XPCS WebPlot Documentation

Complete Guide for Users, Developers, and Administrators

XPCS WebPlot Team

2026-01-20

Contents

1 XPCS WebPlot Documentation

Welcome to the XPCS WebPlot documentation! This directory contains comprehensive documentation for the XPCS WebPlot package.

1.1 Documentation Files

- **Getting Started** - Installation, quick start, and basic usage
- **User Guide** - Detailed guide for all features and commands
- **API Reference** - Complete API documentation for all modules
- **Architecture** - System architecture and design decisions
- **Development Guide** - Contributing, testing, and development workflow
- **Deployment Guide** - Production deployment instructions
- **FAQ** - Frequently asked questions and troubleshooting

1.2 Quick Links

- GitHub Repository
- PyPI Package
- Issue Tracker

1.3 About XPCS WebPlot

XPCS WebPlot is a Python package for converting X-ray Photon Correlation Spectroscopy (XPCS) analysis results into interactive web-viewable formats. It provides:

- **Automated Conversion**: Convert HDF5 XPCS analysis files to HTML with plots
- **Web Visualization**: Flask-based web server for browsing results
- **Batch Processing**: Process multiple files and combine results
- **Real-time Monitoring**: Watch directories for new files and auto-process
- **Flexible Output**: Generate plots, export data, and create summaries

1.4 Getting Help

If you encounter issues or have questions:

1. Check the FAQ for common questions
2. Review the User Guide for detailed usage
3. Search existing GitHub Issues
4. Create a new issue if needed

1.5 License

2 Getting Started with XPCS WebPlot

This guide will help you get started with XPCS WebPlot quickly.

2.1 Table of Contents

2.2 Installation

2.2.1 Prerequisites

- Python 3.8 or higher
- pip package manager

2.2.2 Install from Source

```
# Clone the repository
git clone https://github.com/AZjk/xpcs_webplot.git
cd xpcs_webplot

# Install in development mode
pip install -e ".[dev]"
```

2.2.3 Install for Users

```
# Install from PyPI (when available)
pip install xpcs_webplot

# Or install from source without dev dependencies
pip install .
```

2.2.4 Verify Installation

```
# Check that the command is available
xpcs_webplot --help
```

You should see the help message with available subcommands.

2.3 Quick Start

2.3.1 Convert a Single XPCS File

The most basic operation is converting a single XPCS HDF5 file to web format:

```
xpcs_webplot plot /path/to/your/xpcs_file.hdf
```

This will: - Read the XPCS analysis results from the HDF5 file - Generate plots (SAXS, stability, correlation functions) - Export data to text files - Create an HTML summary page - Save everything to `./html/` directory by default

2.3.2 View Results in Browser

Start the Flask web server to view your results:

```
xpcs_webplot serve ./html
```

Then open your browser to `http://localhost:5000` to view the interactive results.

2.3.3 Process Multiple Files

To process all XPCS files in a directory:

```
xpcs_webplot plot /path/to/directory/*.hdf --target-dir ./html
```

2.3.4 Combine Results

After processing multiple files, combine them into a single index page:

```
xpcs_webplot combine ./html
```

2.4 Basic Usage

2.4.1 The `plot` Subcommand

Convert XPCS HDF files to web format:

```
# Basic usage
xpcs_webplot plot input.hdf

# Specify output directory
xpcs_webplot plot input.hdf --target-dir ./output

# Process with custom settings
xpcs_webplot plot input.hdf --num-img 8 --dpi 300

# Overwrite existing results
xpcs_webplot plot input.hdf --overwrite

# Generate only images (no data export)
xpcs_webplot plot input.hdf --image-only

# Skip saving result files
xpcs_webplot plot input.hdf --no-save-result
```

2.4.2 The `combine` Subcommand

Combine multiple result folders into a single index:

```
# Combine all results in a directory
xpcs_webplot combine ./html

# Specify output filename
xpcs_webplot combine ./html --output combined_index.html
```

2.4.3 The `serve` Subcommand

Start a web server to browse results:

```
# Start server on default port (5000)
xpcs_webplot serve ./html

# Specify custom port and host
xpcs_webplot serve ./html --port 8080 --host 0.0.0.0

# Enable debug mode
xpcs_webplot serve ./html --debug
```

2.5 Common Workflows

2.5.1 Workflow 1: Single File Analysis

```
# 1. Convert the file
xpcs_webplot plot data.hdf

# 2. View results
xpcs_webplot serve ./html
```

2.5.2 Workflow 2: Batch Processing

```
# 1. Process all files in a directory
xpcs_webplot plot /data/experiment/*.hdf --target-dir ./results

# 2. Combine results
xpcs_webplot combine ./results

# 3. Serve results
xpcs_webplot serve ./results
```

2.5.3 Workflow 3: Real-time Monitoring

```
# Monitor a directory and auto-process new files
xpcs_webplot plot /data/incoming --monitor --target-dir ./results
```

This will watch the directory and automatically process new HDF files as they appear.

2.6 Understanding the Output

After running `xpcs_webplot plot`, you'll find the following structure:

```
html/
  your_file_name/
      summary.html          # Main summary page
      metadata.json         # Extracted metadata
      saxs.png             # SAXS pattern plot
      stability.png        # Stability analysis plot
      g2_multitau.png      # Multitau correlation plot
      twotime_*.png        # Twotime correlation maps
      c2_saxs.txt          # SAXS data
      c2_g2.txt            # G2 correlation data
      c2_stability.txt     # Stability data
      c2_twotime_*.txt     # Twotime data
```

2.7 Next Steps

Now that you have the basics, explore more advanced features:

- Read the User Guide for detailed documentation
- Check the API Reference for programmatic usage
- See the Development Guide to contribute
- Review Deployment Guide for production setup

2.8 Getting Help

If you run into issues:

1. Check the FAQ
2. Review the error messages carefully
3. Enable verbose logging with `--verbose` flag (if available)
4. Search or create an issue on GitHub

3 Quick Reference

Quick reference guide for common XPCS WebPlot commands and options.

3.1 Installation

```
# Install from source
pip install -e .

# Install with dev dependencies
pip install -e ".[dev]"
```

3.2 Basic Commands

3.2.1 Convert Single File

```
xpcs_webplot plot input.hdf
```

3.2.2 Convert Multiple Files

```
xpcs_webplot plot /path/to/*.hdf
```

3.2.3 Combine Results

```
xpcs_webplot combine ./html
```

3.2.4 Start Web Server

```
xpcs_webplot serve ./html
```

3.3 Common Options

3.3.1 Plot Command

Option	Default	Description
`--target-dir`	`./html`	Output directory
`--num-img`	`4`	Number of twotime images
`--dpi`	`240`	Plot resolution
`--overwrite`	`False`	Overwrite existing results
`--image-only`	`False`	Skip data export
`--no-save-result`	-	Skip saving text files
`--monitor`	`False`	Watch directory for new files
`--num-processes`	`4`	Parallel processes

3.3.2 Serve Command

Option	Default	Description
`--port`	`5000`	Server port
`--host`	`0.0.0.0`	Server host
`--debug`	`False`	Debug mode (dev only)

3.4 Usage Examples

3.4.1 Quick Preview

```
xpcs_webplot plot data.hdf --dpi 120 --num-img 2 --image-only
xpcs_webplot serve ./html
```

3.4.2 High Quality

```
xpcs_webplot plot data.hdf --dpi 600 --num-img 12
```

3.4.3 Batch Processing

```
xpcs_webplot plot /data/*.hdf --num-processes 8 --target-dir ./results
xpcs_webplot combine ./results
```

3.4.4 Monitoring

```
xpcs_webplot plot /data/incoming --monitor --target-dir ./live
```

3.4.5 Custom Server

```
xpcs_webplot serve ./html --port 8080 --host 127.0.0.1
```

3.5 Output Structure

```
html/
  result_name/
      summary.html          # Main page
      metadata.json         # Metadata
      saxs.png             # SAXS plot
      stability.png        # Stability plot
      g2_multitau.png      # G2 plot
      twotime_*.png        # Twotime plots
      c2_saxs.txt          # SAXS data
      c2_g2.txt            # G2 data
      c2_stability.txt     # Stability data
      c2_twotime_*.txt     # Twotime data
```

3.6 Programmatic Usage

3.6.1 Basic Conversion

```
from xpcs_webplot.converter import convert_xpcs_result

convert_xpcs_result('data.hdf', target_dir='output')
```

3.6.2 Batch Processing

```
from xpcs_webplot.webplot_cli import convert_many_files
import glob

files = glob.glob('/data/*.hdf')
convert_many_files(files, num_processes=8)
```

3.6.3 Custom Flask App

```
from xpcs_webplot.flask_app import create_app

app = create_app('my_results')
app.run(host='0.0.0.0', port=5000)
```

3.7 Troubleshooting

3.7.1 Command Not Found

```
pip install -e .
export PATH="$HOME/.local/bin:$PATH"
```

3.7.2 Port Already in Use

```
xpcs_webplot serve ./html --port 8080
```

3.7.3 Slow Processing

```
xpcs_webplot plot /data/*.hdf --num-processes 8 --dpi 120
```

3.7.4 Out of Memory

```
xpcs_webplot plot /data/*.hdf --num-processes 2
```

3.8 File Locations

Item	Location
Source code	`src/xpcs_webplot/`
Templates	`src/xpcs_webplot/templates/`
Tests	`tests/`
Documentation	`docs/`
Configuration	`pyproject.toml`

3.9 Key Modules

Module	Purpose
`cli.py`	Command-line interface
`converter.py`	Core conversion logic
`flask_app.py`	Web server
`plot_images.py`	Plotting functions
`html_utlits.py`	HTML generation
`monitor_and_process.py`	Directory monitoring

3.10 Environment Variables

Currently, XPCS WebPlot does not use environment variables. All configuration is via command-line arguments.

3.11 Keyboard Shortcuts

3.11.1 In Monitoring Mode

- `Ctrl+C` - Stop monitoring

3.11.2 In Web Browser

- Standard browser shortcuts apply

3.12 URLs

Resource	URL
GitHub	https://github.com/AZjk/xpcs_webplot
PyPI	https://pypi.python.org/pypi/xpcs_webplot
Issues	https://github.com/AZjk/xpcs_webplot/issues
Docs	https://xpcs-webplot.readthedocs.io

3.13 Getting Help

1. Check FAQ
2. Read User Guide
3. Search GitHub Issues
4. Create new issue

3.14 See Full Documentation

- Getting Started - Installation and basics
- User Guide - Complete usage guide
- API Reference - API documentation
- Architecture - System design
- Development Guide - Contributing
- Deployment Guide - Production setup
- FAQ - Common questions

4 User Guide

This comprehensive guide covers all features and commands available in XPCS WebPlot.

4.1 Table of Contents

4.2 Command-Line Interface

XPCS WebPlot provides a main command `xpcs_webplot` with three subcommands:

```
xpcs_webplot <subcommand> [options]
```

4.2.1 Available Subcommands

- **`plot`** - Convert XPCS HDF files to web-viewable format with plots and data exports
- **`combine`** - Combine multiple result folders into a unified index page
- **`serve`** - Start Flask web server to browse and view results interactively

4.3 Plot Subcommand

The `plot` subcommand converts XPCS HDF5 analysis files into web-viewable formats.

4.3.1 Basic Syntax

```
xpcs_webplot plot <input_files> [options]
```

4.3.2 Input Specification

You can specify input files in several ways:

```
# Single file
xpcs_webplot plot data.hdf

# Multiple files
xpcs_webplot plot file1.hdf file2.hdf file3.hdf

# Wildcard patterns
xpcs_webplot plot /data/*.hdf

# Directory (processes all .hdf files)
xpcs_webplot plot /data/experiment/
```

4.3.3 Options

4.3.3.1 `--target-dir` (default: `./html`)

Specify the output directory for generated HTML and plots:

```
xpcs_webplot plot data.hdf --target-dir ./results
```

4.3.3.2 `--num-img` (default: `4`)

Number of twotime correlation images to generate:

```
xpcs_webplot plot data.hdf --num-img 8
```

4.3.3.3 `--dpi` (default: 240)

Resolution (dots per inch) for generated plots:

```
xpcs_webplot plot data.hdf --dpi 300
```

4.3.3.4 `--overwrite` (default: `False`)

Overwrite existing results if they already exist:

```
xpcs_webplot plot data.hdf --overwrite
```

4.3.3.5 `--image-only` (default: `False`)

Generate only image files, skip data export to text files:

```
xpcs_webplot plot data.hdf --image-only
```

4.3.3.6 `--no-create-image-directory` (default: creates directory)

Skip creating an 'images' subdirectory within each result folder:

```
xpcs_webplot plot data.hdf --no-create-image-directory
```

4.3.3.7 `--save-result` / `--no-save-result` (default: `True`)

Control whether to save result data to text files:

```
# Skip saving text files
xpcs_webplot plot data.hdf --no-save-result

# Explicitly enable (default behavior)
xpcs_webplot plot data.hdf --save-result
```

4.3.3.8 `--monitor` (default: `False`)

Enable real-time directory monitoring for new files:

```
xpcs_webplot plot /data/incoming --monitor --target-dir ./results
```

When enabled, the program will: - Watch the input directory for new HDF files - Automatically process new files as they appear - Continue running until manually stopped (Ctrl+C)

4.3.3.9 `--num-processes` (default: 4)

Number of parallel processes for batch processing:

```
xpcs_webplot plot /data/*.hdf --num-processes 8
```

4.3.4 Examples

```
xpcs_webplot plot experiment_001.hdf
```

4.3.4.1 Example 1: Basic Conversion

Creates output in `./html/experiment_001/` with: - HTML summary page - SAXS, stability, and correlation plots - Exported data files

```
xpcs_webplot plot data.hdf --dpi 600 --num-img 12 --target-dir ./publication
```

4.3.4.2 Example 2: High-Quality Output

Generates high-resolution plots suitable for publication.

```
xpcs_webplot plot /data/2024_01/*.hdf --target-dir ./results --num-processes 8
```

4.3.4.3 Example 3: Batch Processing

Processes all HDF files in parallel using 8 CPU cores.

```
xpcs_webplot plot /data/live --monitor --target-dir ./live_results
```

4.3.4.4 Example 4: Monitoring Mode

Continuously monitors `/data/live` and processes new files automatically.

4.4 Combine Subcommand

The `combine` subcommand creates a unified index page from multiple result folders.

4.4.1 Basic Syntax

```
xpcs_webplot combine <html_folder> [options]
```

4.4.2 Options

4.4.2.1 `--output` (default: `combined_summary.html`)

Specify the output filename for the combined index:

```
xpcs_webplot combine ./html --output index.html
```

4.4.3 How It Works

The combine command: 1. Scans the specified directory for result folders 2. Extracts metadata from each `summary.html` file 3. Creates a unified index page with: - Sortable table of all results - Links to individual result pages - Summary statistics - Filtering capabilities

4.4.4 Examples

```
xpcs_webplot combine ./html
```

4.4.4.1 Example 1: Basic Combine

Creates `./html/combined_summary.html` with all results.

```
xpcs_webplot combine ./results --output master_index.html
```

4.4.4.2 Example 2: Custom Output

Creates `./results/master_index.html`.

4.5 Serve Subcommand

The `serve` subcommand starts a Flask web server for interactive result browsing.

4.5.1 Basic Syntax

```
xpcs_webplot serve <html_folder> [options]
```

4.5.2 Options

4.5.2.1 `--port` (default: `5000`)

Specify the port number for the web server:

```
xpcs_webplot serve ./html --port 8080
```

4.5.2.2 `--host` (default: `0.0.0.0`)

Specify the host address to bind to:

```
# Localhost only
xpcs_webplot serve ./html --host 127.0.0.1

# All network interfaces (default)
xpcs_webplot serve ./html --host 0.0.0.0
```

4.5.2.3 `--debug` (default: `False`)

Enable Flask debug mode for development:

```
xpcs_webplot serve ./html --debug
```

Warning: Never use debug mode in production!

4.5.3 Features

The web server provides:

- **Interactive Index**: Browse all results with filtering and sorting
- **Subdirectory Support**: Navigate through nested result structures
- **Combined Views**: View multiple results in a single page
- **Direct File Access**: Access individual plots and data files
- **Responsive Design**: Works on desktop and mobile devices

4.5.4 Examples

```
xpcs_webplot serve ./html
```

4.5.4.1 Example 1: Basic Server

Access at `http://localhost:5000`

```
xpcs_webplot serve ./html --port 8080
```

4.5.4.2 Example 2: Custom Port

Access at `http://localhost:8080`

```
xpcs_webplot serve ./html --host 0.0.0.0 --port 80
```

4.5.4.3 Example 3: Network Access

Access from any computer on the network at `http://<your-ip-address>`

4.6 Configuration

4.6.1 Environment Variables

Currently, XPCS WebPlot does not use environment variables for configuration. All settings are passed via command-line arguments.

4.6.2 Default Settings

Default values for common options:

Option	Default	Description
`target-dir`	`./html`	Output directory
`num-img`	4	Number of twotime images
`dpi`	240	Plot resolution
`num-processes`	4	Parallel processes

Option	Default	Description
`port`	`5000`	Web server port
`host`	`0.0.0.0`	Web server host

4.7 Advanced Features

4.7.1 Real-Time Monitoring

The monitoring feature uses a producer-consumer architecture:

- **Producer**: Watches directory for new files using `watchdog`
- **Consumer**: Processes files from a queue using multiprocessing
- **Queue**: Thread-safe queue for file paths

```
xpcs_webplot plot /data/incoming --monitor --num-processes 4
```

4.7.2 Subdirectory Organization

The serve command supports hierarchical result organization:

```
html/
   experiment_A/
       sample_001/
       sample_002/
   experiment_B/
       sample_003/
       sample_004/
```

The web interface will: - Show subdirectories at the root level - Allow navigation into subdirectories - Display combined summaries for each subdirectory

4.7.3 Parallel Processing

Batch processing uses Python's multiprocessing:

```
# Use all available CPU cores
xpcs_webplot plot /data/*.hdf --num-processes $(nproc)
```

4.7.4 Error Handling

The converter includes robust error handling:

- Failed conversions are logged but don't stop batch processing
- Detailed error messages help diagnose issues
- Partial results are saved when possible

4.7.5 Output Structure

Each result folder contains:

```
result_folder/
   summary.html              # Main summary page
   metadata.json             # Extracted metadata
   images/                   # Optional subdirectory
       saxs.png
       stability.png
       g2_multitau.png
```

```
    twotime_*.png
c2_saxs.txt              # SAXS data
c2_g2.txt                # G2 correlation
c2_stability.txt         # Stability data
c2_twotime_*.txt         # Twotime data
c2_g2_partials.txt       # Partial G2 data
```

4.8 Tips and Best Practices

4.8.1 Performance Optimization

1. **Use parallel processing** for batch jobs:
   ```
   xpcs_webplot plot /data/*.hdf --num-processes 8
   ```
2. **Adjust DPI** based on needs:
   - Quick preview: `--dpi 120`
   - Standard: `--dpi 240` (default)
   - Publication: `--dpi 600`
3. **Use `--image-only`** when you don't need text exports:
   ```
   xpcs_webplot plot data.hdf --image-only
   ```

4.8.2 Workflow Recommendations

1. **Development**: Use `--debug` with serve command
2. **Production**: Use proper web server (nginx, Apache) instead of Flask dev server
3. **Archival**: Use high DPI and save all data
4. **Quick checks**: Use `--image-only` and lower DPI

4.8.3 Troubleshooting

Common issues and solutions:

1. **Out of memory**: Reduce `--num-processes`
2. **Slow processing**: Increase `--num-processes`
3. **Large output**: Use `--image-only` or `--no-save-result`
4. **Port in use**: Change `--port` to different value

4.9 See Also

- API Reference - Programmatic usage
- Architecture - System design
- FAQ - Common questions

5 API Reference

Complete API documentation for all modules in the XPCS WebPlot package.

5.1 Table of Contents

5.2 Module Overview

The XPCS WebPlot package consists of several modules:

Module	Purpose
`cli`	Command-line interface entry point
`converter`	Core conversion logic for XPCS files
`webplot_cli`	High-level conversion functions
`flask_app`	Flask web server for result viewing
`plot_images`	Plotting functions for XPCS data
`html_utlits`	HTML generation utilities
`monitor_and_process`	Directory monitoring and batch processing
`metadata_utils`	Metadata extraction and formatting

5.3 cli Module

Location: `src/xpcs_webplot/cli.py`

Main entry point for the command-line interface.

5.3.1 Functions

5.3.1.1 `main()` Main entry point for the XPCS webplot command-line interface.

Returns: - `int`: Exit code (0 for success, 1 for error)

Description:

Provides a command-line interface with three subcommands: - `plot`: Convert XPCS HDF files to web-viewable format - `combine`: Combine multiple result folders into unified index - `serve`: Start Flask web server for interactive browsing

Example:

```
import sys
from xpcs_webplot.cli import main

if __name__ == "__main__":
    sys.exit(main())
```

5.3.1.2 `run_flask_server(html_folder=".", port=5000, host="0.0.0.0")` Start Flask web server to serve XPCS webplot results.

Parameters: - `html_folder` (str): Path to directory containing HTML results - `port` (int): Port number for web server (default: 5000) - `host` (str): Host address to bind to (default: “0.0.0.0”)

Description:

Launches a Flask development server that provides a web interface for browsing and viewing XPCS analysis results.

Example:

```
from xpcs_webplot.cli import run_flask_server

run_flask_server('html', port=8080, host='127.0.0.1')
```

5.4 converter Module

Location: `src/xpcs_webplot/converter.py`

Core conversion logic for processing XPCS HDF5 files.

5.4.1 Functions

5.4.1.1 `convert_xpcs_result(fname=None, target_dir="html", num_img=4, dpi=240, overwrite=False, image_only=False, create_image_directory=True, save_result=True)` Convert XPCS HDF file to web-viewable format with plots and data exports.

Parameters: - `fname` (str or Path): Path to input XPCS HDF file - `target_dir` (str or Path): Output directory for results (default: "html") - `num_img` (int): Number of twotime correlation images (default: 4) - `dpi` (int): Resolution for plots in dots per inch (default: 240) - `overwrite` (bool): Overwrite existing results (default: False) - `image_only` (bool): Generate only images, skip data export (default: False) - `create_image_directory` (bool): Create 'images' subdirectory (default: True) - `save_result` (bool): Save data to text files (default: True)

Returns: - `bool`: True if conversion successful, False otherwise

Raises: - `FileNotFoundError`: If input file doesn't exist - `ValueError`: If file format is invalid

Example:

```
from xpcs_webplot.converter import convert_xpcs_result

# Basic conversion
convert_xpcs_result('data.hdf', target_dir='output')

# High-quality output
convert_xpcs_result('data.hdf', dpi=600, num_img=12)

# Images only
convert_xpcs_result('data.hdf', image_only=True)
```

5.4.1.2 `save_xpcs_result(xf_obj, top_dir)` Save XPCS analysis results to text and image files.

Parameters: - `xf_obj` (XpcsFile): XPCS file object from pyxpcsviewer - `top_dir` (Path): Output directory for saved files

Description:

Exports SAXS data, correlation functions, and twotime analysis results from an XPCS file object to a structured directory format.

Output Files: - `c2_saxs.txt`: SAXS intensity data - `c2_g2.txt`: G2 correlation function - `c2_stability.txt`: Stability analysis data - `c2_twotime_*.txt`: Twotime correlation maps - `c2_g2_partials.txt`: Segmented g2 from twotime analysis

Example:

```
from pyxpcsviewer.xpcs_file import XpcsFile
from xpcs_webplot.converter import save_xpcs_result
from pathlib import Path
```

```
xf = XpcsFile('data.hdf')
save_xpcs_result(xf, Path('output'))
```

5.4.1.3 `plot_xpcs_result(xf_obj, top_dir, num_img=4, dpi=240, image_only=False)`

Generate plots and HTML summary from XPCS file object.

Parameters: - `xf_obj` (XpcsFile): XPCS file object - `top_dir` (Path): Output directory - `num_img` (int): Number of twotime images (default: 4) - `dpi` (int): Plot resolution (default: 240) - `image_only` (bool): Skip HTML generation (default: False)

Description:

Creates visualization plots for SAXS patterns, stability analysis, and correlation functions, then generates an HTML summary page.

Example:

```
from pyxpcsviewer.xpcs_file import XpcsFile
from xpcs_webplot.converter import plot_xpcs_result
from pathlib import Path

xf = XpcsFile('data.hdf')
plot_xpcs_result(xf, Path('output'), num_img=8, dpi=300)
```

5.4.1.4 `convert_xpcs_result_safe(*args, **kwargs)`

Safe wrapper for convert_xpcs_result with exception handling.

Parameters: - `*args`: Positional arguments for convert_xpcs_result - `**kwargs`: Keyword arguments for convert_xpcs_result

Returns: - `bool`: True if successful, False if exception occurred

Description:

Catches and logs any exceptions that occur during conversion, preventing crashes in batch processing scenarios.

Example:

```
from xpcs_webplot.converter import convert_xpcs_result_safe

# Safe conversion that won't crash on errors
success = convert_xpcs_result_safe('data.hdf', target_dir='output')
if not success:
    print("Conversion failed, but program continues")
```

5.5 webplot_cli Module

Location: `src/xpcs_webplot/webplot_cli.py`

High-level functions for file conversion workflows.

5.5.1 Functions

5.5.1.1 `convert_one_file(fname, target_dir="html", num_img=4, dpi=240, overwrite=False, image_only=False, create_image_directory=True, save_result=True)`

Convert a single XPCS HDF file to web format.

Parameters: - Same as `convert_xpcs_result`

Returns: - `bool`: True if successful

Example:

```
from xpcs_webplot.webplot_cli import convert_one_file

convert_one_file('experiment_001.hdf', target_dir='results')
```

5.5.1.2 `convert_many_files(file_list, target_dir="html", num_img=4, dpi=240, overwrite=False, image_only=False, create_image_directory=True, save_result=True, num_processes=4)`

Convert multiple XPCS files in parallel.

Parameters: - `file_list` (list): List of file paths to convert - `num_processes` (int): Number of parallel processes (default: 4) - Other parameters same as `convert_one_file`

Returns: - `list`: List of boolean results for each file

Example:

```
from xpcs_webplot.webplot_cli import convert_many_files
import glob

files = glob.glob('/data/*.hdf')
results = convert_many_files(files, num_processes=8)
print(f"Successful: {sum(results)}/{len(results)}")
```

5.6 flask_app Module

Location: `src/xpcs_webplot/flask_app.py`

Flask web application for serving and viewing XPCS results.

5.6.1 Functions

5.6.1.1 `create_app(html_folder='html')`

Factory function to create Flask app with custom HTML folder.

Parameters: - `html_folder` (str): Path to HTML results directory

Returns: - `Flask`: Configured Flask application instance

Example:

```
from xpcs_webplot.flask_app import create_app

app = create_app('my_results')
app.run(host='0.0.0.0', port=5000)
```

5.6.1.2 `get_html_data(html_folder, subdir=None)`

Extract metadata from all HTML folders.

Parameters: - `html_folder` (str): Base HTML folder path - `subdir` (str, optional): Subdirectory to scan within html_folder

Returns: - `dict`: Dictionary with 'type', 'subdirs', and 'results' keys

Example:

```
from xpcs_webplot.flask_app import get_html_data

data = get_html_data('html')
print(f"Found {len(data['results'])} results")
```

5.6.1.3 `get_subdirectories(html_folder)`

Get list of subdirectories that contain result folders.

Parameters: - `html_folder` (str): Base HTML folder path

Returns: - `list`: List of subdirectory names

Example:

```
from xpcs_webplot.flask_app import get_subdirectories

subdirs = get_subdirectories('html')
for subdir in subdirs:
    print(f"Subdirectory: {subdir}")
```

5.7 plot_images Module

Location: `src/xpcs_webplot/plot_images.py`

Functions for generating plots from XPCS data.

5.7.1 Functions

5.7.1.1 `plot_crop_mask_saxs(xf_obj, top_dir, dpi=240)`

Plot SAXS pattern with crop mask overlay.

Parameters: - `xf_obj` (XpcsFile): XPCS file object - `top_dir` (Path): Output directory - `dpi` (int): Plot resolution

Returns: - `str`: Path to saved plot file

5.7.1.2 `plot_stability(xf_obj, top_dir, dpi=240)`

Plot stability analysis (intensity vs frame number).

Parameters: - `xf_obj` (XpcsFile): XPCS file object - `top_dir` (Path): Output directory - `dpi` (int): Plot resolution

Returns: - `str`: Path to saved plot file

5.7.1.3 `plot_multitau_correlation(xf_obj, top_dir, dpi=240)`

Plot multitau correlation functions (g2 vs lag time).

Parameters: - `xf_obj` (XpcsFile): XPCS file object - `top_dir` (Path): Output directory - `dpi` (int): Plot resolution

Returns: - `str`: Path to saved plot file

5.7.1.4 `plot_twotime_correlation(xf_obj, top_dir, num_img=4, dpi=240)`

Plot twotime correlation maps.

Parameters: - `xf_obj` (XpcsFile): XPCS file object - `top_dir` (Path): Output directory - `num_img` (int): Number of images to generate - `dpi` (int): Plot resolution

Returns: - `list`: List of paths to saved plot files

5.8 html_utlits Module

Location: `src/xpcs_webplot/html_utlits.py`

Utilities for HTML generation and manipulation.

5.8.1 Functions

5.8.1.1 `convert_to_html(xf_obj, top_dir, image_dir=None)` Generate HTML summary page from XPCS file object.

Parameters: - `xf_obj` (XpcsFile): XPCS file object - `top_dir` (Path): Output directory - `image_dir` (str, optional): Subdirectory for images

Returns: - `str`: Path to generated HTML file

5.8.1.2 `combine_all_htmls(html_folder, output_file='combined_summary.html')` Combine multiple result folders into unified index page.

Parameters: - `html_folder` (str): Directory containing result folders - `output_file` (str): Output filename

Returns: - `str`: Path to generated combined HTML file

5.9 monitor_and_process Module

Location: `src/xpcs_webplot/monitor_and_process.py`

Directory monitoring and batch processing with producer-consumer pattern.

5.9.1 Functions

5.9.1.1 `monitor_and_process(input_path, target_dir="html", num_img=4, dpi=240, overwrite=False, image_only=False, create_image_directory=True, save_result=True, num_processes=4)` Monitor directory for new XPCS files and process them automatically.

Parameters: - `input_path` (str): Directory to monitor - `num_processes` (int): Number of worker processes - Other parameters same as convert_xpcs_result

Description:

Uses watchdog to monitor a directory for new HDF files and processes them using a producer-consumer pattern with multiprocessing.

Example:

```
from xpcs_webplot.monitor_and_process import monitor_and_process

# Monitor directory and auto-process new files
monitor_and_process('/data/incoming', target_dir='results', num_processes=4)
```

5.10 metadata_utils Module

Location: `src/xpcs_webplot/metadata_utils.py`

Utilities for extracting and formatting metadata.

5.10.1 Functions

5.10.1.1 `save_metadata(xf_obj, top_dir)` Extract and save metadata from XPCS file to JSON.

Parameters: - `xf_obj` (XpcsFile): XPCS file object - `top_dir` (Path): Output directory

Returns: - `dict`: Extracted metadata dictionary

5.10.1.2 `convert_to_native_format(obj)`

Convert numpy/HDF5 types to native Python types for JSON serialization.

Parameters: - `obj`: Object to convert

Returns: - Native Python type equivalent

5.11 Usage Examples

5.11.1 Example 1: Programmatic Conversion

```
from xpcs_webplot.converter import convert_xpcs_result
from pathlib import Path

# Convert single file
success = convert_xpcs_result(
    fname='experiment_001.hdf',
    target_dir='results',
    num_img=8,
    dpi=300,
    overwrite=True
)

if success:
    print("Conversion successful!")
```

5.11.2 Example 2: Batch Processing

```
from xpcs_webplot.webplot_cli import convert_many_files
import glob

# Get all HDF files
files = glob.glob('/data/experiment/*.hdf')

# Process in parallel
results = convert_many_files(
    files,
    target_dir='batch_results',
    num_processes=8,
    dpi=240
)

print(f"Processed {sum(results)}/{len(results)} files successfully")
```

5.11.3 Example 3: Custom Flask App

```
from xpcs_webplot.flask_app import create_app

# Create custom app
app = create_app('my_results')

# Add custom route
@app.route('/custom')
def custom_page():
```

```
    return "Custom page"

# Run server
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=False)
```

5.11.4 Example 4: Direct Plotting

```
from pyxpcsviewer.xpcs_file import XpcsFile
from xpcs_webplot.plot_images import (
    plot_crop_mask_saxs,
    plot_stability,
    plot_multitau_correlation
)
from pathlib import Path

# Load XPCS file
xf = XpcsFile('data.hdf')
output_dir = Path('plots')
output_dir.mkdir(exist_ok=True)

# Generate individual plots
plot_crop_mask_saxs(xf, output_dir, dpi=300)
plot_stability(xf, output_dir, dpi=300)
plot_multitau_correlation(xf, output_dir, dpi=300)
```

5.12 See Also

- User Guide - Command-line usage
- Architecture - System design
- Development Guide - Contributing

6 Architecture

This document describes the system architecture and design decisions for XPCS WebPlot.

6.1 Table of Contents

6.2 Overview

XPCS WebPlot is designed as a modular Python package for converting X-ray Photon Correlation Spectroscopy (XPCS) analysis results into web-viewable formats. The architecture emphasizes:

- **Modularity**: Clear separation of concerns across modules
- **Scalability**: Parallel processing for batch operations

- **Flexibility**: Multiple interfaces (CLI, programmatic, web)
- **Robustness**: Error handling and recovery mechanisms

6.3 System Architecture

6.3.1 High-Level Architecture

```
                    User Interfaces

  CLI (cli.py)     Web (flask_app)    Programmatic (API)
```

```
                Core Processing Layer

        • converter.py (conversion logic)
        • webplot_cli.py (workflow mgmt)
        • monitor_and_process.py (watching)
```

```
                Visualization Layer

        • plot_images.py (plotting)
        • html_utlits.py (HTML generation)
        • metadata_utils.py (data extraction)
```

```
                Data Layer

        • pyxpcsviewer (HDF5 reading)
        • File system (output storage)
```

6.3.2 Component Diagram

```
                        XPCS WebPlot


     CLI            Flask App         Monitoring
  Interface          (Web UI)          Service
```

```
          Conversion Orchestrator
          (webplot_cli.py)
```

```
        Core Converter
        (converter.py)


  Plotting         HTML/Metadata
  Engine            Generator
```

6.4 Module Organization

6.4.1 Core Modules

6.4.1.1 cli.py

- **Purpose**: Command-line interface entry point
- **Responsibilities**:
  - Argument parsing with argparse
  - Subcommand routing (plot, combine, serve)
  - User input validation
  - Error reporting

6.4.1.2 converter.py

- **Purpose**: Core conversion logic
- **Responsibilities**:
  - HDF5 file reading via pyxpcsviewer
  - Data extraction and transformation
  - Orchestrating plot generation
  - File output management
  - Error handling and recovery

6.4.1.3 webplot_cli.py

- **Purpose**: High-level workflow management
- **Responsibilities**:
  - Single file conversion workflow
  - Batch processing coordination
  - Multiprocessing pool management
  - Progress tracking

6.4.2 Visualization Modules

6.4.2.1 plot_images.py

- **Purpose**: Generate plots from XPCS data
- **Responsibilities**:
  - SAXS pattern visualization
  - Stability analysis plots
  - Correlation function plots
  - Twotime correlation maps
  - Plot styling and formatting

6.4.2.2 html_utlits.py

- **Purpose**: HTML generation and manipulation
- **Responsibilities**:
  - Jinja2 template rendering
  - Summary page generation
  - Combined index creation
  - Metadata embedding

6.4.2.3 metadata_utils.py

- **Purpose**: Metadata extraction and formatting
- **Responsibilities**:
  - Extract metadata from HDF5 files
  - Convert to JSON-serializable format
  - Type conversion (numpy → Python)
  - Metadata validation

6.4.3 Service Modules

6.4.3.1 flask_app.py

- **Purpose**: Web server for result viewing
- **Responsibilities**:
  - Flask application factory
  - Route definitions
  - Template rendering
  - Static file serving
  - Subdirectory navigation
  - API endpoints

6.4.3.2 monitor_and_process.py

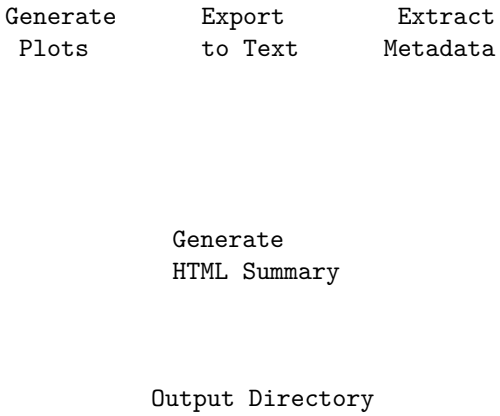
- **Purpose**: Directory monitoring and auto-processing
- **Responsibilities**:
  - File system watching (watchdog)
  - Producer-consumer queue management
  - Multiprocessing coordination
  - Event handling

6.5 Data Flow

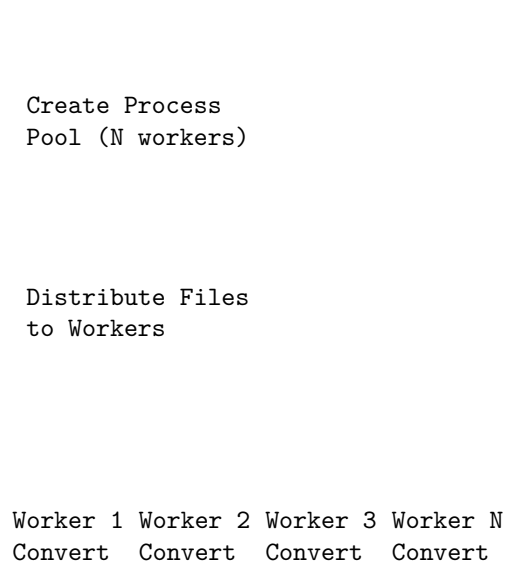
6.5.1 Conversion Pipeline

```
Input HDF5 File


  Load with
  pyxpcsviewer



  Extract Data
  • SAXS
  • Stability
```

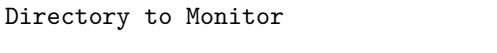
- Correlation
- Twotime

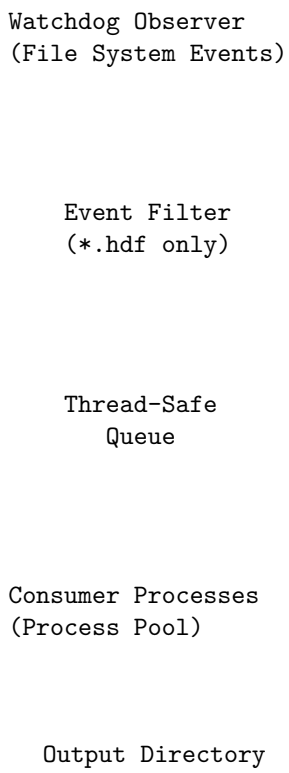


6.5.2 Batch Processing Flow



6.5.3 Monitoring Flow





6.6 Design Patterns

6.6.1 Factory Pattern

Used in `flask_app.py` for creating Flask applications:

```
def create_app(html_folder='html'):
    """Factory function to create Flask app"""
    app = Flask(__name__)
    # Configure app with html_folder
    # Register routes
    return app
```

Benefits: - Allows multiple app instances with different configurations - Facilitates testing with different settings - Enables dynamic configuration

6.6.2 Producer-Consumer Pattern

Used in `monitor_and_process.py` for file monitoring:

```
# Producer: Watchdog observer adds files to queue
# Consumer: Process pool workers take files from queue

queue = Queue()

# Producer thread
observer.schedule(handler, path)
handler.on_created = lambda event: queue.put(event.src_path)
```

```
# Consumer processes
with Pool(num_processes) as pool:
    while True:
        file_path = queue.get()
        pool.apply_async(convert_file, (file_path,))
```

Benefits: - Decouples file detection from processing - Enables parallel processing - Handles variable processing times

6.6.3 Wrapper Pattern

Used for error handling in `converter.py`:

```
def convert_xpcs_result_safe(*args, **kwargs):
    """Safe wrapper with exception handling"""
    try:
        return convert_xpcs_result(*args, **kwargs)
    except Exception as e:
        logger.error(f"Conversion failed: {e}")
        return False
```

Benefits: - Prevents crashes in batch processing - Centralizes error handling - Maintains consistent interface

6.6.4 Template Method Pattern

Used in conversion workflow:

```
def convert_xpcs_result(fname, **options):
    # Template method defining conversion steps
    xf = load_file(fname)
    save_data(xf)      # Optional step
    generate_plots(xf)  # Required step
    create_html(xf)     # Optional step
```

Benefits: - Defines standard workflow - Allows customization via options - Ensures consistency

6.7 Technology Stack

6.7.1 Core Dependencies

Package	Purpose	Version
Python	Runtime	3.8
numpy	Numerical operations	Latest
h5py	HDF5 file I/O	Latest
matplotlib	Plotting	Latest
jinja2	HTML templating	Latest
flask	Web framework	Latest
watchdog	File system monitoring	Latest
pyxpcsviewer	XPCS file reading	Latest

6.7.2 Development Dependencies

Package	Purpose
pytest	Testing framework
pytest-cov	Coverage reporting
black	Code formatting
flake8	Linting
mypy	Type checking

6.7.3 Build System

- **Build backend**: setuptools
- **Configuration**: pyproject.toml (PEP 517/518)
- **Package structure**: src layout

6.8 Design Decisions

6.8.1 Why src Layout?

Decision: Use `src/xpcs_webplot/` instead of `xpcs_webplot/`

Rationale: - Prevents accidental imports of uninstalled package - Ensures tests run against installed version - Follows modern Python packaging best practices - Better isolation between source and installed code

6.8.2 Why pyproject.toml?

Decision: Use `pyproject.toml` instead of `setup.py`

Rationale: - Modern standard (PEP 517/518) - Single configuration file for all tools - Better dependency resolution - Declarative instead of imperative

6.8.3 Why Multiprocessing?

Decision: Use multiprocessing instead of threading

Rationale: - CPU-bound operations (plotting, data processing) - Bypasses Python GIL limitations - Better performance on multi-core systems - Process isolation for robustness

6.8.4 Why Flask Development Server?

Decision: Include Flask dev server, recommend nginx for production

Rationale: - Easy setup for development and testing - Good enough for small-scale deployments - Clear upgrade path to production servers - Familiar to Python developers

6.8.5 Why Jinja2 Templates?

Decision: Use Jinja2 for HTML generation

Rationale: - Powerful templating with inheritance - Well-integrated with Flask - Separates presentation from logic - Widely used and documented

6.8.6 Why Watchdog for Monitoring?

Decision: Use watchdog library for file system monitoring

Rationale: - Cross-platform compatibility - Efficient event-based monitoring - Well-maintained and reliable - Simple API

6.8.7 Why NumPy Docstrings?

Decision: Use NumPy-style docstrings

Rationale: - Standard in scientific Python community - Well-structured and readable - Supported by documentation generators - Familiar to target users

6.9 Performance Considerations

6.9.1 Parallel Processing

- Default: 4 processes
- Configurable via `--num-processes`
- Optimal value depends on:
  - Number of CPU cores
  - Available memory
  - I/O characteristics

6.9.2 Memory Management

- Process files one at a time in each worker
- Close HDF5 files after reading
- Clear matplotlib figures after saving
- Avoid loading entire datasets into memory

6.9.3 I/O Optimization

- Batch file operations where possible
- Use buffered I/O for text exports
- Minimize HDF5 file reopening
- Cache metadata when appropriate

6.10 Extensibility

6.10.1 Adding New Plot Types

1. Add function to `plot_images.py`
2. Call from `plot_xpcs_result()` in `converter.py`
3. Update HTML template to display new plot

6.10.2 Adding New Export Formats

1. Add export function to `converter.py`
2. Call from `save_xpcs_result()`
3. Update documentation

6.10.3 Adding New CLI Commands

1. Add subparser in `cli.py`
2. Implement handler function
3. Update help text and documentation

6.10.4 Customizing Web Interface

1. Modify templates in `src/xpcs_webplot/templates/`
2. Add routes in `flask_app.py`
3. Update static assets if needed

6.11 Security Considerations

6.11.1 File System Access

- Validate all file paths
- Prevent directory traversal attacks
- Use absolute paths internally
- Sanitize user inputs

6.11.2 Web Server

- Don't use Flask dev server in production
- Validate all user inputs
- Sanitize file paths in routes
- Use proper HTTP headers

6.11.3 Data Privacy

- No authentication/authorization built-in
- Assumes trusted network environment
- Consider adding auth for production use
- Be careful with sensitive data

6.12 Future Enhancements

6.12.1 Planned Features

1. **Database Backend**: Store metadata in SQLite/PostgreSQL
2. **REST API**: Programmatic access to results
3. **Real-time Updates**: WebSocket support for live monitoring
4. **Advanced Filtering**: More sophisticated result filtering
5. **Export Formats**: PDF reports, CSV exports
6. **Comparison Tools**: Side-by-side result comparison

6.12.2 Architectural Improvements

1. **Plugin System**: Allow third-party extensions
2. **Caching Layer**: Cache expensive computations
3. **Async Processing**: Use asyncio for I/O operations
4. **Configuration Files**: Support config files for defaults
5. **Logging Improvements**: Structured logging with levels

6.13 See Also

- API Reference - Detailed API documentation
- Development Guide - Contributing guidelines
- User Guide - Usage instructions

7 Development Guide

Guide for developers who want to contribute to XPCS WebPlot.

7.1 Table of Contents

- Code Style
- Testing
- Contributing
- Release Process

7.2 Getting Started

7.2.1 Prerequisites

- Python 3.8 or higher
- Git
- Basic understanding of:
  - XPCS data analysis
  - HDF5 file format
  - Python packaging
  - Web development (for Flask components)

7.2.2 Fork and Clone

```
# Fork the repository on GitHub first, then:
git clone https://github.com/YOUR_USERNAME/xpcs_webplot.git
cd xpcs_webplot
```

7.3 Development Setup

7.3.1 Install in Development Mode

```
# Create virtual environment (recommended)
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install package with development dependencies
pip install -e ".[dev]"
```

This installs: - The package in editable mode (changes reflected immediately) - All runtime dependencies - Development tools (pytest, black, flake8, mypy, pre-commit)

7.3.2 Verify Installation

```
# Check that commands are available
xpcs_webplot --help
xpcs_webplot_server --help

# Run tests to verify setup
pytest
```

7.3.3 Project Structure

```
xpcs_webplot/
  src/
      xpcs_webplot/          # Main package
          __init__.py
          cli.py             # CLI entry point
          converter.py       # Core conversion logic
```

```
        webplot_cli.py     # Workflow management
        flask_app.py       # Web server
        plot_images.py     # Plotting functions
        html_utlits.py     # HTML generation
        monitor_and_process.py  # Monitoring
        metadata_utils.py  # Metadata handling
        templates/         # Jinja2 templates
            flask_index.html
            flask_result.html
            mini-preview/
tests/                      # Test files
    test_xpcs_webplot.py
    test_flask_server.py
docs/                       # Documentation
pyproject.toml             # Project configuration
README.rst                 # Project README
LICENSE                    # MIT License
Makefile                   # Development commands
```

7.4 Code Style

7.4.1 Python Style Guide

We follow PEP 8 with some modifications:

- **Line length**: 88 characters (Black default)
- **Docstrings**: NumPy style
- **Imports**: Sorted with isort (Black-compatible profile)
- **Type hints**: Encouraged but not required

7.4.2 Formatting with Black

```
# Format all code
black src tests

# Check without modifying
black --check src tests
```

7.4.3 Linting with Flake8

```
# Check code style
flake8 src tests

# Common issues to avoid:
# - Unused imports
# - Undefined names
# - Line too long (>88 chars)
```

7.4.4 Import Sorting with isort

```
# Sort imports
isort src tests
```

```
# Check without modifying
isort --check src tests
```

7.4.5 Type Checking with mypy

```
# Run type checker
mypy src

# Note: Type hints are encouraged but not strictly enforced
```

7.4.6 Pre-commit Hooks

Set up pre-commit hooks to automatically check code before committing:

```
# Install pre-commit hooks
pre-commit install

# Run manually on all files
pre-commit run --all-files
```

7.4.7 Docstring Style

Use NumPy-style docstrings for all public functions and classes:

```
def example_function(param1, param2, optional_param=None):
    """
    Brief description of function.

    Longer description with more details about what the function does,
    how it works, and any important considerations.

    Parameters
    ----------
    param1 : str
        Description of param1
    param2 : int
        Description of param2
    optional_param : bool, optional
        Description of optional parameter (default: None)

    Returns
    -------
    result_type
        Description of return value

    Raises
    ------
    ValueError
        When param1 is invalid
    FileNotFoundError
        When file doesn't exist

    See Also
    --------
```

```
    related_function : Related functionality

    Examples
    --------
    >>> example_function('test', 42)
    'result'

    Notes
    -----
    Additional notes about implementation details or usage.
    """
    pass
```

7.5 Testing

7.5.1 Running Tests

```
# Run all tests
pytest

# Run with coverage
pytest --cov=xpcs_webplot --cov-report=term-missing

# Run specific test file
pytest tests/test_xpcs_webplot.py

# Run specific test function
pytest tests/test_xpcs_webplot.py::test_specific_function

# Run with verbose output
pytest -v
```

7.5.2 Writing Tests

Place tests in the `tests/` directory:

```
# tests/test_converter.py
import pytest
from pathlib import Path
from xpcs_webplot.converter import convert_xpcs_result

def test_convert_xpcs_result_basic():
    """Test basic conversion functionality."""
    # Arrange
    input_file = Path('test_data/sample.hdf')
    output_dir = Path('test_output')

    # Act
    result = convert_xpcs_result(input_file, target_dir=output_dir)

    # Assert
    assert result is True
    assert (output_dir / 'sample' / 'summary.html').exists()
```

```python
def test_convert_xpcs_result_invalid_file():
    """Test handling of invalid input file."""
    with pytest.raises(FileNotFoundError):
        convert_xpcs_result('nonexistent.hdf')
```

7.5.3 Test Coverage

Aim for: - **Overall coverage**: >80% - **Critical paths**: 100% - **Error handling**: Well-tested

Check coverage:

```
# Generate coverage report
pytest --cov=xpcs_webplot --cov-report=html

# View in browser
open htmlcov/index.html
```

7.5.4 Test Data

- Store test data in `test_data/` directory
- Use small, representative HDF5 files
- Document test data sources and formats
- Don't commit large files (use Git LFS if needed)

7.6 Contributing

7.6.1 Workflow

1. **Create a branch** for your feature or fix:
   ```
   git checkout -b feature/your-feature-name
   ```
2. **Make changes** following code style guidelines
3. **Add tests** for new functionality
4. **Run tests** to ensure nothing breaks:
   ```
   pytest
   black src tests
   flake8 src tests
   ```
5. **Commit changes** with clear messages:
   ```
   git add .
   git commit -m "Add feature: description of changes"
   ```
6. **Push to your fork**:
   ```
   git push origin feature/your-feature-name
   ```
7. **Create Pull Request** on GitHub

7.6.2 Commit Messages

Follow conventional commit format:

```
type(scope): brief description

Longer description if needed, explaining what changed and why.
```

```
Fixes #123
```

Types: - `feat`: New feature - `fix`: Bug fix - `docs`: Documentation changes - `style`: Code style changes (formatting, etc.) - `refactor`: Code refactoring - `test`: Adding or updating tests - `chore`: Maintenance tasks

Examples:

```
feat(converter): add support for new HDF5 format

fix(flask_app): correct subdirectory navigation bug

docs(api): update API reference for converter module

test(converter): add tests for error handling
```

7.6.3 Pull Request Guidelines

- **Title**: Clear, descriptive title
- **Description**: Explain what changed and why
- **Tests**: Include tests for new functionality
- **Documentation**: Update docs if needed
- **Code style**: Ensure all checks pass
- **Small PRs**: Keep changes focused and manageable

7.6.4 Code Review

All contributions require code review:

- Address reviewer feedback promptly
- Be open to suggestions
- Explain your design decisions
- Update PR based on feedback

7.7 Development Tasks

7.7.1 Common Tasks

Use the Makefile for common development tasks:

```
# Run tests
make test

# Format code
make format

# Lint code
make lint

# Type check
make typecheck

# Clean build artifacts
make clean

# Build package
make build
```

```
# Install in development mode
make install-dev
```

7.7.2 Adding New Features

1. **Plan the feature**:
   - Write design document if complex
   - Discuss in GitHub issue
   - Get feedback from maintainers
2. **Implement**:
   - Write code following style guide
   - Add comprehensive docstrings
   - Include type hints where appropriate
3. **Test**:
   - Write unit tests
   - Write integration tests if needed
   - Ensure good coverage
4. **Document**:
   - Update API reference
   - Update user guide if user-facing
   - Add examples
5. **Submit PR**:
   - Follow PR guidelines
   - Link to related issues
   - Request review

7.7.3 Fixing Bugs

1. **Reproduce the bug**:
   - Create minimal test case
   - Document steps to reproduce
2. **Write failing test**:
   - Test should fail with current code
   - Test should pass after fix
3. **Fix the bug**:
   - Make minimal changes
   - Don't introduce new features
4. **Verify fix**:
   - Ensure test passes
   - Check for regressions
   - Test edge cases
5. **Submit PR**:
   - Reference issue number
   - Explain root cause
   - Describe solution

7.8 Release Process

7.8.1 Version Numbering

Follow Semantic Versioning (SemVer):

- **MAJOR**: Incompatible API changes
- **MINOR**: New functionality, backwards-compatible
- **PATCH**: Bug fixes, backwards-compatible

Examples: - `0.0.1` → `0.0.2`: Bug fix - `0.0.2` → `0.1.0`: New feature - `0.1.0` → `1.0.0`: Breaking change

7.8.2 Release Checklist

1. **Update version** in `pyproject.toml`
2. **Update CHANGELOG**:
   - List all changes since last release
   - Categorize: Added, Changed, Fixed, Removed
3. **Run full test suite**:
   ```
   pytest
   black --check src tests
   flake8 src tests
   mypy src
   ```
4. **Build package**:
   ```
   python -m build
   ```
5. **Test installation**:
   ```
   pip install dist/xpcs_webplot-*.whl
   xpcs_webplot --help
   ```
6. **Create git tag**:
   ```
   git tag -a v0.1.0 -m "Release version 0.1.0"
   git push origin v0.1.0
   ```
7. **Upload to PyPI**:
   ```
   python -m twine upload dist/*
   ```
8. **Create GitHub release**:
   - Go to GitHub releases
   - Create new release from tag
   - Add release notes

7.9 Debugging

7.9.1 Logging

Enable debug logging:

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

7.9.2 Interactive Debugging

Use Python debugger:

```
import pdb; pdb.set_trace()
```

Or use IPython debugger:

```
from IPython import embed; embed()
```

7.9.3 Common Issues

Import errors after changes:

```
# Reinstall in editable mode
pip install -e .
```

Tests failing unexpectedly:

```
# Clear pytest cache
pytest --cache-clear
```

Flask app not updating:

```
# Restart with debug mode
xpcs_webplot serve --debug
```

7.10 Resources

7.10.1 Documentation

- Python Packaging Guide
- NumPy Docstring Guide
- Flask Documentation
- pytest Documentation

7.10.2 Tools

- Black
- Flake8
- mypy
- isort

7.11 Getting Help

- **GitHub Issues**: Report bugs or request features
- **Discussions**: Ask questions or share ideas
- **Email**: Contact maintainers directly

7.12 See Also

- User Guide - Using the package
- API Reference - API documentation
- Architecture - System design

8 Deployment Guide

Guide for deploying XPCS WebPlot in production environments.

8.1 Table of Contents

- Performance Tuning

8.2 Overview

This guide covers deploying XPCS WebPlot for production use, including:

- Web server configuration (nginx, Apache)
- Process management
- Docker containerization
- Security best practices
- Performance optimization
- Monitoring and logging

8.3 Production Deployment

8.3.1 Architecture

For production, use a proper web server instead of Flask's development server:

```
   Client
  (Browser)




   Nginx        (Reverse Proxy)
   Port 80




   Gunicorn     (WSGI Server)
   Port 5000




 Flask App      (XPCS WebPlot)
```

8.3.2 Prerequisites

- Linux server (Ubuntu 20.04+ recommended)
- Python 3.8+
- nginx or Apache
- Gunicorn or uWSGI
- systemd for process management

8.3.3 Installation on Server

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Python and dependencies
sudo apt install python3 python3-pip python3-venv nginx -y
```

```
# Create application user
sudo useradd -m -s /bin/bash xpcs
sudo su - xpcs

# Clone repository
git clone https://github.com/AZjk/xpcs_webplot.git
cd xpcs_webplot

# Create virtual environment
python3 -m venv venv
source venv/bin/activate

# Install application
pip install -e .
pip install gunicorn
```

8.3.4 Gunicorn Configuration

Create `gunicorn_config.py`:

```
# gunicorn_config.py
import multiprocessing

# Server socket
bind = "127.0.0.1:5000"
backlog = 2048

# Worker processes
workers = multiprocessing.cpu_count() * 2 + 1
worker_class = "sync"
worker_connections = 1000
timeout = 30
keepalive = 2

# Logging
accesslog = "/var/log/xpcs_webplot/access.log"
errorlog = "/var/log/xpcs_webplot/error.log"
loglevel = "info"

# Process naming
proc_name = "xpcs_webplot"

# Server mechanics
daemon = False
pidfile = "/var/run/xpcs_webplot/gunicorn.pid"
user = "xpcs"
group = "xpcs"
tmp_upload_dir = None
```

Create startup script `start_server.sh`:

```
#!/bin/bash
# start_server.sh
```

```
cd /home/xpcs/xpcs_webplot
source venv/bin/activate

# Set HTML folder location
export HTML_FOLDER=/data/xpcs/html

# Start Gunicorn
exec gunicorn \
    --config gunicorn_config.py \
    "xpcs_webplot.flask_app:create_app('$HTML_FOLDER')"
```

Make executable:

```
chmod +x start_server.sh
```

8.3.5 Systemd Service

Create `/etc/systemd/system/xpcs_webplot.service`:

```
[Unit]
Description=XPCS WebPlot Flask Application
After=network.target

[Service]
Type=notify
User=xpcs
Group=xpcs
WorkingDirectory=/home/xpcs/xpcs_webplot
Environment="PATH=/home/xpcs/xpcs_webplot/venv/bin"
Environment="HTML_FOLDER=/data/xpcs/html"
ExecStart=/home/xpcs/xpcs_webplot/start_server.sh
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=mixed
TimeoutStopSec=5
PrivateTmp=true
Restart=on-failure
RestartSec=10s

[Install]
WantedBy=multi-user.target
```

Enable and start service:

```
sudo systemctl daemon-reload
sudo systemctl enable xpcs_webplot
sudo systemctl start xpcs_webplot
sudo systemctl status xpcs_webplot
```

8.3.6 Nginx Configuration

Create `/etc/nginx/sites-available/xpcs_webplot`:

```
# Upstream to Gunicorn
upstream xpcs_webplot {
    server 127.0.0.1:5000 fail_timeout=0;
}
```

```
server {
    listen 80;
    server_name xpcs.example.com;

    # Increase client body size for large file uploads
    client_max_body_size 100M;

    # Logging
    access_log /var/log/nginx/xpcs_webplot_access.log;
    error_log /var/log/nginx/xpcs_webplot_error.log;

    # Root location
    location / {
        proxy_pass http://xpcs_webplot;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_redirect off;

        # Timeouts
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }

    # Static files (optional optimization)
    location /static {
        alias /data/xpcs/html;
        expires 30d;
        add_header Cache-Control "public, immutable";
    }
}
```

Enable site:

```
sudo ln -s /etc/nginx/sites-available/xpcs_webplot /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

8.3.7 SSL/TLS Configuration

Use Let's Encrypt for free SSL certificates:

```
# Install certbot
sudo apt install certbot python3-certbot-nginx -y

# Obtain certificate
sudo certbot --nginx -d xpcs.example.com

# Auto-renewal is configured automatically
```

Updated nginx config with SSL:

```
server {
    listen 80;
    server_name xpcs.example.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name xpcs.example.com;

    # SSL certificates
    ssl_certificate /etc/letsencrypt/live/xpcs.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/xpcs.example.com/privkey.pem;

    # SSL configuration
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;

    # Rest of configuration...
}
```

8.4 Docker Deployment

8.4.1 Dockerfile

Create `Dockerfile`:

```
FROM python:3.10-slim

# Set working directory
WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    && rm -rf /var/lib/apt/lists/*

# Copy application files
COPY . /app

# Install Python dependencies
RUN pip install --no-cache-dir -e .
RUN pip install --no-cache-dir gunicorn

# Create non-root user
RUN useradd -m -u 1000 xpcs && chown -R xpcs:xpcs /app
USER xpcs

# Expose port
EXPOSE 5000

# Set environment variables
```

```
ENV HTML_FOLDER=/data/html
ENV PYTHONUNBUFFERED=1

# Run application
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "--workers", "4", \
     "xpcs_webplot.flask_app:create_app('/data/html')"]
```

8.4.2 Docker Compose

Create `docker-compose.yml`:

```
version: '3.8'

services:
  xpcs_webplot:
    build: .
    container_name: xpcs_webplot
    restart: unless-stopped
    ports:
      - "5000:5000"
    volumes:
      - /data/xpcs/html:/data/html:ro
    environment:
      - HTML_FOLDER=/data/html
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  nginx:
    image: nginx:alpine
    container_name: xpcs_nginx
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - /data/xpcs/html:/data/html:ro
      - ./ssl:/etc/nginx/ssl:ro
    depends_on:
      - xpcs_webplot
```

Build and run:

```
docker-compose up -d
docker-compose logs -f
```

8.5 Server Configuration

8.5.1 Directory Structure

`/data/xpcs/`

```
  html/                    # Results directory
      experiment_001/
      experiment_002/
      ...
  logs/                    # Application logs
      access.log
      error.log
  backups/                 # Backup location
```

8.5.2 Permissions

```
# Create directories
sudo mkdir -p /data/xpcs/{html,logs,backups}

# Set ownership
sudo chown -R xpcs:xpcs /data/xpcs

# Set permissions
sudo chmod 755 /data/xpcs
sudo chmod 755 /data/xpcs/html
sudo chmod 755 /data/xpcs/logs
```

8.5.3 Log Rotation

Create /etc/logrotate.d/xpcs_webplot:

```
/var/log/xpcs_webplot/*.log {
    daily
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 xpcs xpcs
    sharedscripts
    postrotate
        systemctl reload xpcs_webplot > /dev/null 2>&1 || true
    endscript
}
```

8.6 Monitoring and Maintenance

8.6.1 Health Checks

Create health check endpoint in Flask app:

```
@app.route('/health')
def health_check():
    return {'status': 'healthy', 'timestamp': datetime.now().isoformat()}
```

8.6.2 Monitoring with systemd

```
# Check service status
sudo systemctl status xpcs_webplot

# View logs
```

```
sudo journalctl -u xpcs_webplot -f

# Restart service
sudo systemctl restart xpcs_webplot
```

8.6.3 Application Monitoring

Use tools like:

- **Prometheus**: Metrics collection
- **Grafana**: Visualization
- **Sentry**: Error tracking
- **New Relic**: APM

8.6.4 Backup Strategy

Automated backup script:

```
#!/bin/bash
# backup.sh

BACKUP_DIR="/data/xpcs/backups"
HTML_DIR="/data/xpcs/html"
DATE=$(date +%Y%m%d_%H%M%S)

# Create backup
tar -czf "$BACKUP_DIR/html_backup_$DATE.tar.gz" -C "$HTML_DIR" .

# Keep only last 30 days
find "$BACKUP_DIR" -name "html_backup_*.tar.gz" -mtime +30 -delete
```

Add to crontab:

```
# Run daily at 2 AM
0 2 * * * /home/xpcs/backup.sh
```

8.7 Security

8.7.1 Firewall Configuration

```
# Allow HTTP and HTTPS
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

# Allow SSH (if needed)
sudo ufw allow 22/tcp

# Enable firewall
sudo ufw enable
```

8.7.2 Application Security

1. **Run as non-root user**: Always use dedicated user
2. **Disable debug mode**: Never use in production
3. **Validate inputs**: Sanitize all user inputs
4. **Use HTTPS**: Always encrypt traffic

5. **Regular updates**: Keep dependencies updated

8.7.3 Access Control

If needed, add basic authentication in nginx:

```
location / {
    auth_basic "Restricted Access";
    auth_basic_user_file /etc/nginx/.htpasswd;
    proxy_pass http://xpcs_webplot;
}
```

Create password file:

```
sudo apt install apache2-utils
sudo htpasswd -c /etc/nginx/.htpasswd username
```

8.8 Performance Tuning

8.8.1 Gunicorn Workers

Calculate optimal workers:

```
workers = (2 × CPU_cores) + 1
```

For 4 CPU cores: 9 workers

8.8.2 Nginx Optimization

```
# Worker processes
worker_processes auto;

# Worker connections
events {
    worker_connections 1024;
    use epoll;
}

# Gzip compression
gzip on;
gzip_vary on;
gzip_min_length 1024;
gzip_types text/plain text/css application/json application/javascript;

# Caching
proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=my_cache:10m max_size=1g;
```

8.8.3 Database Optimization

If using database for metadata:

- Use connection pooling
- Add appropriate indexes
- Regular VACUUM (PostgreSQL)
- Monitor slow queries

8.8.4 File System

- Use SSD for better I/O
- Consider NFS for shared storage
- Regular cleanup of old results

8.9 Troubleshooting

8.9.1 Common Issues

Service won't start:

```
sudo journalctl -u xpcs_webplot -n 50
```

502 Bad Gateway: - Check Gunicorn is running - Verify nginx upstream configuration - Check firewall rules

Slow performance: - Increase Gunicorn workers - Enable nginx caching - Optimize database queries

Out of memory: - Reduce number of workers - Increase server RAM - Optimize image processing

8.10 See Also

- User Guide - Using the application
- Architecture - System design
- Development Guide - Contributing

9 Frequently Asked Questions (FAQ)

Common questions and troubleshooting tips for XPCS WebPlot.

9.1 Table of Contents

9.2 General Questions

9.2.1 What is XPCS WebPlot?

XPCS WebPlot is a Python package that converts X-ray Photon Correlation Spectroscopy (XPCS) analysis results from HDF5 files into interactive web-viewable formats with plots and data exports.

9.2.2 What file formats does it support?

XPCS WebPlot reads HDF5 files (.hdf, .h5) that follow the XPCS analysis output format from the pyxpcsviewer package.

9.2.3 Do I need to know Python to use it?

No! XPCS WebPlot provides a command-line interface that doesn't require Python programming knowledge. However, Python knowledge is helpful for advanced customization.

9.2.4 Is it free?

Yes, XPCS WebPlot is open-source software released under the MIT License. It's free to use, modify, and distribute.

9.2.5 Can I use it for commercial purposes?

Yes, the MIT License allows commercial use.

9.3 Installation Issues

9.3.1 I get "command not found" after installation

Problem: The `xpcs_webplot` command is not found.

Solution:

```
# Ensure the package is installed
pip list | grep xpcs_webplot

# If not installed, install it
pip install -e .

# Check if pip's bin directory is in PATH
which xpcs_webplot

# If not in PATH, add it (example for bash)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

9.3.2 Installation fails with "No module named 'setuptools'"

Problem: Missing build dependencies.

Solution:

```
pip install --upgrade pip setuptools wheel
pip install -e .
```

9.3.3 I get errors about missing dependencies

Problem: Required packages not installed.

Solution:

```
# Install with all dependencies
pip install -e ".[dev]"

# Or install specific missing package
pip install <package_name>
```

9.3.4 Installation fails on Windows

Problem: Some dependencies may have issues on Windows.

Solution: - Use Windows Subsystem for Linux (WSL) - Or use Anaconda/Miniconda: `bash conda create -n xpcs python=3.10 conda activate xpcs pip install -e .`

9.4 Usage Questions

9.4.1 How do I convert a single file?

```
xpcs_webplot plot your_file.hdf
```

The output will be in `./html/your_file/`

9.4.2 How do I process multiple files?

```
# Using wildcards
xpcs_webplot plot /path/to/data/*.hdf

# Or specify directory
xpcs_webplot plot /path/to/data/
```

9.4.3 How do I change the output directory?

```
xpcs_webplot plot input.hdf --target-dir /path/to/output
```

9.4.4 How do I view the results?

```
# Start web server
xpcs_webplot serve ./html

# Then open browser to http://localhost:5000
```

9.4.5 Can I process files automatically as they're created?

Yes! Use monitoring mode:

```
xpcs_webplot plot /data/incoming --monitor --target-dir ./results
```

This will watch the directory and process new files automatically.

9.4.6 How do I stop the monitoring mode?

Press `Ctrl+C` to stop the monitoring process.

9.4.7 Can I customize the plot resolution?

Yes, use the `--dpi` option:

```
# Higher resolution (better quality, larger files)
xpcs_webplot plot input.hdf --dpi 600

# Lower resolution (faster, smaller files)
xpcs_webplot plot input.hdf --dpi 120
```

9.4.8 How do I generate more twotime images?

Use the `--num-img` option:

```
xpcs_webplot plot input.hdf --num-img 12
```

9.4.9 Can I skip the data export and only generate images?

Yes, use the `--image-only` flag:

```
xpcs_webplot plot input.hdf --image-only
```

9.4.10 How do I overwrite existing results?

Use the `--overwrite` flag:

```
xpcs_webplot plot input.hdf --overwrite
```

9.5 Performance Issues

9.5.1 Processing is very slow

Possible causes and solutions:

1. **Too few processes**:
   ```
   # Increase parallel processes
   xpcs_webplot plot /data/*.hdf --num-processes 8
   ```
2. **High DPI setting**:
   ```
   # Reduce DPI for faster processing
   xpcs_webplot plot input.hdf --dpi 120
   ```
3. **Large files**:
   - This is expected for large datasets
   - Consider processing in smaller batches

9.5.2 The web server is slow

Solutions:

1. **Use production server** instead of Flask dev server:
   - See Deployment Guide
2. **Reduce number of results**:
   - Archive old results
   - Split into subdirectories
3. **Enable caching** in nginx (production)

9.5.3 I'm running out of memory

Solutions:

1. **Reduce parallel processes**:
   ```
   xpcs_webplot plot /data/*.hdf --num-processes 2
   ```
2. **Process files in smaller batches**
3. **Close other applications**
4. **Add more RAM to your system**

9.6 Error Messages

9.6.1 "FileNotFoundError: [Errno 2] No such file or directory"

Problem: Input file doesn't exist or path is wrong.

Solution: - Check file path is correct - Use absolute paths if relative paths don't work - Verify file exists: `ls -l /path/to/file.hdf`

9.6.2 "PermissionError: [Errno 13] Permission denied"

Problem: No permission to read input file or write to output directory.

Solution:

```
# Check permissions
ls -l input.hdf
ls -ld output_directory

# Fix permissions if needed
chmod 644 input.hdf
chmod 755 output_directory
```

9.6.3 "KeyError: 'some_key'"

Problem: HDF5 file is missing expected data fields.

Solution: - Verify file is a valid XPCS analysis output - Check file was created by compatible XPCS analysis software - Examine file structure: `h5ls -r file.hdf`

9.6.4 "OSError: Unable to open file"

Problem: HDF5 file is corrupted or in use.

Solution: - Verify file integrity - Ensure file is not being written to - Try copying file and processing the copy

9.6.5 "Address already in use"

Problem: Port is already in use by another application.

Solution:

```
# Use different port
xpcs_webplot serve ./html --port 8080

# Or find and stop process using the port
lsof -i :5000
kill <PID>
```

9.7 Web Server Issues

9.7.1 I can't access the web server from another computer

Problem: Server is bound to localhost only.

Solution:

```
# Bind to all interfaces
xpcs_webplot serve ./html --host 0.0.0.0

# Then access from other computer using server's IP
# http://<server-ip>:5000
```

Security Note: Only do this on trusted networks!

9.7.2 The web page shows "404 Not Found"

Problem: Results directory is empty or path is wrong.

Solution: - Verify HTML folder contains result folders - Check path is correct: `ls -l ./html` - Ensure you've run conversion first

9.7.3 Images don't load on the web page

Problem: Image paths are incorrect or files are missing.

Solution: - Check that image files exist in result folder - Verify file permissions: `ls -l html/result_folder/` - Try regenerating with `--overwrite`

9.7.4 The combined summary page is empty

Problem: No valid result folders found.

Solution: - Ensure result folders contain `summary.html` files - Check folder structure is correct - Run combine command again: `bash xpcs_webplot combine ./html`

9.8 Data and Output

9.8.1 Where is the output saved?

By default, output is saved to `./html/<filename>/` where `<filename>` is the input file name without extension.

9.8.2 What files are created?

For each input file, the following are created:

- `summary.html` - Main summary page
- `metadata.json` - Extracted metadata
- `*.png` - Plot images (SAXS, stability, correlation, twotime)
- `*.txt` - Exported data files (if not using `--image-only`)

9.8.3 Can I change the output structure?

The basic structure is fixed, but you can: - Change output directory with `--target-dir` - Skip image subdirectory with `--no-create-image-directory` - Skip data files with `--no-save-result`

9.8.4 How do I export data in different formats?

Currently, data is exported as tab-separated text files. For other formats:

1. Read the text files with your preferred tool
2. Convert using Python/pandas:

```
import pandas as pd
df = pd.read_csv('c2_g2.txt', sep='\t')
df.to_csv('output.csv')
df.to_excel('output.xlsx')
```

9.8.5 Can I customize the HTML templates?

Yes! Templates are in `src/xpcs_webplot/templates/`. You can:

1. Modify existing templates
2. Create custom templates

3. Use Jinja2 template inheritance

See Development Guide for details.

9.8.6 How do I delete old results?

```
# Delete specific result folder
rm -rf html/old_result/

# Delete all results
rm -rf html/*/

# Keep only recent results (older than 30 days)
find html/ -type d -mtime +30 -exec rm -rf {} +
```

9.9 Advanced Questions

9.9.1 Can I use this programmatically in my Python code?

Yes! See API Reference for details:

```
from xpcs_webplot.converter import convert_xpcs_result

convert_xpcs_result('data.hdf', target_dir='output')
```

9.9.2 Can I run this on a cluster?

Yes! You can:

1. Submit batch jobs for parallel processing
2. Use the monitoring mode on a head node
3. Deploy the web server for result viewing

9.9.3 How do I integrate this into my analysis pipeline?

```
# Example pipeline
from xpcs_webplot.converter import convert_xpcs_result

def my_pipeline(raw_data):
    # Your analysis code
    analyzed_file = run_xpcs_analysis(raw_data)

    # Convert to web format
    convert_xpcs_result(analyzed_file, target_dir='results')

    return analyzed_file
```

9.9.4 Can I add custom plots?

Yes! See Development Guide for instructions on:

1. Adding plot functions to `plot_images.py`
2. Calling them from `converter.py`
3. Updating HTML templates

9.9.5 How do I contribute to the project?

See Development Guide for:

- Setting up development environment
- Code style guidelines
- Testing requirements
- Pull request process

9.10 Still Need Help?

If your question isn't answered here:

1. **Check the documentation**:
   - User Guide
   - API Reference
   - Architecture
2. **Search GitHub Issues**:
   - Existing issues
   - Someone may have had the same problem
3. **Create a new issue**:
   - Provide detailed description
   - Include error messages
   - Share minimal example to reproduce
4. **Contact maintainers**:
   - See README for contact information

9.11 Common Workflows

9.11.1 Quick Preview Workflow

```
# Fast conversion for quick preview
xpcs_webplot plot data.hdf --dpi 120 --num-img 2 --image-only

# View results
xpcs_webplot serve ./html
```

9.11.2 Publication Quality Workflow

```
# High-quality output
xpcs_webplot plot data.hdf --dpi 600 --num-img 12

# Results ready for publication
```

9.11.3 Batch Processing Workflow

```
# Process all files in parallel
xpcs_webplot plot /data/experiment/*.hdf --num-processes 8

# Combine results
xpcs_webplot combine ./html

# Serve for review
xpcs_webplot serve ./html
```

9.11.4 Continuous Monitoring Workflow

```
# Terminal 1: Monitor and process
xpcs_webplot plot /data/incoming --monitor --target-dir ./results

# Terminal 2: Serve results
xpcs_webplot serve ./results

# Results automatically appear as files are processed
```