Logo, company name

AI-generated content may be incorrect.

HERMES Manual! Everybody cheer!

Table of Contents

[1. Overview 2](#_Toc204769046)

[2. Data Acquisition 2](#_Toc204769047)

[2.1 System Overview 2](#_Toc204769048)

[2.2 Directory Structure 2](#_Toc204769049)

[2.3 Configuration File 3](#_Toc204769050)

[2.4 Command Line Interface (CLI) 4](#_Toc204769051)

[2.4.1 Default Behavior 4](#_Toc204769052)

[2.4.2 CLI Flags 4](#_Toc204769053)

[2.4.3 Verbosity Levels 4](#_Toc204769054)

[2.4.4 Dry Run Mode 5](#_Toc204769055)

[2.5 Examples 5](#_Toc204769056)

[2.6 Parameter Precedence 6](#_Toc204769057)

[2.7 Acquisition Process Flow 6](#_Toc204769058)

[3. Unpacking Data 7](#_Toc204769059)

[3.1 Create Unpacker 7](#_Toc204769060)

[3.2 Unpacker Command Line Interface 7](#_Toc204769061)

[3.2.1 Using the CLI 7](#_Toc204769062)

[3.2.2 Unpacker Configuration File 8](#_Toc204769063)

[3.2.3 .rawSignals Structure 10](#_Toc204769064)

[3.2.4 Examples 10](#_Toc204769065)

[4. Analyzing Data 12](#_Toc204769066)

# 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.

1. Unpack

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

1. Analyze

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

# Installation

I have no idea how pixi works.

# Data Acquisition

## 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 by running and connected to Serval. For more information view the Serval manual.

## 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:Text

AI-generated content may be incorrect.

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

## 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 ≤ 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

## 2.4 Command Line Interface (CLI)

### 2.4.1 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:

Text

AI-generated content may be incorrect.

* 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

### 2.4.2 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

### 2.4.3 Verbosity Levels

 **0 (quiet):** Only errors printed

 **1 (info):** Standard information messages (default)

 **2 (debug):** Full configuration printouts and detailed logs

### 2.4.4 Dry Run Mode

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

## Examples

A screenshot of a computer

AI-generated content may be incorrect.

Ill make this official later…

## Parameter Precedence

1.  **CLI flags** (highest priority)
2.  **Configuration file** (-c)
3.  **Built-in defaults** (lowest priority)

## 2.7 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**.

# Unpacking Data

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

## 3.1 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/chermes && 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/chermes/bin and delete tpx3SpidrUnpacker. Re-run the code and everything should work.

## 3.2 Unpacker Command Line Interface

### 3.2.1 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:   


### 3.2.2 Unpacker Configuration File

Similar to HERMES acquisition, all of 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:   
Text

AI-generated content may be incorrect.

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.

### 3.2.3 .rawSignals Structure

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

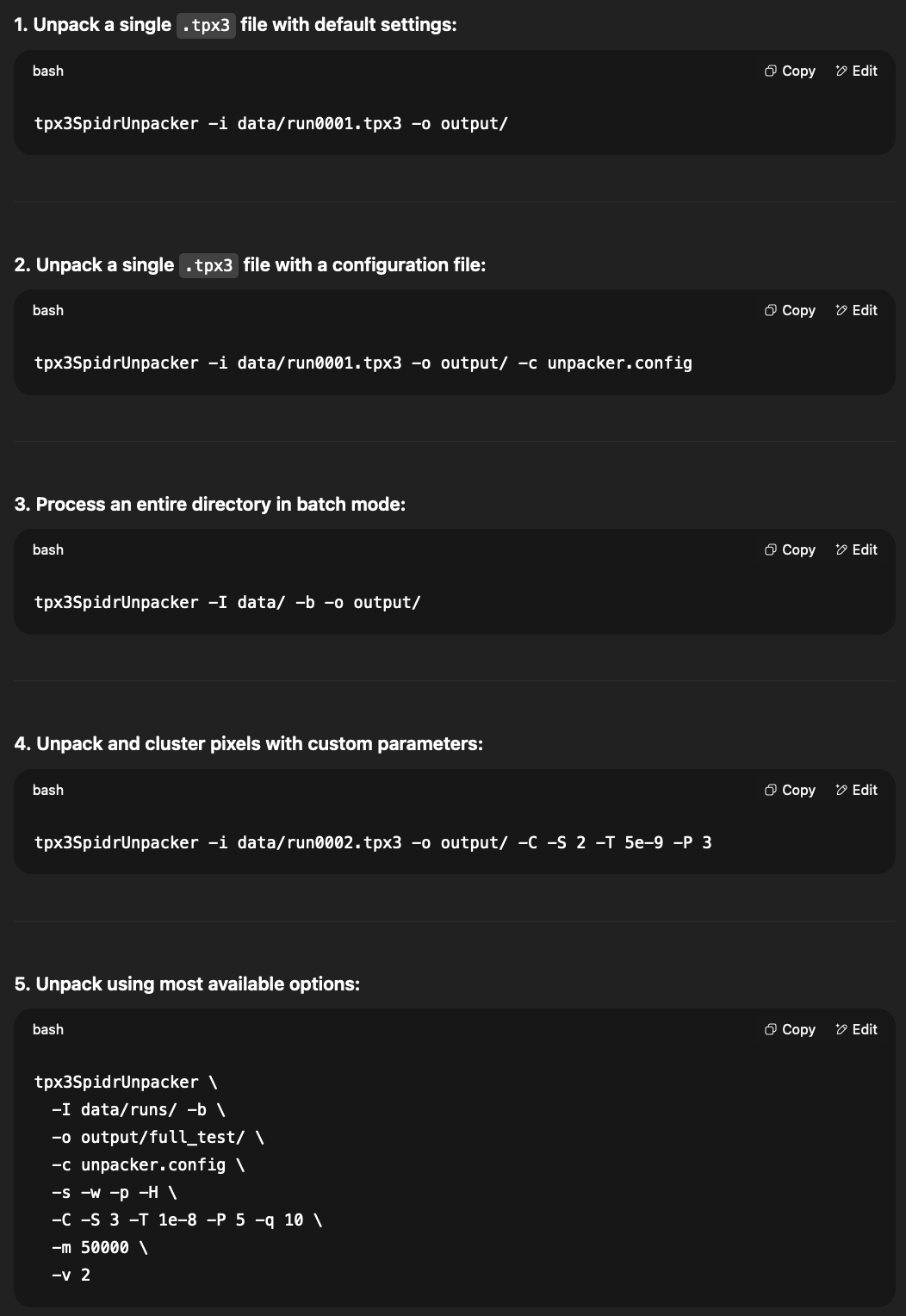
Text

AI-generated content may be incorrect.

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.

### 3.2.4 Examples



Will make more official later.

# Analyzing Data

HERMES offers a handful of features to allow for data analysis of .rawSignals files. In the examples directory of the HERMES folder,