



HERMES Manual! Everybody cheer!

Version 8.1.2025

Table of Contents

1. OVERVIEW	2
2. INSTALLATION	2
3. DATA ACQUISITION	2
3.1 SYSTEM OVERVIEW.....	3
3.2 DIRECTORY STRUCTURE.....	3
3.3 CONFIGURATION FILE	3
3.4 COMMAND LINE INTERFACE (CLI)	4
3.4.1 DEFAULT BEHAVIOR.....	4
3.4.2 CLI FLAGS	5
3.4.3 VERBOSITY LEVELS.....	5
3.4.4 DRY RUN MODE	5
3.5 EXAMPLES.....	6
3.6 PARAMETER PRECEDENCE.....	6
3.7 ACQUISITION PROCESS FLOW	7
4. UNPACKING DATA	7
4.1 CREATE UNPACKER	7
4.2 UNPACKER COMMAND LINE INTERFACE	8

4.2.1 USING THE CLI	8
4.2.2 UNPACKER CONFIGURATION FILE	8
4.2.3 .RAWSIGNALS STRUCTURE	10
4.2.4 EXAMPLES.....	10
5. ANALYZING DATA IN HERMES PACKAGES	12
5.1 READER.PY	12
5.1.1 READING FUNCTIONS	12
5.1.2 EXPORTER FUNCTIONS	13
5.1.3 SUMMARY FUNCTIONS	13
5.1.4 FILTERING FUNCTIONS	14
5.1.1B EXPORTER.PY EXAMPLES	ERROR! BOOKMARK NOT DEFINED.
5.2 PLOTTER.PY	14
5.3 EXAMPLE NOTEBOOK FILES	14

1. Overview

HERMES is an open-source library of Python and C++ tools to acquire, unpack, and analyze data taken on event-based Timepix3 camera systems developed by Amsterdam Scientific Instruments (ASI). HERMES has three primary functions:

1) Acquire

Easily connect to TPX3Cams and simplify the data acquisition process.

2) Unpack

Unpack the .tpx3 binary files from acquisition and unpack into a usable form.

3) Analyze

Provide basic data analysis tools for users of TPX3 camera systems.

2. Installation

I have no idea how pixi works.

3. Data Acquisition

3.1 System Overview

The acquisition scripts interface with the TPX3Cam and SPIDR readout boards using the `tpx3serval` Python library. These scripts are capable of configuring the camera, setting up run directories, logging configuration files, and performing one or more acquisition runs.

To acquire data, the TPX3Cam must already be running and connected to Serval. For more information view the Serval manual.

3.2 Directory Structure

HERMES adopts a structured directory layout. The working directory contains one folder for each run and several subfolders for specific data types.

Example Directory Layout:

```
Working Directory
├── README.md
├── scripts
│   ├── acquireTpx3.py
│   └── acquire_config.ini
├── initFiles
├── [run_1]
│   ├── imageFiles
│   ├── previewFiles
│   ├── statusFiles
│   ├── tpx3Files
│   └── tpx3Logs
├── [run_2]
├── [run_3]
└── [run_N]
```

The `acquireTpx3.py` script will automatically generate these folders if they do not already exist.

3.3 Configuration File

The `acquire_config.ini` file defines all configurable parameters for acquisition.

Sections and Parameters:

- **[WorkingDir]**

- `path_to_working_dir`: Full path to working directory (required)
- `path_to_init_files`: Path for initialization files (default: `initFiles/`)
- `path_to_status_files`: Path for status files
- `path_to_log_files`: Path for log files
- `path_to_image_files`: Path for image files
- `path_to_preview_files`: Path for preview files
- `path_to_rawSignal_files`: Path for `.rawSignals` files
- `path_to_raw_files`: Path for raw `.tpx3` files

- **[ServerConfig]**

- `serverurl`: URL for TPX3Cam server (default: `http://localhost:8080`)
- `path_to_server`: Path to the Serval directory
- `path_to_server_config_files`: Path to camera settings directory
- `bpc_file_name`: Pixel configuration filename
- `dac_file_name`: DAC configuration filename
- `destinations_file_name`: Server destinations file
- `detector_config_file_name`: Detector configuration file

- **[RunSettings]**

- `run_name`: Name for the run (used as folder name and in filenames)
- `run_number`: Starting run number (default: `0000`)
- `trigger_period_in_seconds`: Camera trigger period
- `exposure_time_in_seconds`: Exposure time (must be \leq trigger period)
- `trigger_delay_in_seconds`: Delay before triggers
- `number_of_triggers`: Number of triggers per run
- `number_of_runs`: Total number of runs to perform
- `global_timestamp_interval_in_seconds`: Timestamp interval

3.4 Command Line Interface (CLI)

3.4.1 Default Behavior

The CLI provides a flexible way to run acquisitions. Defaults are built into the script; a configuration file and/or CLI flags can override these defaults.

Usage:

In the same directory as the `acquireTpx3.py` script, run:

```
python acquireTpx3.py [options]
```

- No configuration file is required by default.
- Built-in defaults can be used directly.
- Configuration options can be provided via:
 1. Config file (`-c` or `--config`)
 2. CLI flags (highest precedence)

Default Behavior (no config):

- Trigger period: 10 s
- Exposure time: 9 s
- Number of runs: 1

3.4.2 CLI Flags

General Options

- `-h, --help`: Information on available commands
- `-c, --config`: Path to config file
- `-W, --working-dir`: Working directory path
- `-r, --run-name`: Run name (folder name and filename prefix)
- `-N, --run-number`: Starting run number (integer, zero-padded as 0000)
- `-n, --num-runs`: Total number of runs
- `-t, --trigger-period`: Trigger period (s)
- `-e, --exposure`: Exposure time (s)
- `-T, --num-triggers`: Number of triggers per run
- `-v, --verbose`: Verbosity (0=quiet, 1=info, 2=debug)
- `--dry-run`: Print effective configuration and exit

3.4.3 Verbosity Levels

- **0 (quiet)**: Only errors printed
- **1 (info)**: Standard information messages (default)
- **2 (debug)**: Full configuration printouts and detailed logs



3.4.4 Dry Run Mode

Use `--dry-run` to preview the final merged configuration (defaults + config + CLI flags) without running any acquisition.

3.5 Examples

1. Use defaults and specify working directory



bash

 Copy  Edit

```
python acquireTpx3.py -W /data/acquisition_test
```

2. Load a config file and override exposure time



bash

 Copy  Edit

```
python acquireTpx3.py -c acquire_config.ini -e 5
```

3. Specify working directory, run name, and number of runs



bash

 Copy  Edit

```
python acquireTpx3.py -W /data/beam_test -r beam_2025 -n 10
```

4. Load config file and override multiple parameters



bash

 Copy  Edit

```
python acquireTpx3.py -c acquire_config.ini -e 7 -t 12 -T 25
```

5. Full example with almost all parameters

bash

 Copy  Edit

```
python acquireTpx3.py -c acquire_config.ini -W /data/full_test -r complex_run \
-N 5 -n 8 -t 15 -e 12 -T 30 -v 2
```

I'll make this official later...

3.6 Parameter Precedence

- 4 • **CLI flags** (highest priority)
- 5 • **Configuration file** (-c)
- 6 • **Built-in defaults** (lowest priority)

3.7 Acquisition Process Flow

1. **Configuration:** Script merges defaults, config file, and CLI flags.
2. **Directory Verification:** Working directory and run folder are created or cleaned.
3. **Camera Check:** TPX3Cam connection is verified.
4. **Run Execution:**
 - Run number is incremented and formatted.
 - Configuration files and detector status are logged.
 - Exposure is started using the configured parameters.
 - Data is written into the appropriate subdirectories.

For details on TPX3Cam server and dashboard behavior, refer to the **Serval Camera Manual**.

4. Unpacking Data

Unpacking data in HERMES requires use of C++ files to turn .tpx3 files into .rawSignals files.

4.1 Create Unpacker

In a terminal, navigate to your HERMES directory. This should be the directory that contains folders such as `src`, `workspace`, and `examples`. In this directory, run:

```
cd src/hermes && make && cp unpacker.config ../../workspace/ && cp  
bin/tpx3SpidrUnpacker ../../workspace/ && cd ../../
```

This will create the binary file to run the unpacker and copy it into the workspace area, along with a default configuration file. If you try to run this command multiple times, you may get the error:

```
make: Nothing to be done for 'all'.
```

In this case, navigate to `/src/hermes/bin` and delete `tpx3SpidrUnpacker`. Re-run the code and everything should work.

4.2 Unpacker Command Line Interface

4.2.1 Using the CLI

To run and test the unpacker, navigate to `workspace`. The HERMES unpacker utilizes a command line interface format to unpack data files from a location. You can view a help menu by inputting:

```
tpx3SpidrUnpacker OR tpx3SpidrUnpacker -h OR tpx3SpidrUnpacker --help
```

The following options are available for unpacking:

```
Input/Output Options:
-i, --inputFile <file>      Input TPX3 file
-I, --inputDir <dir>        Input directory (for batch mode)
-b, --batch                  Enable batch mode (requires -I <directo
-o, --outputDir <dir>       Output directory
-c, --configFile <file>    Configuration file

Processing Options:
-s, --sort                   Enable signal sorting
-w, --writeRawSignals        Enable writing raw signals
-W, --no-writeRawSignals    Disable writing raw signals
-C, --clusterPixels          Enable pixel clustering
-p, --writeOutPhotons        Enable writing photon data
-H, --fillHistograms        Enable histogram filling

Clustering Parameters:
-S, --epsSpatial <n>        Spatial epsilon for clustering (pixels)
-T, --epsTemporal <n>       Temporal epsilon for clustering (second
-P, --minPts <n>            Minimum points for clustering
-q, --queryRegion <n>       Query region for clustering

Diagnostic Options:
-m, --maxPackets <n>        Maximum packets to read (0=all)
-v, --verbose <level>      Verbose level (0-3, default: 1)
-h, --help                  Show this help message
```

4.2.2 Unpacker Configuration File

Like HERMES acquisition, all these parameters can also be specified in a configuration file. The template for the unpacker configuration file is given in `unpacker.config`, which was automatically copied into the workspace alongside the binary file.

The default configuration file appears as so:

```
# Path to folder with .tpx3 files
rawTPX3Folder = PATH/TO/YOUR/FOLDER

# File to process (or use ALL for batch mode)
rawTPX3File = ALL

# Output directory
outputFolder = PATH/TO/YOUR/FOLDER

# Signal processing options
writeRawSignals = true
sortSignals = true
fillHistograms = false
clusterPixels = false
writeOutPhotons = true

# Verbosity level
verboseLevel = 2

# Packet limits, set to 0 to unpack all packets
maxPacketsToRead = 0

# Clustering parameters
queryRegion = 0
epsSpatial = 2
epsTemporal = 500.0
minPts = 3
```

A clustering function is built into the unpacker of HERMES; however, many users may desire to cluster using custom parameters or functions, thus allowing for `clusterPixels` to be enabled or disabled. Below are definitions for each customization option:

`rawTPX3Folder` = The path to your .tpx3 directory that you want to unpack.

`rawTPX3File` = The filename of a specific .tpx3 file you want to unpack. Set to 'ALL' for batch mode, unpacking every file in `rawTPX3Folder`.

`outputFolder` = The path to your .rawSignalFiles directory that you want unpacked files to be saved in.

`writeRawSignals` = Enable/Disable ability to write raw signals

`sortSignals` = Enable/Disable ability to sort signals

`fillHistograms` = Enable/Disable ability to fill histograms (*I think this is outdated and needs to be removed).

`clusterPixels` = Enable/Disable ability to cluster pixel hits.

`writeOutPhotons` = I have no idea what this does.

`verboseLevel` = Gives user detailed terminal output depending on value. 0 = Silent mode, 1 = Basic Information, 2 = Detailed logs

`maxPacketsToRead` = Maximum number of packets to read. 1 packet is 64 bits, or generally is one 'event', whether that be a TDC, pixel hit, GTS, or control signal. Set to 0 to unpack all packets.

4.2.3 .rawSignals Structure

The HERMES unpacker saves unpacked files as .rawSignals files, which contain the following structure:

```
// Represents the data for a single raw signal.
struct signalData {
    uint32_t bufferNumber; // Buffer number from where signal was recorded
    uint8_t signalType;    // Type of the signal (TDC=1,Pixel=2,GTS=3)
    uint8_t xPixel;        // X-coordinate of the pixel
    uint8_t yPixel;        // Y-coordinate of the pixel
    double ToaFinal;       // Time of Arrival in seconds (final value)
    uint16_t TotFinal;     // Time over Threshold in nano seconds (final value)
    uint32_t groupID;      // Group ID for clustering.
};
```



This structure is further unpacked into an array with basic processing. See section 4._____ for HERMES functions to analyze .rawSignals files.

Generally, useful data will have a `signalType` of 1, 2, or 3. A `signalType` of 4 or 5 shows a SPIDR Control signal and TPX3 Control signal respectively, and any other `signalType` value is not useful and can be discarded. We are in the process of refining this unpacking to produce less useless information and retain a more memory-efficient system.

4.2.4 Examples

1. Unpack a single `.tpx3` file with default settings:



bash

 Copy  Edit

```
tpx3SpidrUnpacker -i data/run0001.tpx3 -o output/
```

2. Unpack a single `.tpx3` file with a configuration file:



bash

 Copy  Edit

```
tpx3SpidrUnpacker -i data/run0001.tpx3 -o output/ -c unpacker.config
```

3. Process an entire directory in batch mode:



bash

 Copy  Edit

```
tpx3SpidrUnpacker -I data/ -b -o output/
```

4. Unpack and cluster pixels with custom parameters:



bash

 Copy  Edit

```
tpx3SpidrUnpacker -i data/run0002.tpx3 -o output/ -C -S 2 -T 5e-9 -P 3
```

5. Unpack using most available options:

bash

 Copy  Edit

```
tpx3SpidrUnpacker \  
-I data/runs/ -b \  
-o output/full_test/ \  
-c unpacker.config \  
-s -w -p -H \  
-C -S 3 -T 1e-8 -P 5 -q 10 \  
-m 50000 \  
-v 2
```

Will make more official later.

5. Analyzing Data in HERMES packages

HERMES offers a handful of features to allow for data analysis of .rawSignals files.

HERMES has several built-in packages that allow for quick and easy data analysis. These packages are located in the `src/hermes/analysis` folder. In order to use these packages, include the following header as part of your imports:

```
from hermes.analysis._____ import _____
```

Replace the underscores with the specific package and class that you are trying to import. For analysis, it is suggested to include the following imports:

```
from hermes.analysis.reader import SignalDataReader  
from hermes.analysis.plotter import _____
```

Note that any changes made to a package file will only be implemented if the kernel is restarted to load the package once again. Below are all of the packages currently implemented into HERMES and the abilities their functions provide.

5.1 reader.py

This module provides functionality to read raw signal binary files (.rawSignals) and convert them into pandas DataFrames. The binary format is based on the signalData structure from the C++ HERMES code. See section 4.2.3 for this structure.

Other abilities include getting a quick summary of data in a DataFrame, initial filtering of specific values, and exporting DataFrames to .csv files.

5.1.1 Reading functions

This module contains two reading functions to be called by a user. Both work very similarly. The simpler of the two is:

```
read_rawsignals_file(path/to/rawsignals/file)
```

which takes a string path to a .rawSignals file and will export its information to a pandas dataframe. If multiple files must be processed, then it is preferable to use:

```
read_rawsignals_folder(path/to/rawsignals/folder, index_range="")
```

which takes a string path to a .rawSignals folder and a string index range with the form: “#:#:#”. The first number is the index of the first file you want to analyze. The second is the index of the last file you want to analyze. The third number is to step a certain amount. See examples below.

5.1.2 Exporter functions

HERMES has two basic functions to export a pandas DataFrame to either a .csv file or a .parquet file. Both functions are very simple and require a DataFrame and a string path to save the file. Here is an example of each of these exporter functions:

```
reader.export_to_csv(df, "path/to/save/testing.csv")
reader.export_to_parquet(df, "path/to/savetesting.parquet")
```

5.1.3 Summary functions

This module contains one summary function that prints basic information about the loaded DataFrame. To call this function, run:

```
reader.get_summary_stats(df, rows=#)
```

`df` is the DataFrame created by either of the reading functions and the `rows` number is how many rows you would like to see printed out. The default value for `rows` is 10. This function will print out how many signals were loaded, the columns of the DataFrame, the distribution of signal types, the time and pixel range, the number of buffers and groups, and a preview of the first few rows.

5.1.4 Filtering functions

5.1.5 Examples

```
# --- User inputs ---
rawsignal_dir = "PATH/TO/YOUR/RAWSIGNALS/DATA" # Directory containing raw signal files

#==== EXAMPLE USAGE CASES OF EXPORTER =====#

exporter = SignalDataExporter()

# Load all files in the rawsignals folder into a pandas dataframe
df = exporter.read_rawsignals_folder(rawsignal_dir)

# Load specific indices 5 through 50
df = exporter.read_rawsignals_folder(rawsignal_dir, index_range="5:50")

# Load specific indices 5 through 50 with a step of 2.
df = exporter.read_rawsignals_folder(rawsignal_dir, index_range="5:50:2")

# Load a single file directly (instead of using folder):
df = exporter.read_rawsignals_file("/path/to/data/file1.rawSignals")
```

5.2 plotter.py

5.3 Example Notebook Files

HERMES contains an example notebook of many of the functions shown above in the file `analysis_hermes.ipynb`. This file is laid out in a way that is intended to be as easy as possible to get started with. This file also contains a cell of print statements meant to provide with basic diagnostic information about the dataframes created. It is also recommended to use the extension ‘Data Wrangler’ to easily visualize data in a Jupyter notebook.