# pyjoulescope_driver

***Release 1.5.2***

**Jetperch LLC**

**May 08, 2024**

# CONTENTS


**1 Joulescope Driver** **3**
- 1.1 Python Installation . . . 3
- 1.2 Building . . . 4

**2 C API** **7**
- 2.1 JSDRV API . . . 7
- 2.2 C-String utility functions . . . 23
- 2.3 Error codes . . . 27
- 2.4 Metadata handling . . . 30
- 2.5 Time representation and functions . . . 31
- 2.6 Topic string utility functions . . . 37
- 2.7 Union value type . . . 39

**3 Python API** **47**
- 3.1 Driver . . . 47
- 3.2 Record . . . 49
- 3.3 time64 . . . 50

**4 CHANGELOG** **53**
- 4.1 1.5.2 . . . 53
- 4.2 1.5.1 . . . 53
- 4.3 1.5.0 . . . 53
- 4.4 1.4.10 . . . 53
- 4.5 1.4.9 . . . 54
- 4.6 1.4.8 . . . 54
- 4.7 1.4.7 . . . 54
- 4.8 1.4.6 . . . 54
- 4.9 1.4.5 . . . 54
- 4.10 1.4.4 . . . 54
- 4.11 1.4.1 . . . 55
- 4.12 1.4.0 . . . 55
- 4.13 1.3.21 . . . 55
- 4.14 1.3.20 . . . 55
- 4.15 1.3.19 . . . 55
- 4.16 1.3.18 . . . 56
- 4.17 1.3.17 . . . 56
- 4.18 1.3.16 . . . 56
- 4.19 1.3.15 . . . 56
- 4.20 1.3.14 . . . 56
- 4.21 1.3.12 . . . 57

4.22 1.3.11 . . . 57
4.23 1.3.10 . . . 57
4.24 1.3.9 . . . 57
4.25 1.3.8 . . . 57
4.26 1.3.7 . . . 57
4.27 1.3.6 . . . 58
4.28 1.3.5 . . . 58
4.29 1.3.4 . . . 58
4.30 1.3.3 . . . 58
4.31 1.3.2 . . . 59
4.32 1.3.1 . . . 59
4.33 1.3.0 . . . 59
4.34 1.2.2 . . . 59
4.35 1.2.1 . . . 59
4.36 1.2.0 . . . 60
4.37 1.1.4 . . . 60
4.38 1.1.3 . . . 60
4.39 1.1.2 . . . 60
4.40 1.1.1 . . . 60
4.41 1.1.0 . . . 61
4.42 1.0.7 . . . 61
4.43 1.0.6 . . . 61
4.44 1.0.5 . . . 61
4.45 1.0.4 . . . 62
4.46 1.0.3 . . . 62
4.47 1.0.2 . . . 62
4.48 1.0.1 . . . 62
4.49 1.0.0 . . . 63
4.50 0.2.7 . . . 63
4.51 0.2.6 . . . 63
4.52 0.2.5 . . . 63
4.53 0.2.4 . . . 63
4.54 0.2.3 . . . 64
4.55 0.2.2 . . . 64
4.56 0.2.1 . . . 65

**5 Indices and tables** **67**

**Python Module Index** **69**

**Index** **71**

CHAPTER

ONE

# JOULESCOPE DRIVER

Welcome to the Joulescope™ Driver project. Joulescope is an affordable, precision DC energy analyzer that enables you to build better products.

This user-space C library communicates with Joulescope products to configure operation and receive data. The first-generation driver introduced in 2019 was written in Python. While Python proved to be a very flexible language enabling many user scripts, it was difficult to support other languages.
This second-generation driver launched in 2022 addresses several issues with the first-generation python driver including:

1. Improved event-driven API based upon PubSub for easier integration with user interfaces and other complicated software packages.
2. Improved portability for easier language bindings.
3. Improved performance.

For more information, see:

- source code
- documentation
- pypi
- Joulescope (Joulescope web store)
- jls (Joulescope file format)
- forum

## 1.1 Python Installation

The python bindings work with Python 3.9 and later. To use the python bindings, ensure that you have a compatible version of python installed on your host computer. Then:

```
python -m pip install pyjoulescope_driver
```

For Ubuntu, you will also need to *install the udev rules*.

You can then run the pyjoulescope_driver python entry points:

```
python -m pyjoulescope_driver --help
python -m pyjoulescope_driver scan
python -m pyjoulescope_driver info
python -m pyjoulescope_driver info * --verbose
```

Note that you may need to change "python" to "python3" or the full path.
You can also use a python virtual environment.

## 1.2 Building

Ensure that your computer has a development environment including CMake.

### 1.2.1 Windows

Install cmake and your favorite build toolchain such as Visual Studio, mingw64, wsl, ninja.

### 1.2.2 macOS

For macOS, install homebrew, then:

```
brew install pkgconfig python3
```

### 1.2.3 Ubuntu 22.04 LTS

For Ubuntu:

```
sudo apt install cmake build-essential ninja-build libudev-dev
```

You will also need to install the udev rules:

```
$ wget https://raw.githubusercontent.com/jetperch/joulescope_driver/main/99-joulescope.
↪rules
$ sudo cp 99-joulescope.rules /etc/udev/rules.d/
$ sudo udevadm control --reload-rules
```

### 1.2.4 Common

```
cd {your/repos/joulescope_driver}
mkdir build && cd build
cmake ..
cmake --build . && ctest .
```

This package includes a command-line tool, jsdrv:

```
jsdrv --help
jsdrv scan
```

### 1.2.5 Build python bindings

Install a compatible version of Python 3.9 or later. To install the pyjoulescope_driver dependencies:

```
cd {your/repos/joulescope_driver}
python -m pip install -U requirements.txt
```

You should then be able to build the native bindings:

```
python setup.py build_ext --inplace
```

You can build the package using isolation:

```
python -m build
```

Depending upon your system configuration, you may need to replace "python" with "python3" or the full path to your desired python installation.

On Windows, you may be prompted to install the Microsoft C++ Build Tools.

CHAPTER

TWO

# C API

## 2.1 JSDRV API

*group* **jsdrv_api**

The Joulescope driver host application interface.

This API is based upon a distributed publish-subscribe (PubSub) implementation. The system is arranged as a hierarchical set of topics which have values. Devices expose topics that control and configure operation. Devices also publish the measured data to topics for consumption by the host application.

The host code can set topic values using *jsdrv_publish()*. Host code can subscribe to topic value changes using *jsdrv_subscribe()*. This driver will then call the registered subscriber callback for each change on a matching topic. When the host code calls *jsdrv_subscribe()* with flags (JSDRV_SFLAG_RETAIN | JSDRV_SFLAG_PUB), then the driver immediately provides the retained values to synchronize the host state.

This implementation provides several nice features:

- Multiple value data types including integers, floats, str, json, binary
- Retained values
- Topic metadata for automatically populating user interfaces.
- Thread-safe, in-order operation. The driver provides thread-safe synchronous publish with timeout. However, subscriber callbacks are always invoked from the internal PubSub thread, and the host code is responsible for resynchronization.
- Guaranteed in-order topic traversal for retained messages based upon creation order.
- Device connection options:
  - connect and restore default settings
  - connect and resume using existing device settings.

Topic names use a hierarchical naming convention. Topics associated with a device start with a prefix: {p} = {backend}/{model}/{serial_number}

- {backend}: 1 character backend identifier 0-9, a-z, A-Z.
- {model}: The device model number, such as js220.
- {serial_number}: The device serial number string.

The rest of the topic is also hierarchical, and the following characters are reserved:

```
/?#$'"`&@%
```

Topics that start with '_' or '@' are directed only to the central PubSub instance within the driver itself.

Topics are presumed to have retained values unless the subtopic starts with the character '!'.

Topics with the following suffix characters have special meanings:

- '': metadata request (value should be NULL)
- '$': metadata response
- '?': query request (value should be NULL)
- '&': query response
- '#': return code. Value is i32 return code. 0=success.

Note that the topic owner will respond to jsdrv_publish operations with a return code to allow for fully synchronous operation.

The metadata values are JSON-formatted strings with the following data structure:

- dtype: one of [str, json, bin, f32, f64, u8, u16, u32, u64, i8, i16, i32, i64]
- brief: A brief string description (recommended).
- detail: A more detailed string description (optional).
- default: The recommended default value (optional).
- options: A list of options, where each option is each a flat list of: [value [, alt1 [, ...]]] The alternates must be given in preference order. The first value must be the value as dtype. The second value alt1 (when provided) is used to automatically populate user interfaces, and it can be the same as value. Additional values will be interpreted as equivalents.
- range: The list of [v_min, v_max] or [v_min, v_max, v_step]. Both v_min and v_max are *inclusive*. v_step defaults to 1 if omitted.
- format: Formatting hints string:
  - version: The u32 dtype should be interpreted as major8.minor8.patch16.
- flags: A list of flags for this topic. Options include:
  - ro: This topic cannot be updated.
  - hide: This topic should not appear in the user interface.
  - dev: Developer option that should not be used in production.

### Defines

**JSDRV_PAYLOAD_LENGTH_MAX**

The maximum size for normal PubSub messages.

**JSDRV_STREAM_HEADER_SIZE**

The header size of *jsdrv_stream_signal_s* before the data field.

**JSDRV_STREAM_DATA_SIZE**

The size of data in *jsdrv_stream_signal_s*.

**JSDRV_TIMEOUT_MS_DEFAULT**

The recommended default timeout.

**JSDRV_TIMEOUT_MS_INIT**

The recommended default *jsdrv_initialize()* timeout.

### Typedefs

typedef void (***jsdrv_subscribe_fn**)(void *user_data, const char *topic, const struct *jsdrv_union_s* *value)

Function called on topic updates.

This function will be called from the Joulescope driver frontend thread. The function is responsible for performing any resynchronization to an application target thread, if needed. However, the value only remains valid for the duration of the callback. Any binary or string data must be copied!

**Param user_data**
The arbitrary user data.

**Param topic**
The topic for this update.

**Param value**
The value for this update.

### Enums

enum **jsdrv_payload_type_e**

The payload type for *jsdrv_union_s.app*.

*Values:*

enumerator **JSDRV_PAYLOAD_TYPE_UNION**

enumerator **JSDRV_PAYLOAD_TYPE_STREAM**

enumerator **JSDRV_PAYLOAD_TYPE_STATISTICS**

enumerator **JSDRV_PAYLOAD_TYPE_BUFFER_INFO**

enumerator **JSDRV_PAYLOAD_TYPE_BUFFER_REQ**

enumerator **JSDRV_PAYLOAD_TYPE_BUFFER_RSP**

enum **jsdrv_element_type_e**

The element base type for streaming data.

**See also:**

*jsdrv_stream_signal_s*

*Values:*

enumerator **JSDRV_DATA_TYPE_UNDEFINED**

enumerator **JSDRV_DATA_TYPE_INT**

enumerator **JSDRV_DATA_TYPE_UINT**

enumerator **JSDRV_DATA_TYPE_FLOAT**

enum **jsdrv_field_e**

The signal field type for streaming data.

**See also:**

*jsdrv_stream_signal_s*

*Values:*

enumerator **JSDRV_FIELD_UNDEFINED**

enumerator **JSDRV_FIELD_CURRENT**

enumerator **JSDRV_FIELD_VOLTAGE**

enumerator **JSDRV_FIELD_POWER**

enumerator **JSDRV_FIELD_RANGE**

enumerator **JSDRV_FIELD_GPI**

enumerator **JSDRV_FIELD_UART**

enumerator **JSDRV_FIELD_RAW**

enum **jsdrv_time_type_e**

The time specification type.

*Values:*

enumerator **JSDRV_TIME_UTC**

Time in i64 34Q30 UTC. See jsdrv/time.h.

enumerator **JSDRV_TIME_SAMPLES**

Time in sample_ids for the corresponding channel.

enum **jsdrv_buffer_response_type_e**

The buffer response type.

*Values:*

enumerator **JSDRV_BUFFER_RESPONSE_SAMPLES**

Data contains samples.

enumerator **JSDRV_BUFFER_RESPONSE_SUMMARY**

Data contains summary statistics.

enum **jsdrv_subscribe_flag_e**

The subscriber flags for *jsdrv_subscribe()*.

*Values:*

enumerator **JSDRV_SFLAG_NONE**

No flags (always 0).

enumerator **JSDRV_SFLAG_RETAIN**

Immediately forward retained PUB and/or METADATA, depending upon JSDRV_PUBSUB_SFLAG_PUB and JSDRV_PUBSUB_SFLAG_METADATA_RSP.

enumerator **JSDRV_SFLAG_PUB**

Receive normal topic publish.

enumerator **JSDRV_SFLAG_METADATA_REQ**

Subscribe to receive metadata requests like “$” and “a/b/$”.

enumerator **JSDRV_SFLAG_METADATA_RSP**

Subscribe to receive metadata responses like “a/b/c$.

enumerator **JSDRV_SFLAG_QUERY_REQ**

Subscribe to receive query requests like “?” and “a/b/?”.

enumerator **JSDRV_SFLAG_QUERY_RSP**

Subscribe to receive query responses like “a/b/c?”.

enumerator **JSDRV_SFLAG_RETURN_CODE**

Subscribe to receive return code messages like “a/b/c#”.

enum **jsdrv_device_open_mode_e**

The driver mode for device open.

*Values:*

enumerator **JSDRV_DEVICE_OPEN_MODE_DEFAULTS**

Restore the device to its default, power-on state.

enumerator **JSDRV_DEVICE_OPEN_MODE_RESUME**

Update the driver with the device's existing state.

enumerator **JSDRV_DEVICE_OPEN_MODE_RAW**

Low-level open only, for use by internal tools.

### Functions

int32_t **jsdrv_initialize**(struct jsdrv_context_s **context, const struct *jsdrv_arg_s* *args, uint32_t timeout_ms)

Initialize the Joulescope driver (synchronous).

**Parameters**

- **context** – **[out]** The Joulescope driver context for future API calls.
- **args** – The initialization arguments or NULL. The argument list is terminated with an argument with an empty string for topic.
- **timeout_ms** – This function is always blocking and waits for up to timeout_ms for the operation to complete. When 0, use the default timeout [recommended]. When nonzero, override the default timeout.

**Returns**
0 or error code.

void **jsdrv_finalize**(struct jsdrv_context_s *context, uint32_t timeout_ms)

Finalize the Joulescope driver (synchronous).

This function releases all Joulescope driver resources including memory and threads.

This function must not be called from the driver frontend thread. Therefore, do not call this function directly from a subscriber callback.

**Parameters**

- **context** – The context from *jsdrv_initialize()*.
- **timeout_ms** – This function is always blocking and waits for up to timeout_ms for the operation to complete. When 0, use the default timeout [recommended]. When nonzero, override the default timeout.

int32_t **jsdrv_publish**(struct jsdrv_context_s *context, const char *topic, const struct *jsdrv_union_s* *value, uint32_t timeout_ms)

Publish a new value.

**Parameters**

- **context** – The Joulescope driver context.
- **topic** – The topic to publish.
- **value** – The new topic value.
- **timeout_ms** – When 0, publish asynchronously without awaiting the result. When nonzero, block awaiting the return code message.

**Returns**

0 or error code. All calls may return *JSDRV_ERROR_PARAMETER_INVALID*. Each topic may return other error codes.

int32_t **jsdrv_query**(struct jsdrv_context_s *context, const char *topic, struct *jsdrv_union_s* *value, uint32_t timeout_ms)

Query a retained value.

**Parameters**

- **context** – The Joulescope driver context.
- **topic** – The topic to query.
- **value** – **[inout]** The topic value. For string, JSON, and binary values, the provided value must be initialized with an allocated buffer. The value will be copied into this buffer or return *JSDRV_ERROR_TOO_SMALL*. If provided, this buffer will be ignored for other value types.
- **timeout_ms** – This function is always blocking and waits for up to timeout_ms for the operation to complete. When 0, use the default timeout [recommended]. When nonzero, override the default timeout.

**Returns**

0 or error code. All calls may return *JSDRV_ERROR_PARAMETER_INVALID* and *JSDRV_ERROR_TIMED_OUT*.

int32_t **jsdrv_subscribe**(struct jsdrv_context_s *context, const char *topic, uint8_t flags, *jsdrv_subscribe_fn* cbk_fn, void *cbk_user_data, uint32_t timeout_ms)

Subscribe to topic updates.

Synchronous, blocking subscription is most useful when flags contains *JSDRV_SFLAG_RETAIN*. The operation will not complete until cbk_fn() is called with all retained values.

**Parameters**

- **context** – The Joulescope driver context.
- **topic** – The subscription topic. The cbk_fn will be called whenever this topic or a child is updated.
- **flags** – The *jsdrv_subscribe_flag_e* bitmap. 0 (*JSDRV_SFLAG_NONE*) for no flags.
- **cbk_fn** – The function to call with topic updates. When called, this function will be invoked from the Joulescope driver thread. The function must NOT call *jsdrv_finalize()* or jsdrv_wait().
- **cbk_user_data** – The arbitrary data provided to cbk_fn.
- **timeout_ms** – When 0, subscribe asynchronously. When nonzero, block awaiting the subscription operation to complete.

**Returns**
0 or error code.

int32_t **jsdrv_unsubscribe**(struct jsdrv_context_s *context, const char *topic, *jsdrv_subscribe_fn* cbk_fn, void *cbk_user_data, uint32_t timeout_ms)

Unsubscribe to topic updates.

Most callers will want to use blocking, synchronous unsubscribe. Blocking unsubscribe ensures that cbk_fn() will not be called when this function returns. With asynchronous unsubscribe, cbk_fn() may be called for some indefinite number of times and duration until the unsubscribe request is handled internally.

**See also:**

*jsdrv_subscribe*

**See also:**

*jsdrv_unsubscribe_all*

**Parameters**

- **context** – The Joulescope driver context.
- **topic** – The subscription topic for unsubscription.
- **cbk_fn** – The previous subscribed function.
- **cbk_user_data** – The arbitrary data provided to cbk_fn which must match the value provided to *jsdrv_subscribe()*.
- **timeout_ms** – When 0, unsubscribe asynchronously. When nonzero, block awaiting the unsubscribe operation to complete.

**Returns**
0 or error code.

int32_t **jsdrv_unsubscribe_all**(struct jsdrv_context_s *context, *jsdrv_subscribe_fn* cbk_fn, void *cbk_user_data, uint32_t timeout_ms)

Unsubscribe from all topic updates (asynchronous).

Most callers will want to use blocking, synchronous unsubscribe. See *jsdrv_unsubscribe()* for details.

**See also:**

*jsdrv_unsubscribe*

**Parameters**

- **context** – The Joulescope driver context.
- **cbk_fn** – The previous subscribed function.
- **cbk_user_data** – The arbitrary data provided to cbk_fn which must match the value provided to *jsdrv_subscribe()*.
- **timeout_ms** – When 0, unsubscribe asynchronously. When nonzero, block awaiting the unsubscribe operation to complete.

**Returns**
0 or error code.

int32_t **jsdrv_open**(struct jsdrv_context_s *context, const char *device_prefix, int32_t mode)

Open a device.

This is a convenience function that wraps a single call to *jsdrv_publish()* with an i32 value. Language wrappers should not wrap this function and instead call *jsdrv_publish()* directly.

**Parameters**

- **context** – The Joulescope driver context.
- **device_prefix** – The device prefix string.
- **mode** – The *jsdrv_device_open_mode_e*.

**Returns**
0 or error code.

int32_t **jsdrv_close**(struct jsdrv_context_s *context, const char *device_prefix)

Close a device.

This is a convenience function that wraps a single call to *jsdrv_publish()*. Language wrappers should not wrap this function and instead call *jsdrv_publish()* directly.

**Parameters**

- **context** – The Joulescope driver context.
- **device_prefix** – The device prefix string.

**Returns**
0 or error code.

void **jsdrv_calibration_hash**(const uint32_t *msg, uint32_t length, uint32_t *hash)

Compute the calibration hash.

**Parameters**

- **msg[in]** – The calibration message.
- **length[in]** – The length of message in bytes which must be a multiple of 32 bytes.
- **hash[out]** – The u32[16] hash of the message. The total length is u32 x 16 = 64-byte = 512 bit.

struct **jsdrv_stream_signal_s**

*#include* <*jsdrv.h*> A contiguous, uncompressed sample block for a channel.

### Public Members

uint64_t **sample_id**

the starting sample id, which increments by decimate_factor.

uint8_t **field_id**

jsdrv_field_e

uint8_t **index**

The channel index within the field.

uint8_t **element_type**

jsdrv_element_type_e

uint8_t **element_size_bits**

The element size in bits.

uint32_t **element_count**

size of data in elements

uint32_t **sample_rate**

The frequency for sample_id.

uint32_t **decimate_factor**

The decimation factor from sample_id to data samples.

struct *jsdrv_time_map_s* **time_map**

The time map between sample_id (before decimate_factor) and UTC.

uint8_t **data**[JSDRV_STREAM_DATA_SIZE]

The channel data.

struct **jsdrv_statistics_s**

*#include <jsdrv.h>* The payload data structure for statistics updates.

### Public Members

uint8_t **version**

The version, only 1 currently supported.

uint8_t **rsv1_u8**

Reserved = 0.

uint8_t **rsv2_u8**

Reserved = 0.

uint8_t **decimate_factor**

The decimate factor from sample_id to calculated samples = 2.

uint32_t **block_sample_count**

Samples used to compute this block, in decimated samples.

uint32_t **sample_freq**

The samples per second for *_sample_id (undecimated)

uint32_t **rsv3_u8**

Reserved = 0.

uint64_t **block_sample_id**

First sample in this block's statistics computation.

uint64_t **accum_sample_id**

First sample in the integration statistics computation.

double **i_avg**

The average current over the block.

double **i_std**

The standard deviation of current over the block.

double **i_min**

The minimum current value in the block.

double **i_max**

The maximum current value in the block.

double **v_avg**

The average voltage over the block.

double **v_std**

The standard deviation of voltage over the block.

double **v_min**

The minimum voltage value in the block.

double **v_max**

The maximum voltage value in the block.

double **p_avg**

The average power over the block.

double **p_std**

The standard deviation of power over the block.

double **p_min**

The minimum power value in the block.

double **p_max**

The maximum power value in the block.

double **charge_f64**

The charge (integral of current) from accum_sample_id as a 64-bit float.

double **energy_f64**

The energy (integral of power) from accum_sample_id as a 64-bit float.

uint64_t **charge_i128**[2]

The charge (integral of current) from accum_sample_id as a 128-bit signed integer with 2**-31 scale.

uint64_t **energy_i128**[2]

The energy (integral of power) from accum_sample_id as a 128-bit signed integer with 2**-31 scale.

struct *jsdrv_time_map_s* **time_map**

The time map between sample_id and UTC.

struct **jsdrv_time_range_utc_s**

*#include* <*jsdrv.h*> A UTC-defined time range.

Times are int64 34Q30 in UTC from the epoch. See jsdrv/time.h.

### Public Members

int64_t **start**

The time for data[0] (inclusive).

int64_t **end**

The time for data[-1] (inclusive).

uint64_t **length**

The number of evenly-spaced entries.

struct **jsdrv_time_range_samples_s**

*#include* <*jsdrv.h*> A sample_id-defined time range.

Times are in uint64_t sample_ids for the corresponding channel.

### Public Members

uint64_t **start**

The time for data[0] (inclusive).

uint64_t **end**

The time for data[-1] (inclusive).

uint64_t **length**

The number of evenly-spaced entries.

struct **jsdrv_buffer_info_s**

*#include* <*jsdrv.h*> The signal buffer information.

This structure contains both the size and range. The size is populated immediately, even if the buffer is still empty. As the buffer contents grows, the buffer range will approach the buffer size. When the buffer is full, the range will be close to the size. The range when full is not required to be the entire size to allow for buffering optimizations.

### Public Members

uint8_t **version**

The response format version == 1.

uint8_t **rsv1_u8**

Reserved, set to 0.

uint8_t **rsv2_u8**

Reserved, set to 0.

uint8_t **rsv3_u8**

Reserved, set to 0.

uint8_t **field_id**

jsdrv_field_e

uint8_t **index**

The channel index within the field.

uint8_t **element_type**

jsdrv_element_type_e

uint8_t **element_size_bits**

The element size in bits.

char **topic**[JSDRV_TOPIC_LENGTH_MAX]

The source topic that provides *jsdrv_stream_signal_s*.

int64_t **size_in_utc**

The total buffer size in UTC time.

uint64_t **size_in_samples**

The total buffer size in samples.

struct *jsdrv_time_range_utc_s* **time_range_utc**

In UTC time.

struct *jsdrv_time_range_samples_s* **time_range_samples**

In sample time.

struct *jsdrv_time_map_s* **time_map**

The map between samples and utc time.

union **jsdrv_buffer_request_time_range_u**

*#include* <*jsdrv.h*> The time range union for buffer requests.

### Public Members

struct *jsdrv_time_range_utc_s* **utc**

struct *jsdrv_time_range_samples_s* **samples**

struct **jsdrv_buffer_request_s**

*#include* <*jsdrv.h*> Request data from the streaming sample buffer.

To make a sample request that returns jsdrv_buffer_sample_response_s, only specify end or length. Set the unused value to zero.

If both end and length are specified, then the request may return jsdrv_buffer_summary_response_s. However, if the specified increment is less than or equal to one sample duration, then the request will return jsdrv_buffer_sample_response_s.

For JSDRV_TIME_UTC with end and length > 1, the time increment between samples is: time_incr = (time_end - time_start) / (length - 1)

Using python, the x-axis time is then: x = np.linspace(time_start, time_end, length, dtype=np.int64)

For JSDRV_TIME_SAMPLES with end and length > 1, the sample increment between samples is: sample_id_incr = (sample_id_end - sample_id_start) / (length - 1)

The buffer implementation may deduplicate requests using the combination rsp_topic and rsp_id.

### Public Members

uint8_t **version**

The request format version == 1.

int8_t **time_type**

jsdrv_time_type_e

uint8_t **rsv1_u8**

Reserved, set to 0.

uint8_t **rsv2_u8**

Reserved, set to 0.

uint32_t **rsv3_u32**

Reserved, set to 0.

char **rsp_topic**[JSDRV_TOPIC_LENGTH_MAX]

The topic for this response.

int64_t **rsp_id**

The additional identifier to include in the response.

struct **jsdrv_summary_entry_s**

*#include <jsdrv.h>* A single summary statistics entry.

### Public Members

float **avg**

The average (mean) over the window.

float **std**

The standard deviation over the window.

float **min**

The maximum value over the window.

float **max**

The minimum value over the window.

struct **jsdrv_buffer_response_s**

*#include <jsdrv.h>* The response to *jsdrv_buffer_request_s* produced by the memory buffer.

The response populates both info.time_range_utc and info.time_range_samples. Both length values are equal, and specify the number of returned data samples. For response_type JSDRV_BUFFER_RESPONSE_SUMMARY, the data

is *jsdrv_summary_entry_s*[info.time_range_utc.length]. info.element_type is JSDRV_DATA_TYPE_UNDEFINED and info.element_size_bits is sizeof(jsdrv_summary_entry_s) * 8.

For response_type JSDRV_BUFFER_RESPONSE_SAMPLES, the data type depends upon info.element_type and info.element_size_bits.

### Public Members

uint8_t **version**

The response format version == 1.

uint8_t **response_type**

jsdrv_buffer_response_type_e

uint8_t **rsv1_u8**

Reserved, set to 0.

uint8_t **rsv2_u8**

Reserved, set to 0.

uint32_t **rsv3_u32**

Reserved, set to 0.

int64_t **rsp_id**

The value provided to *jsdrv_buffer_request_s*.

struct *jsdrv_buffer_info_s* **info**

The response information.

uint64_t **data**[]

The response data.

The data type for the response varies. Unfortunately, C does not support flexible arrays in union types. Your code should cast the data to the appropriate type. Use info.time_range_samples.length for the number of data elements in this response data.

For response_type JSDRV_BUFFER_RESPONSE_SUMMARY, the data is *jsdrv_summary_entry_s*[info.time_range_samples.length].

For response_type JSDRV_BUFFER_RESPONSE_SAMPLES, the data is defined by info.element_type and info.element_size_bits.

struct **jsdrv_arg_s**

*#include <jsdrv.h>* The initialization argument structure.

#### Public Members

const char *`topic`

The argument name.

struct *jsdrv_union_s* `value`

The argument value.

## 2.2 C-String utility functions

*group* `jsdrv_cstr`

C-style string utilities.

This module supplies c-style (null terminated byte array) string utilities. The utilities are designed for memory safety (unlike <string.h>).

#### Defines

`jsdrv_cstr_array_copy`(tgt, src)

Safely copy src to tgt.

**Parameters**

- `tgt` – The null-terminated destination character array.
- `src` – The null-terminated source string.

**Returns**

0 on success, 1 if truncated, -1 on other errors.

#### Functions

int `jsdrv_cstr_copy`(char *tgt, char const *src, size_t tgt_size)

Safely copy src to tgt.

**Parameters**

- `tgt` – The null-terminated destination string.
- `src` – The null-terminated source string. If NULL, then tgt will be populated with an empty string.
- `tgt_size` – The total number of total_bytes available in tgt.

**Returns**

0 on success, 1 if truncated, -1 on tgt NULL or tgt_size <= 0.

int `jsdrv_cstr_join`(char *tgt, char const *src1, char const *src2, size_t tgt_size)

Safely copy src to tgt.

**Parameters**

- `tgt` – The null-terminated destination string.
- `src1` – The first null-terminated source string. tgt may be src1.

- **src2** – The second null-terminated source string. tgt may NOT be src2.
- **tgt_size** – The total number of total_bytes available in tgt.

**Returns**
0 on success, 1 if truncated, -1 on tgt NULL or tgt_size <= 0.

int **jsdrv_cstr_casecmp**(const char *s1, const char *s2)

Compare strings ignoring case.

**Parameters**

- **s1** – The first null-terminated string.
- **s2** – The second null-terminated string.

**Returns**
0 if s1 and s2 are comparable, -1 if s1 < s2, 1 if s2 > s1.

const char ***jsdrv_cstr_starts_with**(const char *s, const char *prefix)

Determine if a string starts with another string.

**Parameters**

- **s** – The string to search.
- **prefix** – The case-sensitive string prefix to match in s.

**Returns**
0 on no match. On match, return the pointer to s at the location of the first character after the matching prefix.

const char ***jsdrv_cstr_ends_with**(const char *s, const char *prefix)

Determine if a string ends with another string.

**Parameters**

- **s** – The string to search.
- **prefix** – The case-sensitive string suffix to match in s.

**Returns**
0 on no match. On match, return the pointer to s at the location of the first character matching the suffix.

int **jsdrv_cstr_to_u32**(const char *src, uint32_t *value)

Convert a string to an unsigned 32-bit integer.

**Parameters**

- **src** – The input source string containing an integer. Strings that start with "0x" are processed as case-insensitive hexadecimal.
- **value** – The output unsigned 32-bit integer value.

**Returns**
0 on success or error code. On error, the value will not be modified. To allow default values on parsing errors, set value before calling this function.

int **jsdrv_cstr_to_i32**(const char *src, int32_t *value)

Convert a string to an signed 32-bit integer.

**Parameters**

- **src** – The input source string containing an integer.

- **value** – The output integer value.

**Returns**

0 on success or error code. On error, the value will not be modified. To allow default values on parsing errors, set value before calling this function.

int **jsdrv_cstr_to_i32s**(const char *src, int32_t exponent, int32_t *value)

Convert a fractional value into a scaled 32-bit integer.

Examples:

```
jsdrv_cstr_to_i32s("1", 0, &x) => 1
jsdrv_cstr_to_i32s("1", 2, &x) => 100
jsdrv_cstr_to_i32s("1.01", 2, &x) => 101
jsdrv_cstr_to_i32s("   1.01   ", 2, &x) => 101
jsdrv_cstr_to_i32s("+1.01", 2, &x) => 101
jsdrv_cstr_to_i32s("-1.01", 2, &x) => -101
jsdrv_cstr_to_i32s("1.019", 2, &x) => 101
```

**Parameters**

- **src** – The input source string. Excess fractional digits will be truncated and ignored, not rounded.
- **exponent** – The base10 exponent.
- **value** – The resulting integer value.

**Returns**

0 on success or error code. On error, the value will not be modified. To allow default values on parsing errors, set value before calling this function.

int **jsdrv_cstr_to_u64**(const char *src, uint64_t *value)

Convert a string to an unsigned 64-bit integer.

**Parameters**

- **src** – The input source string containing an integer. Strings that start with "0x" are processed as case-insensitive hexadecimal.
- **value** – The output unsigned 64-bit integer value.

**Returns**

0 on success or error code. On error, the value will not be modified. To allow default values on parsing errors, set value before calling this function.

int **jsdrv_cstr_to_i64**(const char *src, int64_t *value)

Convert a string to an signed 64-bit integer.

**Parameters**

- **src** – The input source string containing an integer.
- **value** – The output integer value.

**Returns**

0 on success or error code. On error, the value will not be modified. To allow default values on parsing errors, set value before calling this function.

int **jsdrv_cstr_to_f32**(const char *src, float *value)

Convert a string to a floating point number.

**Parameters**

- **src** – The input source string containing a floating point number.
- **value** – The output floating point value.

**Returns**
0 on success or error code. On error, the value will not be modified. To allow default values on parsing errors, set value before calling this function.

int **jsdrv_u32_to_cstr**(uint32_t u32, char *str, size_t str_size)

Convert an unsigned 32-bit integer to a string.

**Parameters**

- **u32** – The input unsigned 32-bit integer value.
- **str** – The output string. To support all values, it should have at least 11 bytes free.
- **str_size** – The available bytes in str.

**Returns**
0 on success or error code. On error, return str[0] = 0 unless size is <= 0.

int **jsdrv_cstr_toupper**(char *s)

Convert a string to upper case.

Equivalent to nonstandard strupr().

**Parameters**
**s** – **[inout]** The null-terminated ASCII string to convert which is modified in place. This function will corrupt UTF-8 data! NULL will cause error. All other inputs are valid and return success.

**Returns**
0 on success or error code.

int **jsdrv_cstr_to_index**(char const *s, char const *const *table, int *index)

Convert a string value into an index into table.

**Parameters**

- **s** – **[in]** The null-terminated ASCII string value input.
- **table** – **[in]** The list of possible null-terminated string values. The list is terminated with a NULL entry.
- **index** – **[inout]** The output index. Index is only modified on a successful conversion, a default value can be set before calling this function.

**Returns**
0 or error code.

int **jsdrv_cstr_to_bool**(char const *s, bool *value)

Convert a string value into a boolean.

**Parameters**

- **s** – **[in]** The null-terminated ASCII string value input which is not case sensitive. True values include “TRUE”, “ON”, “1”, “ENABLE”. False values include “FALSE”, “OFF”, “0”, “DISABLE”.

- **value** – **[inout]** The output value. Value is only modified on a successful conversion, a default value can be set before calling this function.

**Returns**
SUCCESS or error code.

uint8_t **jsdrv_cstr_hex_to_u4**(char v)

Convert a hex character to a 4-bit nibble.

**Parameters**
**v** – The ASCII character value to convert.

**Returns**
The nibble value (0 to 16) or 0 on error.

char **jsdrv_cstr_u4_to_hex**(uint8_t v)

Convert a 4-bit nibble to a hex character.

**Parameters**
**v** – The 4-bit nibble value.

**Returns**
The ASCII character value or '0' on error.

## 2.3 Error codes

*group* **jsdrv_ec**

Standardize error code definitions.

See errc

### Defines

**JSDRV_SUCCESS**

A shorter, less confusing alias for success.

### Enums

enum **jsdrv_error_code_e**

The list of error codes.

*Values:*

enumerator **JSDRV_ERROR_SUCCESS**

Success (no error)

enumerator **JSDRV_ERROR_UNSPECIFIED**

Unspecified error.

enumerator **JSDRV_ERROR_NOT_ENOUGH_MEMORY**

Insufficient memory to complete the operation.

enumerator **JSDRV_ERROR_NOT_SUPPORTED**

Operation is not supported.

enumerator **JSDRV_ERROR_IO**

Input/output error.

enumerator **JSDRV_ERROR_PARAMETER_INVALID**

The parameter value is invalid.

enumerator **JSDRV_ERROR_INVALID_RETURN_CONDITION**

The function return condition is invalid.

enumerator **JSDRV_ERROR_INVALID_CONTEXT**

The context is invalid.

enumerator **JSDRV_ERROR_INVALID_MESSAGE_LENGTH**

The message length in invalid.

enumerator **JSDRV_ERROR_MESSAGE_INTEGRITY**

The message integrity check failed.

enumerator **JSDRV_ERROR_SYNTAX_ERROR**

A syntax error was detected.

enumerator **JSDRV_ERROR_TIMED_OUT**

The operation did not complete in time.

enumerator **JSDRV_ERROR_FULL**

The target of the operation is full.

enumerator **JSDRV_ERROR_EMPTY**

The target of the operation is empty.

enumerator **JSDRV_ERROR_TOO_SMALL**

The target of the operation is too small.

enumerator **JSDRV_ERROR_TOO_BIG**

The target of the operation is too big.

enumerator **JSDRV_ERROR_NOT_FOUND**

The requested resource was not found.

enumerator **JSDRV_ERROR_ALREADY_EXISTS**

The requested resource already exists.

enumerator **JSDRV_ERROR_PERMISSIONS**

Insufficient permissions to perform the operation.

enumerator **JSDRV_ERROR_BUSY**

The requested resource is currently busy.

enumerator **JSDRV_ERROR_UNAVAILABLE**

The requested resource is currently unavailable.

enumerator **JSDRV_ERROR_IN_USE**

The requested resource is currently in use.

enumerator **JSDRV_ERROR_CLOSED**

The requested resource is currently closed.

enumerator **JSDRV_ERROR_SEQUENCE**

The requested operation was out of sequence.

enumerator **JSDRV_ERROR_ABORTED**

The requested operation was previously aborted.

enumerator **JSDRV_ERROR_SYNCHRONIZATION**

The target is not synchronized with the originator.

### Functions

const char ***jsdrv_error_code_name**(int ec)

Convert an error code into its short name.

**Parameters**

**ec** – **[in]** The error code (jsdrv_error_code_e).

**Returns**

The short string name for the error code.

const char ***jsdrv_error_code_description**(int ec)

Convert an error code into its description.

**Parameters**

**ec** – **[in]** The error code (jsdrv_error_code_e).

**Returns**

The user-meaningful description of the error.

## 2.4 Metadata handling

*group* **jsdrv_meta**

Handle JSON-formatted metadata.

### Functions

int32_t **jsdrv_meta_syntax_check**(const char *meta)

Check the JSON metadata syntax.

**Parameters**
**meta** – The JSON metadata.

**Returns**
0 or error code.

int32_t **jsdrv_meta_dtype**(const char *meta, uint8_t *dtype)

Get the data type.

**Parameters**

- **meta** – The JSON metadata.
- **dtype** – **[out]** The jsdrv_union_e data type.

**Returns**
0 or error code.

int32_t **jsdrv_meta_default**(const char *meta, struct *jsdrv_union_s* *value)

Get the default value.

**Parameters**

- **meta** – The JSON metadata.
- **value** – **[out]** The parsed default value.

**Returns**
0 or error code.

int32_t **jsdrv_meta_value**(const char *meta, struct *jsdrv_union_s* *value)

Validate a parameter value using the metadata.

**Parameters**

- **meta** – The JSON metadata.
- **value** – **[inout]** The value, which is modified in place.

**Returns**
0 or error code.

## 2.5 Time representation and functions

*group* **jsdrv_time**

JSDRV time representation.

The C standard library includes time.h which is very inconvenient for embedded systems. This module defines a much simpler 64-bit fixed point integer for representing time. The value is 34Q30 with the upper 34 bits to represent whole seconds and the lower 30 bits to represent fractional seconds. A value of 2**30 (1 << 30) represents 1 second. This representation gives a resolution of 2 ** -30 (approximately 1 nanosecond) and a range of +/- 2 ** 33 (approximately 272 years). The value is signed to allow for simple arithmetic on the time either as a fixed value or as deltas.

Certain elements may elect to use floating point time given in seconds. The macros *JSDRV_TIME_TO_F64()* and *JSDRV_F64_TO_TIME()* facilitate converting between the domains. Note that double precision floating point is not able to maintain the same resolution over the time range as the 64-bit representation. *JSDRV_TIME_TO_F32()* and *JSDRV_F32_TO_TIME()* allow conversion to single precision floating point which has significantly reduce resolution compared to the 34Q30 value.

Float64 only has 53 bits of precision, which can only represent up to 104 days with nanosecond precision. While this duration and precision is often adequate for relative time, it is insufficient to store absolute time from the epoch. In contrast, the int64 34Q30 time with nanosecond resolution can store ±272 years relative to its epoch.

For applications that need floating-point time, store a separate offset in seconds relative to the starting time. This improves precision between intervals. For example:

```
x = np.linspace(0, time_end - time_start, length, dtype=np.float64)
```

### Defines

**JSDRV_TIME_Q**

The number of fractional bits in the 64-bit time representation.

**JSDRV_TIME_MAX**

The maximum (positive) time representation.

**JSDRV_TIME_MIN**

The minimum (negative) time representation.

**JSDRV_TIME_EPOCH_UNIX_OFFSET_SECONDS**

The offset from the standard UNIX (POSIX) epoch.

This offset allows translation between JSDRV time and the standard UNIX (POSIX) epoch of Jan 1, 1970.

The value was computed using python3:

```
import dateutil.parser
dateutil.parser.parse('2018-01-01T00:00:00Z').timestamp()
```

JSDRV chooses a different epoch to advance "zero" by 48 years!

**JSDRV_TIME_SECOND**

The fixed-point representation for 1 second.

**JSDRV_FRACT_MASK**

The mask for the fractional bits.

**JSDRV_TIME_MILLISECOND**

The approximate fixed-point representation for 1 millisecond.

**JSDRV_TIME_MICROSECOND**

The approximate fixed-point representation for 1 microsecond.

CAUTION: this value is 0.024% accurate (240 ppm)

**JSDRV_TIME_NANOSECOND**

The approximate fixed-point representation for 1 nanosecond.

WARNING: this value is only 6.7% accurate!

**JSDRV_TIME_MINUTE**

The fixed-point representation for 1 minute.

**JSDRV_TIME_HOUR**

The fixed-point representation for 1 hour.

**JSDRV_TIME_DAY**

The fixed-point representation for 1 day.

**JSDRV_TIME_WEEK**

The fixed-point representation for 1 week.

**JSDRV_TIME_MONTH**

The average fixed-point representation for 1 month (365 day year).

**JSDRV_TIME_YEAR**

The approximate fixed-point representation for 1 year (365 days).

**JSDRV_TIME_TO_F64**(x)

Convert the 64-bit fixed point time to a double.

**Parameters**

- **x** – The 64-bit signed fixed point time.

**Returns**

The time as a double p. Note that IEEE 747 doubles only have 52 bits of precision, so the result will be truncated for very small deltas.

**JSDRV_TIME_TO_F32**(x)

Convert the 64-bit fixed point time to single precision float.

**Parameters**

- **x** – The 64-bit signed fixed point time.

**Returns**

The time as a float p in seconds. Note that IEEE 747 singles only have 23 bits of precision, so the result will likely be truncated.

`JSDRV_TIME_TO_SECONDS(x)`

Convert to 32-bit unsigned seconds.

**Parameters**

- **x** – The 64-bit signed fixed point time.

**Returns**

The 64-bit unsigned time in seconds, rounded to nearest.

`JSDRV_TIME_TO_MILLISECONDS(x)`

Convert to milliseconds.

**Parameters**

- **x** – The 64-bit signed fixed point time.

**Returns**

The 64-bit signed time in milliseconds, rounded to nearest.

`JSDRV_TIME_TO_MICROSECONDS(x)`

Convert to microseconds.

**Parameters**

- **x** – The 64-bit signed fixed point time.

**Returns**

The 64-bit signed time in microseconds, rounded to nearest.

`JSDRV_TIME_TO_NANOSECONDS(x)`

Convert to nanoseconds.

**Parameters**

- **x** – The 64-bit signed fixed point time.

**Returns**

The 64-bit signed time in nanoseconds, rounded to nearest.

`JSDRV_SECONDS_TO_TIME(x)`

Convert to 64-bit signed fixed point time.

**Parameters**

- **x** – he 32-bit unsigned time in seconds.

**Returns**

The 64-bit signed fixed point time.

`JSDRV_MILLISECONDS_TO_TIME(x)`

Convert to 64-bit signed fixed point time.

**Parameters**

- **x** – The 32-bit unsigned time in milliseconds.

**Returns**
The 64-bit signed fixed point time.

**JSDRV_MICROSECONDS_TO_TIME**(x)

Convert to 64-bit signed fixed point time.

**Parameters**

- **x** – The 32-bit unsigned time in microseconds.

**Returns**
The 64-bit signed fixed point time.

**JSDRV_NANOSECONDS_TO_TIME**(x)

Convert to 64-bit signed fixed point time.

**Parameters**

- **x** – The 32-bit unsigned time in microseconds.

**Returns**
The 64-bit signed fixed point time.

**JSDRV_TIME_STRING_LENGTH**

The length of the ISO 8601 string produced by *jsdrv_time_to_str()*.

### Functions

int64_t **JSDRV_F64_TO_TIME**(double x)

Convert the double precision time to 64-bit fixed point time.

**Parameters**
**x** – The double-precision floating point time in seconds.

**Returns**
The time as a 34Q30.

int64_t **JSDRV_F32_TO_TIME**(float x)

Convert the single precision float time to 64-bit fixed point time.

**Parameters**
**x** – The single-precision floating point time in seconds.

**Returns**
The time as a 34Q30.

int64_t **JSDRV_TIME_TO_COUNTER**(int64_t x, uint64_t z)

Convert to counter ticks, rounded to nearest.

**Parameters**

- **x** – The 64-bit signed fixed point time.
- **z** – The counter frequency in Hz.

**Returns**
The 64-bit time in counter ticks.

int64_t **JSDRV_TIME_TO_COUNTER_RZERO**(int64_t x, uint64_t z)

Convert to counter ticks, rounded towards zero.

**Parameters**

- **x** – The 64-bit signed fixed point time.
- **z** – The counter frequency in Hz.

**Returns**
The 64-bit time in counter ticks.

int64_t **JSDRV_TIME_TO_COUNTER_RINF**(int64_t x, uint64_t z)

Convert to counter ticks, rounded towards infinity.

**Parameters**

- **x** – The 64-bit signed fixed point time.
- **z** – The counter frequency in Hz.

**Returns**
The 64-bit time in counter ticks.

int64_t **JSDRV_COUNTER_TO_TIME**(uint64_t x, uint64_t z)

Convert a counter to 64-bit signed fixed point time.

**Parameters**

- **x** – The counter value in ticks.
- **z** – The counter frequency in Hz.

**Returns**
The 64-bit signed fixed point time.

int64_t **JSDRV_TIME_ABS**(int64_t t)

Compute the absolute value of a time.

**Parameters**
**t** – The time.

**Returns**
The absolute value of t.

int64_t **jsdrv_time_min**(int64_t a, int64_t b)

Return the minimum time.

**Parameters**

- **a** – The first time value.
- **b** – The second time value.

**Returns**
The smaller value of a and b.

int64_t **jsdrv_time_max**(int64_t a, int64_t b)

Return the maximum time.

**Parameters**

- **a** – The first time value.
- **b** – The second time value.

**Returns**
The larger value of a and b.

int32_t **jsdrv_time_to_str**(int64_t t, char *str, size_t size)

Converts jsdrv time to an ISO 8601 string.

**See also:**

http://howardhinnant.github.io/date_algorithms.html

**See also:**

https://stackoverflow.com/questions/7960318/math-to-convert-seconds-since-1970-into-date-and-vice-versa

**Parameters**

- **t** – The jsdrv time.
- **str** – The string buffer.
- **size** – The size of str in bytes, which should be at least JSDRV_TIME_STRING_LENGTH bytes to fit the full ISO 8601 string with the null terminator.

**Returns**
The number of characters written to buf, not including the null terminator. If this value is less than (JSDRV_TIME_STRING_LENGTH - 1), then the full string was truncated.

int64_t **jsdrv_time_from_counter**(const struct *jsdrv_time_map_s* *self, uint64_t counter)

Convert time from a counter value to JSDRV time.

**Parameters**

- **self** – The time mapping instance.
- **counter** – The counter value u64.

**Returns**
The JSDRV time i64.

uint64_t **jsdrv_time_to_counter**(const struct *jsdrv_time_map_s* *self, int64_t time64)

Convert time from JSDRV time to a counter value.

**Parameters**

- **self** – The time mapping instance.
- **time64** – The JSDRV time i64.

**Returns**
The counter value u64.

struct **jsdrv_time_map_s**

*#include <time.h>* Define a mapping between JSDRV time and a counter.

This mapping is often used to convert between an increment sample identifier value and JSDRV time.

The counter_rate is specified as a double which contains a 52-bit mantissa with fractional accuracy of 2e-16.

#### Public Members

int64_t **offset_time**

The offset specified as JSDRV time i64.

uint64_t **offset_counter**

The offset specified as counter values u64.

double **counter_rate**

The counter increment rate (Hz).

## 2.6 Topic string utility functions

*group* **jsdrv_topic**

Topic string utility functions.

#### Defines

**JSDRV_TOPIC_INIT**

Empty topic structure initializer.

#### Functions

void **jsdrv_topic_clear**(struct *jsdrv_topic_s* *topic)

Clear a topic structure instance to reset it to zero length.

**Parameters**

**topic** – **[inout]** The topic structure, which is modified in place.

void **jsdrv_topic_truncate**(struct *jsdrv_topic_s* *topic, uint8_t length)

Truncate a topic structure to a specified length.

If you store length before calling *jsdrv_topic_append()*, you can use this function to revert the append.

**Parameters**

- **topic** – **[inout]** The topic structure, which is modified in place.
- **length** – The desired length.

void **jsdrv_topic_append**(struct *jsdrv_topic_s* *topic, const char *subtopic)

Append a subtopic to a topic structure.

This function intelligently adds the '/' separator. If the topic does not already end with '/', it will be inserted first.

**See also:**

*jsdrv_topic_remove*

**Parameters**

- **topic** – **[inout]** The topic structure, which is modified in place.
- **subtopic** – The subtopic string to add.

int32_t **jsdrv_topic_remove**(struct *jsdrv_topic_s* *topic)

Remove a subtopic from the end of a topic structure.

This function intelligently removes the '/' separator. If the topic already ends with '/', then this it is ignored. This function then removes the "/subtopic". The resulting topic does NOT end in '/'.

**See also:**

*jsdrv_topic_append*

**Parameters**

**topic** – **[inout]** The topic structure, which is modified in place.

**Returns**

The number of characters removed.

void **jsdrv_topic_set**(struct *jsdrv_topic_s* *topic, const char *str)

Set the topic to the provided value.

This function will assert if no room remains.

**Parameters**

- **topic** – **[inout]** The topic structure, which is modified in place.
- **str** – The desired topic value.

void **jsdrv_topic_suffix_add**(struct *jsdrv_topic_s* *topic, char ch)

Adds a suffix character to the topic.

This function will assert if no room remains.

**Parameters**

- **topic** – **[inout]** The topic structure, which is modified in place.
- **ch** – The special character to append.

char **jsdrv_topic_suffix_remove**(struct *jsdrv_topic_s* *topic)

Remove the suffix character from a topic.

**Parameters**

**topic** – **[inout]** The topic structure, which is modified in place.

**Returns**

ch The suffix character removed or 0 if no character was removed.

struct **jsdrv_topic_s**

*#include <topic.h>* The topic structure.

#### Public Members

char **topic**[JSDRV_TOPIC_LENGTH_MAX]

The topic string.

uint8_t **length**

The length in bytes ignoring the null terminator.

## 2.7 Union value type

*group* **jsdrv_union**

A generic union value type.

#### Defines

**jsdrv_union_null**()

**jsdrv_union_null_r**()

**jsdrv_union_f32**(_value)

**jsdrv_union_f32_r**(_value)

**jsdrv_union_f64**(_value)

**jsdrv_union_f64_r**(_value)

**jsdrv_union_u8**(_value)

**jsdrv_union_u8_r**(_value)

**jsdrv_union_u16**(_value)

**jsdrv_union_u16_r**(_value)

**jsdrv_union_u32**(_value)

**jsdrv_union_u32_r**(_value)

**jsdrv_union_u64**(_value)

**jsdrv_union_u64_r**(_value)

**jsdrv_union_i8**(_value)

**jsdrv_union_i8_r**(_value)

**jsdrv_union_i16**(_value)

**jsdrv_union_i16_r**(_value)

**jsdrv_union_i32**(_value)

**jsdrv_union_i32_r**(_value)

**jsdrv_union_i64**(_value)

**jsdrv_union_i64_r**(_value)

**jsdrv_union_str**(_value)

**jsdrv_union_cstr**(_value)

**jsdrv_union_cstr_r**(_value)

**jsdrv_union_json**(_value)

**jsdrv_union_cjson**(_value)

**jsdrv_union_cjson_r**(_value)

**jsdrv_union_bin**(_value, _size)

**jsdrv_union_cbin**(_value, _size)

**jsdrv_union_cbin_r**(_value, _size)

### Enums

enum **jsdrv_union_e**

The allowed data types.

*Values:*

enumerator **JSDRV_UNION_NULL**

NULL value. Also used to clear existing value.

enumerator **JSDRV_UNION_STR**

UTF-8 string value, null terminated.

enumerator **JSDRV_UNION_JSON**

UTF-8 JSON string value, null terminated.

enumerator **JSDRV_UNION_BIN**

Raw binary value.

enumerator **JSDRV_UNION_RSV0**

Reserved, do not use.

enumerator **JSDRV_UNION_RSV1**

Reserved, do not use.

enumerator **JSDRV_UNION_F32**

32-bit IEEE 754 floating point

enumerator **JSDRV_UNION_F64**

64-bit IEEE 754 floating point

enumerator **JSDRV_UNION_U8**

Unsigned 8-bit integer value.

enumerator **JSDRV_UNION_U16**

Unsigned 16-bit integer value.

enumerator **JSDRV_UNION_U32**

Unsigned 32-bit integer value.

enumerator **JSDRV_UNION_U64**

Unsigned 64-bit integer value.

enumerator **JSDRV_UNION_I8**

Signed 8-bit integer value.

enumerator **JSDRV_UNION_I16**

Signed 16-bit integer value.

enumerator **JSDRV_UNION_I32**

Signed 32-bit integer value.

enumerator **JSDRV_UNION_I64**

Signed 64-bit integer value.

enum **jsdrv_union_flag_e**

The standardized Joulescope driver union flags.

Applications may define custom flags in *jsdrv_union_s.app*.

*Values:*

enumerator **JSDRV_UNION_FLAG_NONE**

No flags specified.

enumerator **JSDRV_UNION_FLAG_RETAIN**

The PubSub instance should retain this value.

enumerator **JSDRV_UNION_FLAG_CONST**

The value points to a const that will remain valid indefinitely.

enumerator **JSDRV_UNION_FLAG_HEAP_MEMORY**

The value uses dynamically allocated heap memory that must be freed.

### Functions

bool **jsdrv_union_eq**(const struct *jsdrv_union_s* *v1, const struct *jsdrv_union_s* *v2)

Check if two values are equal.

This check only compares the type and value. The types and values must match exactly. However, if ignores the additional fields [flags, op, app]. Use jsdrv_union_eq_strict() to also compare these fields. Use *jsdrv_union_equiv()* to more loosely check the value.

**Parameters**

- **v1** – The first value.
- **v2** – The second value.

**Returns**

True if equal, false if not equal.

bool **jsdrv_union_eq_exact**(const struct *jsdrv_union_s* *v1, const struct *jsdrv_union_s* *v2)

Check if two values are equal.

This check strictly compares every field.

**Parameters**

- **v1** – The first value.
- **v2** – The second value.

**Returns**

True if equal, false if not equal.

bool **jsdrv_union_equiv**(const struct *jsdrv_union_s* *v1, const struct *jsdrv_union_s* *v2)

Check if two values are equivalent.

This check performs type up-conversions to attempt to match the two fields.

**Parameters**

- **v1** – The first value.
- **v2** – The second value.

**Returns**

True if equal, false if not equal.

void **jsdrv_union_widen**(struct *jsdrv_union_s* *x)

Widen to the largest-size, compatible numeric type.

This widening conversion preserves float, signed, and unsigned characteristics. This check performs type up-conversions to attempt to match the two fields.

**See also:**

*jsdrv_union_as_type()*

**Parameters**

**x** – The value to widen in place.

int32_t **jsdrv_union_as_type**(struct *jsdrv_union_s* *x, uint8_t type)

Convert value to a specific type.

**Parameters**

- **x** – The value to convert in place.
- **type** – The target type.

**Returns**
0 or error code.

int32_t **jsdrv_union_to_bool**(const struct *jsdrv_union_s* *value, bool *rv)

Convert a value to a boolean.

**Parameters**

- **value** – The value to convert. For numeric types, and non-zero value is presumed true. String and JSON compare against a case insensitive string list using jsdrv_cstr_to_bool.. True is ["true", "on", "enable", "enabled", "yes"] and False is ["false", "off", "disable", "disabled", "no"]. All other values return an error.
- **rv** – The resulting boolean value.

**Returns**
0 or error code.

static inline bool **jsdrv_union_is_type_ptr**(const struct *jsdrv_union_s* *value)

Check if the union contains a pointer type.

**Parameters**
**value** – The union value.

**Returns**
True is the union contains a pointer type, false otherwise.

const char ***jsdrv_union_type_to_str**(uint8_t type)

Convert the type to a user-meaningful string.

**Parameters**
**type** – The jsdrv_union_e type.

**Returns**
The user-meaningful string representation for the type.

int32_t **jsdrv_union_value_to_str**(const struct *jsdrv_union_s* *value, char *str, uint32_t str_len, uint32_t opts)

Convert the value to a user-meaningful string.

**Parameters**

- **value** – The value.
- **str** – **[out]** The string to hold the value.
- **str_len** – The maximum length of str, in bytes.
- **opts** – The formatting options. 0=value only, 1=verbose with type and flags.

**Returns**
0 or error code.

union **jsdrv_union_inner_u**

*#include <union.h>* The actual value holder for *jsdrv_union_s*.

### Public Members

const char ***str**

JSDRV_UNION_STR, JSDRV_UNION_JSON.

const uint8_t ***bin**

JSDRV_UNION_BIN.

float **f32**

JSDRV_UNION_F32.

double **f64**

JSDRV_UNION_F64.

uint8_t **u8**

JSDRV_UNION_U8.

uint16_t **u16**

JSDRV_UNION_U16.

uint32_t **u32**

JSDRV_UNION_U32.

uint64_t **u64**

JSDRV_UNION_U64.

int8_t **i8**

JSDRV_UNION_I8.

int16_t **i16**

JSDRV_UNION_I16.

int32_t **i32**

JSDRV_UNION_I32.

int64_t **i64**

JSDRV_UNION_I64.

struct **jsdrv_union_s**

*#include <union.h>* The value holder for all types.

### Public Members

uint8_t **type**

The jsdrv_union_e data format indicator.

uint8_t **flags**

The jsdrv_union_flag_e flags.

uint8_t **op**

The application-specific operation.

uint8_t **app**

Application specific data. If unused, write to 0.

uint32_t **size**

payload size for pointer types, including null terminator for strings.

union *jsdrv_union_inner_u* **value**

The actual value.

CHAPTER

THREE

# PYTHON API

## 3.1 Driver

**class** pyjoulescope_driver.**Driver**

The Joulescope driver class.

**Parameters**

**timeout** – The optional timeout for open. None (default) uses the default timeout.

**close**(*device_prefix*, *timeout=None*)

Close an attached device.

**Parameters**

- **device_prefix** – The prefix name for the device.
- **timeout** – The timeout in seconds. None uses the default timeout.

**device_paths**(*timeout=None*)

List the currently connected devices.

**Parameters**

**timeout** – The timeout in seconds. None (default) uses the default timeout.

**Returns**

The list of device path strings.

**finalize**(*timeout=None*)

Finalize the driver.

**Parameters**

**timeout** – The timeout in seconds. None (default) uses the default timeout.

**log_level**

The current log level.

See *LogLevel*.

**open**(*device_prefix*, *mode=None*, *timeout=None*)

Open an attached device.

**Parameters**

- **device_prefix** – The prefix name for the device.
- **mode** – The open mode which is one of: * 'defaults': Reconfigure the device for default operation. * 'restore': Update our state with the current device state. * 'raw': Open the device in raw mode for development or firmware update. * None: equivalent to 'defaults'.

- **timeout** – The timeout in seconds. None uses the default timeout.

**publish**(*topic: str*, *value*, *timeout=None*)

Publish a value to a topic.

**Parameters**

- **topic** – The topic string.
- **value** – The value, which must pass validation for the topic.
- **timeout** – The timeout in seconds. None (default) uses the default timeout.

**Raise**
On error.

**query**(*topic: str*, *timeout=None*)

Query the value for a topic.

**Parameters**

- **topic** – The topic name.
- **timeout** – The timeout in seconds. None (default) uses the default timeout.

**Returns**
The value for the topic.

**Raise**
On error.

**subscribe**(*topic: str*, *flags*, *fn*, *timeout=None*)

Subscribe to receive topic updates.

**Parameters**

- **self** – The driver instance.
- **topic** – Subscribe to this topic string.
- **flags** – The flags or list of flags for this subscription. The flags can be int32 js-drv_subscribe_flag_e or string mnemonics, which are:
  - pub: Subscribe to normal values
  - pub_retain: Subscribe to normal values and immediately publish all matching retained values. With timeout, this function does not return successfully until all retained values have been published.
  - metadata_req: Subscribe to metadata requests (not normally useful).
  - metadata_rsp: Subscribe to metadata updates.
  - metadata_rsp_retain: Subscribe to metadata updates and immediately publish all matching retained metadata values.
  - query_req: Subscribe to all query requests (not normally useful).
  - query_rsp: Subscribe to all query responses.
  - return_code: Subscribe to all return code responses.
- **fn** – The function to call on each publish. Note that python dynamically constructs bound methods. To unsubscribe a method, provide the exact same bound method instance to unsubscribe.

- **timeout** – The timeout in float seconds to wait for this operation to complete. None waits the default amount. 0 does not wait and subscription will occur asynchronously.

**Raises**
**RuntimeError** – on subscribe failure.

**unsubscribe**(*topic*, *fn*, *timeout=None*)

Unsubscribe from a topic.

**Parameters**

- **topic** – The topic name string.
- **fn** – The function previously provided to *subscribe()*.
- **timeout** – The timeout in seconds. None (default) uses the default timeout.

**Raise**
On error.

**unsubscribe_all**(*fn*, *timeout=None*)

Unsubscribe a callback from all topics.

**Parameters**

- **fn** – The function previously provided to *subscribe()*.
- **timeout** – The timeout in seconds. None (default) uses the default timeout.

**Raise**
On error.

**class** pyjoulescope_driver.**ElementType**

The element type enumeration.

**class** pyjoulescope_driver.**ErrorCode**

The error code enumeration.

**class** pyjoulescope_driver.**Field**

The field enumeration.

**class** pyjoulescope_driver.**LogLevel**

The log level enumeration.

**class** pyjoulescope_driver.**SubscribeFlags**

The available subscribe flags.

## 3.2 Record

**class** pyjoulescope_driver.**Record**(*driver*, *device_path*, *signals=None*, *auto=None*)

Record streaming sample data to a JLS v2 file.

**Parameters**

- **driver** – The active driver instance.
- **device_path** – The device prefix path.
- **signals** – The list of signals to record. None=['current', 'voltage']

- **auto** – Configure automatic operation. Provide the list of automatic operations to perform, which can be: * signal_enable * signal_disable None (default) is equivalent to ['signal_enable', 'signal_disable']

Call `open()` to start recording and `close()` to stop.

**close**()

Close the recording and release all resources.

**open**(*filename*)

Start the recording.

**Parameters**
**filename** – The filename for the recording. Use time64.filename to produce a filename from timestamp.

**Returns**
self.

## 3.3 time64

The Python time standard uses POSIX (UNIX) time which is defined relative to Jan 1, 1970. Python uses floating point seconds to express time. This module defines a much simpler 64-bit fixed point integer for representing time which is much friendlier for microcontrollers. The value is 34Q30 with the upper 34 bits to represent whole seconds and the lower 30 bits to represent fractional seconds. A value of 2**30 (1 << 30) represents 1 second. This representation gives a resolution of 2 ** -30 (approximately 1 nanosecond) and a range of +/- 2 ** 33 (approximately 272 years). The value is signed to allow for simple arithmetic on the time either as a fixed value or as deltas.

For more details, see jsdrv/time.h.

pyjoulescope_driver.time64.**as_datetime**(*t*)

Convert a time to a python UTC timestamp.

**Parameters**
**t** – The time which can be: * datetime.datetime * python timestamp * integer Joulescope timestamp

pyjoulescope_driver.time64.**as_time64**(*t*)

Convert a time to a Joulescope timestamp.

**Parameters**
**t** – The time which can be: * datetime.datetime * python timestamp * integer Joulescope timestamp

pyjoulescope_driver.time64.**as_timestamp**(*t*)

Convert a time to a python UTC timestamp.

**Parameters**
**t** – The time which can be: * datetime.datetime * python timestamp * integer Joulescope timestamp

pyjoulescope_driver.time64.**duration_to_seconds**(*d*)

Convert a duration to float seconds. :param d: The duration specification, which is one of:

- A string formatted as fz where f is a valid floating-point value and z is either omitted, 's', 'm', 'h', 'd'.
- An integer in seconds.
- A floating-point value in seconds.

pyjoulescope_driver.time64.**filename**(*extension=None*, *t=None*)

Construct a filename from the time.

**Parameters**

- **extension** – The filename extension, such as '.png'. None (default) uses '.jls'.
- **t** – The time. None (default) uses the current time.

**Returns**

The filename.

pyjoulescope_driver.time64.**now**()

Get the current timestamp.

**Returns**

The time64 representation of the current time.

CHAPTER

FOUR

# CHANGELOG

This file contains the list of changes made to the Joulescope driver.

## 4.1 1.5.2

2024 May 8

- Added “capture” subcommand to jsdrv example executable.

## 4.2 1.5.1

2024 Apr 27

- Fixed JSDRV_DOWNSAMPLE_MODE_AVERAGE glitches due to C type conversion.
- Fixed incorrect downsampling initialization.

## 4.3 1.5.0

2024 Apr 24

- Added node.js binding.
- Fixed downsampling to round to nearest integer for integer types.
- Added downsampling mode JSDRV_DOWNSAMPLE_MODE_AVERAGE, but not yet connected.

## 4.4 1.4.10

2024 Mar 20

- Fixed JS110 to only issue one CTRL IN status request at a time.

## 4.5 1.4.9

2024 Mar 18

- Refactored record entry point.
- Fixed JS110 performance degradation (blocking status in device thread) #8

## 4.6 1.4.8

2024 Feb 26

- Fixed python statistics entry point to remove 1.0 second default duration.

## 4.7 1.4.7

2024 Feb 14

- Initialized buffer_mgr_s instance_.context to NULL.
- Improved python binding error reporting.
- Added optional duration to statistics entry point.
- Bumped minimum pyjls version from 0.8.2 to 0.9.2.

## 4.8 1.4.6

2023 Dec 9

- Fixed timestamp on POSIX (macOS, linux) systems.

## 4.9 1.4.5

2023 Dec 8

- Fixed stream buffer use-after-free and remove timeout.
- Added fuzz tester.

## 4.10 1.4.4

2023 Dec 7

- Improved J110 & JS220 state reset on open. Fixes JS110 fixes UTC time sync.
- Fixed JS220 communication reliability with FPGA 1.2.1 and FW 1.2.1.
- Improved JS110 time map for long-term stability.

## 4.11 1.4.1

2023 Nov 30

- Fixed JS220 "h/fs" restore.
- Improved JS220 UTC time sync with FW 1.2.0 & FPGA 1.2.0 support.
- Improved JS220 skip / drop sample handling.

## 4.12 1.4.0

2023 Nov 11

- Added jsdrv_calibration_hash and pyjoulescope_driver.calibration_hash.

## 4.13 1.3.21

2023 Nov 10

- Updated to FW 1.1.1.
- Added measure entry point to pyjoulescope_driver.

## 4.14 1.3.20

2023 Oct 26

- Updated to FPGA 1.1.0 for beta release (was mistakenly left at 1.0.4).

## 4.15 1.3.19

2023 Oct 25

- Fixed help text for "program" entry point.
- Build for Python 3.12.
- Upgraded to FW & FPGA 1.1.0 stable releases.
- Upgraded record entry point to pyjls 0.8.2 to 1.0.0.

## 4.16 1.3.18

2023 Jul 24

- Fixed buffer_signal summaryN incorrect when computing multiple values in a single call.
- Added "noexcept" to python callbacks. Cython 3.0 deprecates implicit noexcept.

## 4.17 1.3.17

2023 Jul 11

- Fixed "in frame_id mismatch" warning log message on first frame.
- Fixed JS220 signal "s/X/ctrl" 0 not correctly closing signal.
- Added Record "auto" parameter to optionally bypass automatic signal enable/disable.

## 4.18 1.3.16

2023 Jun 14

- Parallelized GitHub Actions build. Removed cibuildwheel.
- Fixed JS220 statistics for macOS. 1.3.15 was segfaulting on unaligned accesses.
- Added args parameter to pyjoulescope_driver.**main**.run().
- Added quit_ handling to jsdrv.exe examples.
- Bumped pyjls version check from 0.7.0 to 0.7.2.

## 4.19 1.3.15

2023 Jun 8

- Fixed JS220 firmware images omitted by 1.3.14 build process changes.
- Changed firmware image download to script invoke, not import.

## 4.20 1.3.14

2023 Jun 8

- Improved documentation.
- Improved GitHub Actions build process.
- Moved test/jsdrv_util to example/jsdrv.
- Bumped minimum python version from 3.8 to 3.9.

## 4.21 1.3.12

2023 May 31

- Reduced libusb backend log level. Was too active for JS110 statistics.
- Bumped JLS version to 0.7.0.

## 4.22 1.3.11

2023 May 24

- Added JS110 on-instrument (sensor) statistics option #3.

## 4.23 1.3.10

2023 May 19

- Improved threading and priorities on Windows.
- Improved Windows timer resolution (timeBeginPeriod).

## 4.24 1.3.9

2023 May 17

- Fixed installation on Ubuntu #6.

## 4.25 1.3.8

2023 May 16

- Updated info entry point.
- Added automatic include path for jsdrv static CMake builds.
- Added support for building a shared library. Initialize build subdir with "cmake -DBUILD_SHARED_LIBS=ON ..."

## 4.26 1.3.7

2023 Apr 28

- Fixed "info" entry point to correctly display multiple Joulescopes.
- Fixed "record" entry point to correctly add first UTC timestamp.
- Cleaned up build & install process (needs more work).

## 4.27 1.3.6

2023 Apr 27

- Fixed JS110 sample_id for downsampled 1-bit channels.
- Updated to pyjls 0.6.1 for improved robustness.

## 4.28 1.3.5

2023 Apr 26

- Added JS220 FW 1.0.7 and FPGA 1.0.4 as alpha & beta.

## 4.29 1.3.4

2023 Apr 26

- Fixed record jls version check.
- Send buffer signal clear on free.
- Updated to pyjls 0.6.0.
  - Fixed pyjoulescope_driver.record to not remove sample_id offset.

## 4.30 1.3.3

2023 Apr 19

- Cleared all message fields at allocation.
- Added api_timeout entry point test.
- Improved thread entry point test.
- Reordered unsubscribe to ensure callback validity.
- Added malloc/free mutex for guaranteed thread safety.
- Added runtime pyjls version check.
- Improved logging robustness and thread safety.
- Fixed JS110 open causing IN+ to OUT+ disconnect.
- Added JS110 open modes: defaults, resume.

## 4.31 1.3.2

2023 Apr 13

- Improved record close error handling.
- Added JS220 streaming data ignore when device not open.
- Improved record entry point.
  - Open in "defaults" mode by default with optional "restore".
  - Added parameter "–set" option.

## 4.32 1.3.1

2023 Apr 4

- Decreased JS110 status polling interval to reduce USB message spamming.
- Added JS110 streaming when only statistics requested (uses host-side stats).
- Increased process priority and backend thread priority for Windows.

## 4.33 1.3.0

2023 Mar 30

- Added pyjoulescope_driver.time64 module (from UI).
- Fixed buffer_signal summary_get handling on zero size.
- Added "record" module and entry point to record streaming samples.
- Fixed buffer_signal range advertisement when empty.
- Improved skipped / duplicate sample handling for JS220.

## 4.34 1.2.2

2023 Mar 19

- Fixed intermittent timeout broken for API calls.

## 4.35 1.2.1

2023 Mar 16

- Truncate memory buffer sample responses that are too long (segfault).
- Fixed buffer signal shift correction overflowing buffer (segfault).
- Fixed garbage data at end when shift required for u1 and u4 data types.
- Fixed zero length message send for highly downsampled signals.

## 4.36 1.2.0

2023 Mar 9

- Added memory buffer for f32, u4, u1 data types.
- Bumped python support (3.8 - 3.11). Dropped 3.7.
- Added API struct jsdrv_time_map_s and functions jsdrv_time_from_counter(), jsdrv_time_to_counter.
- Added jsdrv_time_map_s to jsdrv_stream_signal_s and jsdrv_statistics_s.
- Added host-side time map.
- Fixed JS110 sample stream message size.
- Fixed buffer_signal sample and utc response time entries.

## 4.37 1.1.4

2023 Jan 25

- Added JS110 GPI read request: s/gpi/+/!req -> s/gpi/+/!value.
- Fixed incorrect topic match on {device}/@/!finalize

## 4.38 1.1.3

2023 Jan 24

- Fixed open to correctly handle error on lower-level device open.
- Improved statistics output to include time/sample_freq and time/range.
- Reduced DEVICES_MAX for libusb backend to prevent breaking select.

## 4.39 1.1.2

2022 Dec 20

- Fixed i128 math functions js220_i128_neg() and js220_i128_lshift().
- Fixed incorrect constant for platforms with i32 constants.

## 4.40 1.1.1

2022 Nov 30

- Fixed build dependencies.
- Specified python 3.10 for GitHub actions.

## 4.41 1.1.0

2022 Nov 11

- Modified jsdrv_stream_signal_s (requires app recompile).
  - Fixes pyjoulescope and pyjoulescope_ui integration.
  - Added sample_rate.
  - Added decimate_factor.
- Added JS220 host-side downsampling.
- Rename JS110 topic s/fs to h/fs, matches new JS220 topic.
- Fixed old JS220 data sent on stream enable after disable.
- Added free for partial input stream messages on device close.
- Fixed firmware image include and added pkgdata load check.

## 4.42 1.0.7

2022 Nov 8

- Improved documentation.

## 4.43 1.0.6

2022 Nov 2

- Fixed build from tar.gz package.
- Updated README.

## 4.44 1.0.5

2022 Nov 1

- Fixed JS110 current range processing for window N and M.
- Fixed JS110 sample alignment.
- Fixed JS110 statistics generation time and rate.

## 4.45 1.0.4

2022 Oct 30

- Added JS220_CTRL_OP_DISCONNECT for cleaner disconnect/reconnect. Requires js220_ctrl firmware 1.0.4, no effect on earlier versions.
- Fixed JS110 support.
- Improve firmware update support.

## 4.46 1.0.3

2022 Oct 24

- Fixed linux by disabling BULK IN timeout. Linux was losing data on timeout. Fixed cancel, which still allows graceful shutdown.
- Added JS110 downsampling on host.
- Switched from obsolete pypiwin32 to pywin32.
- Fixed js110_sp_process when i_range was off or missing.

## 4.47 1.0.2

2022 Oct 15

- Fixed jsdrv_util app_match to work with multiple Joulescopes.

## 4.48 1.0.1

2022 Oct 9

- Added native and python threading demonstrations.
- Improve python binding to support u64 and i64 types.
- Added firmware/gateware updates.
  - Added build support for including most recent firmware images.
  - Added "program" entry point.
- Fixed access violation on publish to close device.
- Fixed python Driver.device_paths returning [""] instead of [] when no devices found.
- Reduced bulk in error on device removal to warning.

## 4.49 1.0.0

2022 Oct 6

- Fixed JS220 power computation.
- Fixed trigger GPI spec (7 not 255).
- Fixed sample_id 32-bit rollover handling.
- Added publish bytes support (needed for memory writes).

## 4.50 0.2.7

2022 Oct 4

- Added host-side power computation to JS220.
- Improved JS110 driver to provide current_range, gpi0, gpi1.
- Fixed power, charge and energy computation. Power fixes require FPGA 0.2.6 upgrade.
- Fixed JS110 stop/start streaming.
- Added JS110 statistics.

## 4.51 0.2.6

2022 Sep 29

- Reduced POSIX file descriptor limit request to 4096 max.

## 4.52 0.2.5

2022 Sep 26

- Fixed process hang due to logging on macOS & Linux.
- Fixed device disconnect for macOS & Linux.
- Increased file handle limit on macOS & Linux.

## 4.53 0.2.4

2022 Sep 24

- Added responder for removed devices.
- Fixed JSDRV_MSG_DEVICE_ADD retained subscribe. Was sending each device then full comma-separated list.
- Added check for API call from jsdrv frontend thread.

## 4.54 0.2.3

2022 Sep 21

- Fixed support for multiple (not just first found) of same device type.
- Improved JS110 support.
- Added macOS and Linux support using libusb library.

## 4.55 0.2.2

2022 Sep 9

- Completed JS220 memory read operation.
- Completed JS220 memory erase operation.
- Modifed jsdrvp_msg_alloc_value to support heap allocated values, not just jsdrv_publish(). Needed for read data messages.
- Added jsdrv_topic_remove().
- Completed controller-side signed & encrypted & signed firmware update.
- Move UI to private repository. Need to avoid Qt dependencies here.
- Removed "noise.py" entry point, which is not meaningful to end users.
- Added python command-line –log_level and –jsdrv_log_level options.
- Fixed python publish string values encoding.
- Improved native metadata validating to accept integer values as strings.
- Renamed python "set.py" to "set_parameter.py" to avoid reserved word "set".
- Added JS220 "opening" state to "h/state" metadata.
- Simplified native jsdrv_util set to use native support for string values.
- Fixed driver based upon beta build feedback.
- Fixed JS220 handling of missing/duplicated sample ids.
- Converted to element_size_bits from element_size_pow2.
- Added GPIO streaming support (u1 data type).
- Redefined JSDRV_DATA_TYPE_FLOAT to align with JLS v2.
- Reduced logging verbosity.

## 4.56 0.2.1

2022 Aug 1

- Initial public commit.

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## p

pyjoulescope_driver, 47
pyjoulescope_driver.time64, 50

# INDEX


## A

as_datetime() (*in module pyjoulescope_driver.time64*), 50
as_time64() (*in module pyjoulescope_driver.time64*), 50
as_timestamp() (*in module pyjoulescope_driver.time64*), 50

## C

close() (*pyjoulescope_driver.Driver method*), 47

## D

device_paths() (*pyjoulescope_driver.Driver method*), 47
Driver (*class in pyjoulescope_driver*), 47
duration_to_seconds() (*in module pyjoulescope_driver.time64*), 50

## E

ElementType (*class in pyjoulescope_driver*), 49
ErrorCode (*class in pyjoulescope_driver*), 49

## F

Field (*class in pyjoulescope_driver*), 49
filename() (*in module pyjoulescope_driver.time64*), 50
finalize() (*pyjoulescope_driver.Driver method*), 47

## J

jsdrv_arg_s (*C++ struct*), 22
jsdrv_arg_s::topic (*C++ member*), 23
jsdrv_arg_s::value (*C++ member*), 23
jsdrv_buffer_info_s (*C++ struct*), 19
jsdrv_buffer_info_s::element_size_bits (*C++ member*), 19
jsdrv_buffer_info_s::element_type (*C++ member*), 19
jsdrv_buffer_info_s::field_id (*C++ member*), 19
jsdrv_buffer_info_s::index (*C++ member*), 19
jsdrv_buffer_info_s::rsv1_u8 (*C++ member*), 19
jsdrv_buffer_info_s::rsv2_u8 (*C++ member*), 19
jsdrv_buffer_info_s::rsv3_u8 (*C++ member*), 19
jsdrv_buffer_info_s::size_in_samples (*C++ member*), 20
jsdrv_buffer_info_s::size_in_utc (*C++ member*), 20
jsdrv_buffer_info_s::time_map (*C++ member*), 20
jsdrv_buffer_info_s::time_range_samples (*C++ member*), 20
jsdrv_buffer_info_s::time_range_utc (*C++ member*), 20
jsdrv_buffer_info_s::topic (*C++ member*), 19
jsdrv_buffer_info_s::version (*C++ member*), 19
jsdrv_buffer_request_s (*C++ struct*), 20
jsdrv_buffer_request_s::rsp_id (*C++ member*), 21
jsdrv_buffer_request_s::rsp_topic (*C++ member*), 21
jsdrv_buffer_request_s::rsv1_u8 (*C++ member*), 21
jsdrv_buffer_request_s::rsv2_u8 (*C++ member*), 21
jsdrv_buffer_request_s::rsv3_u32 (*C++ member*), 21
jsdrv_buffer_request_s::time_type (*C++ member*), 21
jsdrv_buffer_request_s::version (*C++ member*), 21
jsdrv_buffer_request_time_range_u (*C++ union*), 20
jsdrv_buffer_request_time_range_u::samples (*C++ member*), 20
jsdrv_buffer_request_time_range_u::utc (*C++ member*), 20
jsdrv_buffer_response_s (*C++ struct*), 21
jsdrv_buffer_response_s::data (*C++ member*), 22
jsdrv_buffer_response_s::info (*C++ member*), 22
jsdrv_buffer_response_s::response_type (*C++ member*), 22
jsdrv_buffer_response_s::rsp_id (*C++ member*), 22
jsdrv_buffer_response_s::rsv1_u8 (*C++ member*), 22
jsdrv_buffer_response_s::rsv2_u8 (*C++ mem-*

*ber*), 22
jsdrv_buffer_response_s::rsv3_u32 (*C++ member*), 22
jsdrv_buffer_response_s::version (*C++ member*), 22
jsdrv_buffer_response_type_e (*C++ enum*), 11
jsdrv_buffer_response_type_e::JSDRV_BUFFER_RESPONSE_SAMPLES (*C++ enumerator*), 11
jsdrv_buffer_response_type_e::JSDRV_BUFFER_RESPONSE_SUMMARY (*C++ enumerator*), 11
jsdrv_calibration_hash (*C++ function*), 15
jsdrv_close (*C++ function*), 15
JSDRV_COUNTER_TO_TIME (*C++ function*), 35
jsdrv_cstr_array_copy (*C macro*), 23
jsdrv_cstr_casecmp (*C++ function*), 24
jsdrv_cstr_copy (*C++ function*), 23
jsdrv_cstr_ends_with (*C++ function*), 24
jsdrv_cstr_hex_to_u4 (*C++ function*), 27
jsdrv_cstr_join (*C++ function*), 23
jsdrv_cstr_starts_with (*C++ function*), 24
jsdrv_cstr_to_bool (*C++ function*), 26
jsdrv_cstr_to_f32 (*C++ function*), 25
jsdrv_cstr_to_i32 (*C++ function*), 24
jsdrv_cstr_to_i32s (*C++ function*), 25
jsdrv_cstr_to_i64 (*C++ function*), 25
jsdrv_cstr_to_index (*C++ function*), 26
jsdrv_cstr_to_u32 (*C++ function*), 24
jsdrv_cstr_to_u64 (*C++ function*), 25
jsdrv_cstr_toupper (*C++ function*), 26
jsdrv_cstr_u4_to_hex (*C++ function*), 27
jsdrv_device_open_mode_e (*C++ enum*), 11
jsdrv_device_open_mode_e::JSDRV_DEVICE_OPEN_MODE_DEFAULT (*C++ enumerator*), 12
jsdrv_device_open_mode_e::JSDRV_DEVICE_OPEN_MODE_RAW (*C++ enumerator*), 12
jsdrv_device_open_mode_e::JSDRV_DEVICE_OPEN_MODE_RESUME (*C++ enumerator*), 12
jsdrv_element_type_e (*C++ enum*), 9
jsdrv_element_type_e::JSDRV_DATA_TYPE_FLOAT (*C++ enumerator*), 10
jsdrv_element_type_e::JSDRV_DATA_TYPE_INT (*C++ enumerator*), 10
jsdrv_element_type_e::JSDRV_DATA_TYPE_UINT (*C++ enumerator*), 10
jsdrv_element_type_e::JSDRV_DATA_TYPE_UNDEFINED (*C++ enumerator*), 10
jsdrv_error_code_description (*C++ function*), 29
jsdrv_error_code_e (*C++ enum*), 27
jsdrv_error_code_e::JSDRV_ERROR_ABORTED (*C++ enumerator*), 29
jsdrv_error_code_e::JSDRV_ERROR_ALREADY_EXISTS (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_BUSY (*C++ enumerator*), 29
jsdrv_error_code_e::JSDRV_ERROR_CLOSED (*C++ enumerator*), 29
jsdrv_error_code_e::JSDRV_ERROR_EMPTY (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_FULL (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_IN_USE (*C++ enumerator*), 29
jsdrv_error_code_e::JSDRV_ERROR_INVALID_CONTEXT (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_INVALID_MESSAGE_LENGTH (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_INVALID_RETURN_CONDITION (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_IO (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_MESSAGE_INTEGRITY (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_NOT_ENOUGH_MEMORY (*C++ enumerator*), 27
jsdrv_error_code_e::JSDRV_ERROR_NOT_FOUND (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_NOT_SUPPORTED (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_PARAMETER_INVALID (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_PERMISSIONS (*C++ enumerator*), 29
jsdrv_error_code_e::JSDRV_ERROR_SEQUENCE (*C++ enumerator*), 29
jsdrv_error_code_e::JSDRV_ERROR_SUCCESS (*C++ enumerator*), 27
jsdrv_error_code_e::JSDRV_ERROR_SYNCHRONIZATION (*C++ enumerator*), 29
jsdrv_error_code_e::JSDRV_ERROR_SYNTAX_ERROR (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_TIMED_OUT (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_TOO_BIG (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_TOO_SMALL (*C++ enumerator*), 28
jsdrv_error_code_e::JSDRV_ERROR_UNAVAILABLE (*C++ enumerator*), 29
jsdrv_error_code_e::JSDRV_ERROR_UNSPECIFIED (*C++ enumerator*), 27
jsdrv_error_code_name (*C++ function*), 29
JSDRV_F32_TO_TIME (*C++ function*), 34
JSDRV_F64_TO_TIME (*C++ function*), 34
jsdrv_field_e (*C++ enum*), 10
jsdrv_field_e::JSDRV_FIELD_CURRENT (*C++ enumerator*), 10
jsdrv_field_e::JSDRV_FIELD_GPI (*C++ enumerator*), 10

jsdrv_field_e::JSDRV_FIELD_POWER (*C++ enumerator*), 10
jsdrv_field_e::JSDRV_FIELD_RANGE (*C++ enumerator*), 10
jsdrv_field_e::JSDRV_FIELD_RAW (*C++ enumerator*), 10
jsdrv_field_e::JSDRV_FIELD_UART (*C++ enumerator*), 10
jsdrv_field_e::JSDRV_FIELD_UNDEFINED (*C++ enumerator*), 10
jsdrv_field_e::JSDRV_FIELD_VOLTAGE (*C++ enumerator*), 10
jsdrv_finalize (*C++ function*), 12
JSDRV_FRACT_MASK (*C macro*), 32
jsdrv_initialize (*C++ function*), 12
jsdrv_meta_default (*C++ function*), 30
jsdrv_meta_dtype (*C++ function*), 30
jsdrv_meta_syntax_check (*C++ function*), 30
jsdrv_meta_value (*C++ function*), 30
JSDRV_MICROSECONDS_TO_TIME (*C macro*), 34
JSDRV_MILLISECONDS_TO_TIME (*C macro*), 33
JSDRV_NANOSECONDS_TO_TIME (*C macro*), 34
jsdrv_open (*C++ function*), 14
JSDRV_PAYLOAD_LENGTH_MAX (*C macro*), 8
jsdrv_payload_type_e (*C++ enum*), 9
jsdrv_payload_type_e::JSDRV_PAYLOAD_TYPE_BUFFER_INFO (*C++ enumerator*), 9
jsdrv_payload_type_e::JSDRV_PAYLOAD_TYPE_BUFFER_REQ (*C++ enumerator*), 9
jsdrv_payload_type_e::JSDRV_PAYLOAD_TYPE_BUFFER_RSP (*C++ enumerator*), 9
jsdrv_payload_type_e::JSDRV_PAYLOAD_TYPE_STATISTICS (*C++ enumerator*), 9
jsdrv_payload_type_e::JSDRV_PAYLOAD_TYPE_STREAM (*C++ enumerator*), 9
jsdrv_payload_type_e::JSDRV_PAYLOAD_TYPE_UNION (*C++ enumerator*), 9
jsdrv_publish (*C++ function*), 12
jsdrv_query (*C++ function*), 13
JSDRV_SECONDS_TO_TIME (*C macro*), 33
jsdrv_statistics_s (*C++ struct*), 16
jsdrv_statistics_s::accum_sample_id (*C++ member*), 17
jsdrv_statistics_s::block_sample_count (*C++ member*), 16
jsdrv_statistics_s::block_sample_id (*C++ member*), 17
jsdrv_statistics_s::charge_f64 (*C++ member*), 18
jsdrv_statistics_s::charge_i128 (*C++ member*), 18
jsdrv_statistics_s::decimate_factor (*C++ member*), 16
jsdrv_statistics_s::energy_f64 (*C++ member*), 18
jsdrv_statistics_s::energy_i128 (*C++ member*), 18
jsdrv_statistics_s::i_avg (*C++ member*), 17
jsdrv_statistics_s::i_max (*C++ member*), 17
jsdrv_statistics_s::i_min (*C++ member*), 17
jsdrv_statistics_s::i_std (*C++ member*), 17
jsdrv_statistics_s::p_avg (*C++ member*), 17
jsdrv_statistics_s::p_max (*C++ member*), 17
jsdrv_statistics_s::p_min (*C++ member*), 17
jsdrv_statistics_s::p_std (*C++ member*), 17
jsdrv_statistics_s::rsv1_u8 (*C++ member*), 16
jsdrv_statistics_s::rsv2_u8 (*C++ member*), 16
jsdrv_statistics_s::rsv3_u8 (*C++ member*), 17
jsdrv_statistics_s::sample_freq (*C++ member*), 16
jsdrv_statistics_s::time_map (*C++ member*), 18
jsdrv_statistics_s::v_avg (*C++ member*), 17
jsdrv_statistics_s::v_max (*C++ member*), 17
jsdrv_statistics_s::v_min (*C++ member*), 17
jsdrv_statistics_s::v_std (*C++ member*), 17
jsdrv_statistics_s::version (*C++ member*), 16
JSDRV_STREAM_DATA_SIZE (*C macro*), 8
JSDRV_STREAM_HEADER_SIZE (*C macro*), 8
jsdrv_stream_signal_s (*C++ struct*), 15
jsdrv_stream_signal_s::data (*C++ member*), 16
jsdrv_stream_signal_s::decimate_factor (*C++ member*), 16
jsdrv_stream_signal_s::element_count (*C++ member*), 16
jsdrv_stream_signal_s::element_size_bits (*C++ member*), 16
jsdrv_stream_signal_s::element_type (*C++ member*), 16
jsdrv_stream_signal_s::field_id (*C++ member*), 15
jsdrv_stream_signal_s::index (*C++ member*), 15
jsdrv_stream_signal_s::sample_id (*C++ member*), 15
jsdrv_stream_signal_s::sample_rate (*C++ member*), 16
jsdrv_stream_signal_s::time_map (*C++ member*), 16
jsdrv_subscribe (*C++ function*), 13
jsdrv_subscribe_flag_e (*C++ enum*), 11
jsdrv_subscribe_flag_e::JSDRV_SFLAG_METADATA_REQ (*C++ enumerator*), 11
jsdrv_subscribe_flag_e::JSDRV_SFLAG_METADATA_RSP (*C++ enumerator*), 11
jsdrv_subscribe_flag_e::JSDRV_SFLAG_NONE (*C++ enumerator*), 11
jsdrv_subscribe_flag_e::JSDRV_SFLAG_PUB (*C++ enumerator*), 11

jsdrv_subscribe_flag_e::JSDRV_SFLAG_QUERY_REQ (C++ enumerator), 11
jsdrv_subscribe_flag_e::JSDRV_SFLAG_QUERY_RSP (C++ enumerator), 11
jsdrv_subscribe_flag_e::JSDRV_SFLAG_RETAIN (C++ enumerator), 11
jsdrv_subscribe_flag_e::JSDRV_SFLAG_RETURN_CODE (C++ enumerator), 11
jsdrv_subscribe_fn (C++ type), 9
JSDRV_SUCCESS (C macro), 27
jsdrv_summary_entry_s (C++ struct), 21
jsdrv_summary_entry_s::avg (C++ member), 21
jsdrv_summary_entry_s::max (C++ member), 21
jsdrv_summary_entry_s::min (C++ member), 21
jsdrv_summary_entry_s::std (C++ member), 21
JSDRV_TIME_ABS (C++ function), 35
JSDRV_TIME_DAY (C macro), 32
JSDRV_TIME_EPOCH_UNIX_OFFSET_SECONDS (C macro), 31
jsdrv_time_from_counter (C++ function), 36
JSDRV_TIME_HOUR (C macro), 32
jsdrv_time_map_s (C++ struct), 36
jsdrv_time_map_s::counter_rate (C++ member), 37
jsdrv_time_map_s::offset_counter (C++ member), 37
jsdrv_time_map_s::offset_time (C++ member), 37
JSDRV_TIME_MAX (C macro), 31
jsdrv_time_max (C++ function), 35
JSDRV_TIME_MICROSECOND (C macro), 32
JSDRV_TIME_MILLISECOND (C macro), 32
JSDRV_TIME_MIN (C macro), 31
jsdrv_time_min (C++ function), 35
JSDRV_TIME_MINUTE (C macro), 32
JSDRV_TIME_MONTH (C macro), 32
JSDRV_TIME_NANOSECOND (C macro), 32
JSDRV_TIME_Q (C macro), 31
jsdrv_time_range_samples_s (C++ struct), 18
jsdrv_time_range_samples_s::end (C++ member), 19
jsdrv_time_range_samples_s::length (C++ member), 19
jsdrv_time_range_samples_s::start (C++ member), 19
jsdrv_time_range_utc_s (C++ struct), 18
jsdrv_time_range_utc_s::end (C++ member), 18
jsdrv_time_range_utc_s::length (C++ member), 18
jsdrv_time_range_utc_s::start (C++ member), 18
JSDRV_TIME_SECOND (C macro), 31
JSDRV_TIME_STRING_LENGTH (C macro), 34
JSDRV_TIME_TO_COUNTER (C++ function), 34
jsdrv_time_to_counter (C++ function), 36
JSDRV_TIME_TO_COUNTER_RINF (C++ function), 35
JSDRV_TIME_TO_COUNTER_RZERO (C++ function), 34
JSDRV_TIME_TO_F32 (C macro), 32
JSDRV_TIME_TO_F64 (C macro), 32
JSDRV_TIME_TO_MICROSECONDS (C macro), 33
JSDRV_TIME_TO_MILLISECONDS (C macro), 33
JSDRV_TIME_TO_NANOSECONDS (C macro), 33
JSDRV_TIME_TO_SECONDS (C macro), 33
jsdrv_time_to_str (C++ function), 36
jsdrv_time_type_e (C++ enum), 10
jsdrv_time_type_e::JSDRV_TIME_SAMPLES (C++ enumerator), 11
jsdrv_time_type_e::JSDRV_TIME_UTC (C++ enumerator), 10
JSDRV_TIME_WEEK (C macro), 32
JSDRV_TIME_YEAR (C macro), 32
JSDRV_TIMEOUT_MS_DEFAULT (C macro), 8
JSDRV_TIMEOUT_MS_INIT (C macro), 9
jsdrv_topic_append (C++ function), 37
jsdrv_topic_clear (C++ function), 37
JSDRV_TOPIC_INIT (C macro), 37
jsdrv_topic_remove (C++ function), 38
jsdrv_topic_s (C++ struct), 38
jsdrv_topic_s::length (C++ member), 39
jsdrv_topic_s::topic (C++ member), 39
jsdrv_topic_set (C++ function), 38
jsdrv_topic_suffix_add (C++ function), 38
jsdrv_topic_suffix_remove (C++ function), 38
jsdrv_topic_truncate (C++ function), 37
jsdrv_u32_to_cstr (C++ function), 26
jsdrv_union_as_type (C++ function), 43
jsdrv_union_bin (C macro), 40
jsdrv_union_cbin (C macro), 40
jsdrv_union_cbin_r (C macro), 40
jsdrv_union_cjson (C macro), 40
jsdrv_union_cjson_r (C macro), 40
jsdrv_union_cstr (C macro), 40
jsdrv_union_cstr_r (C macro), 40
jsdrv_union_e (C++ enum), 40
jsdrv_union_e::JSDRV_UNION_BIN (C++ enumerator), 40
jsdrv_union_e::JSDRV_UNION_F32 (C++ enumerator), 40
jsdrv_union_e::JSDRV_UNION_F64 (C++ enumerator), 40
jsdrv_union_e::JSDRV_UNION_I16 (C++ enumerator), 41
jsdrv_union_e::JSDRV_UNION_I32 (C++ enumerator), 41
jsdrv_union_e::JSDRV_UNION_I64 (C++ enumerator), 41
jsdrv_union_e::JSDRV_UNION_I8 (C++ enumerator), 41
jsdrv_union_e::JSDRV_UNION_JSON (C++ enumerator), 40

jsdrv_union_e::JSDRV_UNION_NULL (*C++ enumerator*), 40
jsdrv_union_e::JSDRV_UNION_RSV0 (*C++ enumerator*), 40
jsdrv_union_e::JSDRV_UNION_RSV1 (*C++ enumerator*), 40
jsdrv_union_e::JSDRV_UNION_STR (*C++ enumerator*), 40
jsdrv_union_e::JSDRV_UNION_U16 (*C++ enumerator*), 41
jsdrv_union_e::JSDRV_UNION_U32 (*C++ enumerator*), 41
jsdrv_union_e::JSDRV_UNION_U64 (*C++ enumerator*), 41
jsdrv_union_e::JSDRV_UNION_U8 (*C++ enumerator*), 41
jsdrv_union_eq (*C++ function*), 42
jsdrv_union_eq_exact (*C++ function*), 42
jsdrv_union_equiv (*C++ function*), 42
jsdrv_union_f32 (*C macro*), 39
jsdrv_union_f32_r (*C macro*), 39
jsdrv_union_f64 (*C macro*), 39
jsdrv_union_f64_r (*C macro*), 39
jsdrv_union_flag_e (*C++ enum*), 41
jsdrv_union_flag_e::JSDRV_UNION_FLAG_CONST (*C++ enumerator*), 41
jsdrv_union_flag_e::JSDRV_UNION_FLAG_HEAP_MEMORY (*C++ enumerator*), 41
jsdrv_union_flag_e::JSDRV_UNION_FLAG_NONE (*C++ enumerator*), 41
jsdrv_union_flag_e::JSDRV_UNION_FLAG_RETAIN (*C++ enumerator*), 41
jsdrv_union_i16 (*C macro*), 39
jsdrv_union_i16_r (*C macro*), 39
jsdrv_union_i32 (*C macro*), 39
jsdrv_union_i32_r (*C macro*), 39
jsdrv_union_i64 (*C macro*), 40
jsdrv_union_i64_r (*C macro*), 40
jsdrv_union_i8 (*C macro*), 39
jsdrv_union_i8_r (*C macro*), 39
jsdrv_union_inner_u (*C++ union*), 43
jsdrv_union_inner_u::bin (*C++ member*), 44
jsdrv_union_inner_u::f32 (*C++ member*), 44
jsdrv_union_inner_u::f64 (*C++ member*), 44
jsdrv_union_inner_u::i16 (*C++ member*), 44
jsdrv_union_inner_u::i32 (*C++ member*), 44
jsdrv_union_inner_u::i64 (*C++ member*), 44
jsdrv_union_inner_u::i8 (*C++ member*), 44
jsdrv_union_inner_u::str (*C++ member*), 44
jsdrv_union_inner_u::u16 (*C++ member*), 44
jsdrv_union_inner_u::u32 (*C++ member*), 44
jsdrv_union_inner_u::u64 (*C++ member*), 44
jsdrv_union_inner_u::u8 (*C++ member*), 44
jsdrv_union_is_type_ptr (*C++ function*), 43
jsdrv_union_json (*C macro*), 40
jsdrv_union_null (*C macro*), 39
jsdrv_union_null_r (*C macro*), 39
jsdrv_union_s (*C++ struct*), 44
jsdrv_union_s::app (*C++ member*), 45
jsdrv_union_s::flags (*C++ member*), 45
jsdrv_union_s::op (*C++ member*), 45
jsdrv_union_s::size (*C++ member*), 45
jsdrv_union_s::type (*C++ member*), 45
jsdrv_union_s::value (*C++ member*), 45
jsdrv_union_str (*C macro*), 40
jsdrv_union_to_bool (*C++ function*), 43
jsdrv_union_type_to_str (*C++ function*), 43
jsdrv_union_u16 (*C macro*), 39
jsdrv_union_u16_r (*C macro*), 39
jsdrv_union_u32 (*C macro*), 39
jsdrv_union_u32_r (*C macro*), 39
jsdrv_union_u64 (*C macro*), 39
jsdrv_union_u64_r (*C macro*), 39
jsdrv_union_u8 (*C macro*), 39
jsdrv_union_u8_r (*C macro*), 39
jsdrv_union_value_to_str (*C++ function*), 43
jsdrv_union_widen (*C++ function*), 42
jsdrv_unsubscribe (*C++ function*), 14
jsdrv_unsubscribe_all (*C++ function*), 14

## L

log_level (*pyjoulescope_driver.Driver attribute*), 47
LogLevel (*class in pyjoulescope_driver*), 49

## M

module
    pyjoulescope_driver, 47
    pyjoulescope_driver.time64, 50

## N

now() (*in module pyjoulescope_driver.time64*), 51

## O

open() (*pyjoulescope_driver.Driver method*), 47

## P

publish() (*pyjoulescope_driver.Driver method*), 48
pyjoulescope_driver
    module, 47
pyjoulescope_driver.time64
    module, 50

## Q

query() (*pyjoulescope_driver.Driver method*), 48

## S

subscribe() (*pyjoulescope_driver.Driver method*), 48

SubscribeFlags (*class in pyjoulescope_driver*), 49

## U

unsubscribe() (*pyjoulescope_driver.Driver method*), 49
unsubscribe_all() (*pyjoulescope_driver.Driver method*), 49