 nm"), light source change wavelength ("340.0 nm"), S/R exchange mode ("Normal").
 - **Operation Parameters:** Smoothing threshold, points, interpolation, averaging settings.
3. **DataSetHistory Stream:** Extracts creation log strings and timestamps (e.g., "Created new data set: C153-EtOH_dil - RawData.").

Extracted metadata is automatically stored in `CaryFile.header` and made accessible in each `Spectrum.metadata` object ([Table 3](#)).

Table 3: Sample acquisition dates, software versions, and instrumental metadata populated in CaryFile.header and Spectrum.metadata.

Metadata Key	Origin Stream / Format	Example Value
date_time	ASCII Header / DataSetHistory	"17/05/2021 16:46:35"
software	Header String / DataSetHeaderInfo	"Varian Cary WinUV" / "UVProbe 2.43"
instrument	Header String / DataSetHeaderInfo	"Cary 5000" / "UV-2400PC Series"
scan_rate	Header ASCII Block	"600.00 nm/min"
ave_time	Header ASCII Block	"0.1000 s"
slit_width	Header / MethodStorage	"2.00 nm" / "1.0 nm"
scan_mode	Header / TContinuumStore	"Continuous Wavelength Sweep (TContinuumStore)"
path_length	MethodStorage/PageTexts1	"2 mm"
header_info	DataSetHeaderInfo	["Institution Name", "UVProbe 2.43"]

4 Command-Line Interface (CLI Mode)

The parseuv command-line entry point supports file inspection, CSV conversion, plot generation, and batch directory processing directly from the terminal.

4.1 CLI Syntax and Command Options

```
parseuv [filepath] [--gui] [--batch] [-f {png,pdf,svg}] [-o OUTPUT] [-p PLOT] [--show-plot]
```

Table 4 summarizes the command-line flags accepted by parseuv.

Table 4: Command-line interface arguments for parseuv.

Argument / Option	Description
filepath	Target file path (.DSW, .BSW, .SPC) or folder for batch processing
-gui	Launches the interactive PyQt6 Drag-and-Drop GUI application
-batch	Enables batch conversion mode on the target folder
-f, -format	Plot output image format for batch mode (png, pdf, svg)
-o, -output	Destination path for exported CSV file or output directory
-p, -plot	Path to save rendered plot image (e.g., plot.png)
-show-plot	Opens an interactive window displaying the rendered plot

4.2 Single-File Conversion Examples

Inspect and convert a single Varian Cary .DSW file to CSV and PNG figure:

```
parseuv path/to/spectrum.DSW -o output.csv -p plot.png
```

Parse a Shimadzu .SPC file and display an interactive plot window:

```
parseuv path/to/spectrum.spc --show-plot
```

4.3 Batch Directory Processing

Convert an entire folder of spectrometer files into CSV datasets and vector PDF figures:

```
parseuv --batch path/to/folder -f pdf -o path/to/output
```

If `-output` is omitted, `parseuv` automatically creates format-specific subdirectories:

- `CSV/`: Contains exported CSV spectra files.
- `PDF/` (or `PNG/`, `SVG/`): Contains high-resolution rendered figures.

5 PyQt6 Graphical User Interface (GUI Mode)

`parseUV` includes an interactive desktop GUI built using PyQt6.

5.1 Launching the Interface

Launch the GUI application from the command line using any of the following entry points:

```
parseuv-gui
# or
parseuv --gui
# or
python run_gui.py
```

On Windows systems, `launch_gui()` automatically registers the application user model ID (`parseuv.spectrometer.gui.1.0`) to display the custom prism taskbar icon.

5.2 GUI Architecture and Components

The graphical interface consists of three primary custom Qt widgets:

1. **DropZoneWidget** (QFrame): An interactive drag-and-drop landing area accepting `.DSW`, `.BSW`, and `.SPC` files. Provides visual drag-over feedback (border highlight change from blue to green) and a file browser fallback button.
2. **MplPlotWidget** (QWidget): Wraps Matplotlib's `FigureCanvasQTAgg` canvas, `NavigationToolbar2QT` toolbar, and a read-only metadata display text box (`QTextEdit`) positioned directly below the spectral plot.
3. **MainWindow** (QMainWindow): Manages top-level window state, menu bar actions, status bar messages, and spectrum export triggers.

5.3 Interactive Metadata Display Text Block

Directly below the spectral canvas, `MplPlotWidget` renders a styled **File & Acquisition Metadata** group box (`QGroupBox`). When a file is parsed, the text block formats all extracted acquisition timestamps, operating software, instrument model designations, scan rates, integration times, slit widths, and sample path lengths into bulleted, formatted text lines. If no additional metadata can be parsed from the file, the panel displays "No additional metadata available".

5.4 Automatic Spectrum Categorization and Subplot Layout

When a binary file is loaded into the GUI, `parseUV` categorizes spectra into **sample spectra** and **background / baseline recordings** by matching titles against key terms ("baseline", "background", "dark").

The plot canvas dynamically selects its layout based on the content:

- **Single Plot Layout**: Used when no background spectra exist. All sample spectra are overlaid on a single set of axes.

- **2-Row Subplot Layout (3:1 Height Ratio):** Automatically activated when baseline recordings are present:
 - **Top Subplot** (3x height): Displays sample absorption spectra with solid lines and color cycling.
 - **Bottom Subplot** (1x height): Displays baseline recordings with dashed line styling. Both subplots share an aligned X-axis (wavelength in nm).

5.5 Export Controls and Data Reset

The interface provides three primary control buttons at the bottom of the window:

- **Export Spectra as CSV:** Exports sample spectra combined into a single CSV file.
- **Export Background as CSV:** Exports background and baseline recordings to CSV.
- **Reset:** Clears loaded memory arrays, hides the plot canvas, and restores the drag-and-drop landing zone.

6 Practical Code Examples

This section details practical Python code examples located in the `examples/` directory.

6.1 Example 1 — Reading, Inspecting Acquisition Metadata, and Exporting (.DSW)

The following code (adapted from `examples/demo.py`) parses a single Varian Cary .DSW file, inspects extracted acquisition timestamps, software, and scan speed parameters, exports CSV data, and saves a figure:

```
import os
import matplotlib.pyplot as plt
from parseuv import parse_uv

# Parse Cary .DSW single-spectrum binary file
cary_dsw = parse_uv("path/to/spectrum.DSW")

print(f"File Type: {cary_dsw.file_type}") # DSW
print(f"Software: {cary_dsw.header.get('software')}") # Varian Cary WinUV
print(f"Instrument: {cary_dsw.header.get('instrument')}") # Cary 5000
print(f>Date/Time: {cary_dsw.header.get('date_time')}") # 17/05/2021 16:46:35
print(f"Scan Rate: {cary_dsw.header.get('scan_rate')}") # 600.00 nm/min
print(f"Ave Time: {cary_dsw.header.get('ave_time')}") # 0.1000 s

spectrum = cary_dsw[0]
print(f"Title: {spectrum.title}")
print(f"Wavelengths: {spectrum.start_wavelength} -> {spectrum.end_wavelength} nm")
print(f>Data Points: {spectrum.num_points} (Step: {spectrum.step_size} nm)")

# Export spectrum data to CSV
cary_dsw.to_csv("exported_spectrum.csv")

# Save plot figure
fig, ax = plt.subplots(figsize=(8, 5))
spectrum.plot(show=False, ax=ax, color="#1f77b4")
ax.set_title(f"Cary 5000 Spectrum: {spectrum.title}")
plt.tight_layout()
plt.savefig("dsw_spectrum_plot.png", dpi=300)
plt.close(fig)
```

6.2 Example 2 — Reading Batch Spectra Files (.BSW)

.BSW files contain multiple spectral recordings. The example below parses a batch series, iterates over individual sample and baseline spectra, exports a combined pandas DataFrame, and constructs a two-panel subplot figure:

```
from parseuv import parse_uv
import matplotlib.pyplot as plt

# Parse Cary .BSW batch file
cary_bsw = parse_uv("path/to/spectrum.BSW")
print(f"Found {len(cary_bsw.spectra)} spectra in batch file.")
print(f"Acquisition Date: {cary_bsw.header.get('date_time')}")

# Iterate over all spectra in the file
for s in cary_bsw.spectra:
    print(f" '{s.title}' | {s.num_points} pts | {s.start_wavelength} -> {s.end_wavelength} nm")
```

```

# Convert all spectra into a single aligned pandas DataFrame
df = cary_bsw.to_dataframe()
print(df.head())

# Create a two-row subplot figure for sample and baseline spectra
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8), sharex=True)

# Top panel: Sample spectra
ax1.plot(cary_bsw[0].wavelengths, cary_bsw[0].absorbances, label=cary_bsw[0].title,
        color="#d62728")
ax1.plot(cary_bsw[1].wavelengths, cary_bsw[1].absorbances, label=cary_bsw[1].title,
        color="#ff7f0e", linestyle="--")
ax1.set_ylabel("Absorbance")
ax1.set_title("Sample Measurements")
ax1.legend()

# Bottom panel: Baseline recordings
ax2.plot(cary_bsw[2].wavelengths, cary_bsw[2].absorbances, label=cary_bsw[2].title,
        color="#2ca02c")
ax2.plot(cary_bsw[3].wavelengths, cary_bsw[3].absorbances, label=cary_bsw[3].title,
        color="#9467bd", linestyle=".")
ax2.set_xlabel("Wavelength (nm)")
ax2.set_ylabel("Absorbance")
ax2.set_title("Baseline Recordings")
ax2.legend()

plt.tight_layout()
plt.savefig("bsw_spectra_plot.png", dpi=300)
plt.close(fig)

```

6.3 Example 3 — Parsing Shimadzu Files (.SPC) and Accessing Instrumental Parameters

Reading Shimadzu .SPC files extracts software, institution, and instrumental parameters into both header and metadata:

```

from parseuv import parse_uv

spc_file = parse_uv("path/to/spectrum.spc")
print(f"File Type: {spc_file.file_type}")
print(f"Spectrum: '{spc_file[0].title}' ({spc_file[0].num_points} points)")

# Access extracted instrumental metadata
header = spc_file.header
print(f"Software: {header.get('software')}") # UVProbe 2.43
print(f"Instrument: {header.get('instrument_type')}") # UV-2400PC Series
print(f"Scan Speed: {header.get('scan_speed')}") # Slow
print(f"Slit Width: {header.get('slit_width')}") # 1.0 nm
print(f"Path Length: {header.get('path_length')}") # 2 mm

# Export to CSV
spc_file.to_csv("spc_exported.csv")

```

6.4 Example 4 — Programmatic Directory Batch Processing

The script `examples/batch_demo.py` demonstrates processing an entire directory of mixed .BSW, .DSW, and .SPC files:

```
import os
from parseuv.batch import batch_process_folder

spectra_dir = os.path.abspath("path/to/folder")

# Batch process directory and generate PNG figures
batch_process_folder(spectra_dir, plot_format="png")
```

Executing this script automatically parses every supported binary file in the specified directory and populates output subfolders CSV/ and PNG/.