

Generated by Doxygen 1.8.19

1 Module Index	1
1.1 Modules	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Module Documentation	7
4.1 Filter Match Cache	7
4.1.1 Detailed Description	7
4.1.2 Function Documentation	8
4.1.2.1 step_cache_add()	8
4.1.2.2 step_cache_check()	8
4.2 Filter Definitions	10
4.2.1 Detailed Description	10
4.2.2 Enumeration Type Documentation	10
4.2.2.1 step_filter_op	10
4.2.3 Function Documentation	11
4.2.3.1 step_filt_evaluate()	11
4.2.3.2 step_filt_print()	12
4.3 Nodes	13
4.3.1 Detailed Description	13
4.3.2 Typedef Documentation	13
4.3.2.1 step_node_callback_t	13
4.3.2.2 step_node_error_t	14
4.3.2.3 step_node_evaluate_t	14
4.3.2.4 step_node_init_t	15
4.3.3 Function Documentation	16
4.3.3.1 step_node_print()	16
4.4 Processor Node Management	17
4.4.1 Detailed Description	17
4.4.2 Function Documentation	18
4.4.2.1 step_pm_clear()	18
4.4.2.2 step_pm_disable_node()	18
4.4.2.3 step_pm_enable_node()	18
4.4.2.4 step_pm_list()	19
4.4.2.5 step_pm_node_get()	19
4.4.2.6 step_pm_poll()	19
4.4.2.7 step_pm_process()	20
4.4.2.8 step_pm_register()	20
4.4.2.9 step_pm_resume()	21

4.4.2.10 step_pm_suspend()	21
4.5 Sample Pool Management	22
4.5.1 Detailed Description	22
4.5.2 Function Documentation	22
4.5.2.1 step_sp_alloc()	22
4.5.2.2 step_sp_bytes_alloc()	23
4.5.2.3 step_sp_free()	23
4.5.2.4 step_sp_get()	24
4.5.2.5 step_sp_put()	24
4.6 Secure Telemetry Pipeline (STeP) API	25
4.6.1 Detailed Description	25
4.7 Measurements	26
4.7.1 Detailed Description	27
4.7.2 Enumeration Type Documentation	31
4.7.2.1 step_mes_compression	31
4.7.2.2 step_mes_encoding	32
4.7.2.3 step_mes_format	32
4.7.2.4 step_mes_fragment	32
4.7.2.5 step_mes_timestamp	32
4.7.2.6 step_mes_vector_sz	33
4.7.3 Function Documentation	33
4.7.3.1 step_mes_print()	33
4.7.3.2 step_mes_sz_payload()	33
4.7.3.3 step_mes_validate()	34
5 Data Structure Documentation	35
5.1 step_cache_rec Struct Reference	
5.1.1 Detailed Description	35
5.2 step_filter Struct Reference	35
5.2.1 Detailed Description	36
5.2.2 Field Documentation	36
5.2.2.1 ignore_mask	36
5.3 step_filter_chain Struct Reference	36
5.3.1 Detailed Description	37
5.3.2 Field Documentation	37
5.3.2.1 chain	37
5.3.2.2 count	37
5.4 step_measurement Struct Reference	37
5.4.1 Detailed Description	38
5.4.2 Field Documentation	38
5.4.2.1 header	38
5.4.2.2 payload	38

5.4.2.3 reserved	. 38
5.5 step_mes_header Struct Reference	. 39
5.5.1 Detailed Description	. 39
5.5.2 Field Documentation	. 40
5.5.2.1 "@1	. 40
5.5.2.2 "@3	. 40
5.5.2.3 "@5	. 40
5.5.2.4 _rsvd	. 40
5.5.2.5 base_type	. 40
5.5.2.6 compression	. 40
5.5.2.7 ctype	. 40
5.5.2.8 data_format	. 41
5.5.2.9 encoding	. 41
5.5.2.10 ext_type	. 41
5.5.2.11 filter_bits	. 41
5.5.2.12 flags_bits	. 41
5.5.2.13 fragment	. 41
5.5.2.14 len	. 41
5.5.2.15 samples	. 42
5.5.2.16 sourceid	. 42
5.5.2.17 srclen_bits	. 42
5.5.2.18 timestamp	. 42
5.5.2.19 unit_bits	. 42
5.5.2.20 vec_sz	. 42
5.6 step_node Struct Reference	. 43
5.6.1 Detailed Description	. 43
5.7 step_node_callbacks Struct Reference	. 43
5.7.1 Detailed Description	. 44
5.7.2 Field Documentation	. 44
5.7.2.1 evaluate_handler	. 44
5.7.2.2 init_handler	. 44
5.7.2.3 matched_handler	. 44
5.7.2.4 start_handler	. 45
5.8 step_pm_node_record Struct Reference	. 45
5.8.1 Detailed Description	. 45
5.9 step_sp_stats Struct Reference	. 46
5.9.1 Detailed Description	. 46
6 File Documentation	47
6.1 include/step/cache.h File Reference	. 47
6.2 include/step/filter.h File Reference	
6.3 include/step/instrumentation.h File Reference	. 48

6.3.1 Detailed Description	. 48
6.4 include/step/measurement/base.h File Reference	. 49
6.4.1 Detailed Description	. 50
6.4.2 Enumeration Type Documentation	. 50
6.4.2.1 step_mes_type	. 50
6.5 include/step/measurement/ext_color.h File Reference	. 51
6.5.1 Detailed Description	. 51
6.5.2 Enumeration Type Documentation	. 51
6.5.2.1 step_mes_ext_color	. 51
6.6 include/step/measurement/ext_light.h File Reference	. 52
6.6.1 Detailed Description	. 52
6.6.2 Enumeration Type Documentation	. 52
6.6.2.1 step_mes_ext_light	. 53
6.7 include/step/measurement/ext_temperature.h File Reference	. 53
6.7.1 Detailed Description	. 54
6.7.2 Enumeration Type Documentation	. 54
6.7.2.1 step_mes_ext_temperature	. 54
6.8 include/step/measurement/measurement.h File Reference	. 54
6.8.1 Detailed Description	. 56
6.9 include/step/measurement/unit.h File Reference	. 56
6.9.1 Detailed Description	. 58
6.9.2 Enumeration Type Documentation	. 58
6.9.2.1 step_mes_si_scale	. 58
6.9.2.2 step_mes_unit_ctype	. 59
6.9.2.3 step_mes_unit_si	. 60
6.10 include/step/node.h File Reference	. 62
6.11 include/step/proc_mgr.h File Reference	. 63
6.12 include/step/sample_pool.h File Reference	. 63
6.13 include/step/step.h File Reference	. 64
Index	65

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Secure Telemetry Pipeline (STeP) API	25
Filter Match Cache	7
Filter Definitions	10
Nodes	13
Processor Node Management	17
Sample Pool Management	22
Measurements	26

2 Module Index

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

step_cache_rec	
Cache record	35
step_filter	
An individual filter entry	35
step_filter_chain	
A filter chain	36
step_measurement	
Measurement packet wrapper	37
step_mes_header	
Measurement header. All fields in little endian	39
step_node	
Node implementation	43
step_node_callbacks	
Optional callback handlers for nodes	43
step_pm_node_record	
Struct used to represent a step_node in the registry	45
step_sp_stats	
Sample pool statistics	46

Data Structure Index

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

nclude/step/cache.h	. 47
nclude/step/filter.h	. 47
nclude/step/instrumentation.h	
Instrumentation helper macros	. 48
nclude/step/node.h	. 62
nclude/step/proc_mgr.h	. 63
nclude/step/sample_pool.h	. 63
nclude/step/step.h	. 64
nclude/step/measurement/base.h	
Base measurement type definitions	. 49
nclude/step/measurement/ext_color.h	
STEP_MES_TYPE_COLOR extended type definitions	. 51
nclude/step/measurement/ext_light.h	
STEP_MES_TYPE_LIGHT extended type definitions	. 52
nclude/step/measurement/ext_temperature.h	
STEP_MES_TYPE_TEMPERATURE extended type definitions	. 53
nclude/step/measurement/measurement.h	
API header file for STEP measurements	. 54
nclude/step/measurement/unit.h	
SI unit type, ctype and scale definitions	. 56

6 File Index

Chapter 4

Module Documentation

4.1 Filter Match Cache

API header file for a least recently used (LRU) cache.

Files

· file cache.h

Data Structures

• struct step_cache_rec Cache record.

Functions

void step_cache_print (void)

Prints the current cache contents using printk.

void step_cache_print_stats (void)

Prints the current cache statistics using printk.

void step_cache_clear (void)

Clears all existing records from cache memory.

• int step_cache_check (uint32_t filter, uint32_t handle, int *result)

Evaluates the supplied filter and node handle against the cache.

• int step_cache_add (uint32_t filter, uint32_t handle, int result)

Inserts a new record in cache memory. If cache memory is full, the least recently used record will be removed from cache memory to make room for the new record.

4.1.1 Detailed Description

API header file for a least recently used (LRU) cache.

Provides a 'least recently used' cache mechanism for evaluating measurment filter values against processor nodes or node chains. As the cache fills up the least recently used cache record will be removed to make room for a new entry.

4.1.2 Function Documentation

4.1.2.1 step_cache_add()

Inserts a new record in cache memory. If cache memory is full, the least recently used record will be removed from cache memory to make room for the new record.

Parameters

filter	The input filter value to add to cache.
handle	The node handle to add to cache.
result	The evaluation result to add to cache for this filter and handle combination.

Returns

int Zero on normal execution, otherwise a negative error code.

4.1.2.2 step cache check()

Evaluates the supplied filter and node handle against the cache.

This function evaluates the supplied filter and node handle against the cache to determine if there is an existing record that matches the input values. If a record is available in cache memory, the cached results will be assigned to result, and 1 will be returned. If no result is found, result will be set to 0, and 0 will be returned.

Calling this function will update the timestamp associated with any matching record in cache, to ensure that the most frequently accessed values remain in cache.

Parameters

filter	The input filter value to evalute for a match.
handle	The node handle to evaluate for a match.
result	Pointer to the evaluation result. Set to cached result on a match, otherwise 0.

4.1 Filter Match Cache

int 1 if a match was found in cache, otherwise 0.

4.2 Filter Definitions

API header file for the STeP filter engine.

Files

· file filter.h

Data Structures

```
    struct step_filter
        An individual filter entry.

    struct step_filter_chain
        A filter chain.
```

Enumerations

```
    enum step_filter_op {
        STEP_FILTER_OP_IS = 0,
        STEP_FILTER_OP_NOT = 1,
        STEP_FILTER_OP_AND = 2,
        STEP_FILTER_OP_AND_NOT = 3,
        STEP_FILTER_OP_OR = 4,
        STEP_FILTER_OP_OR_NOT = 5,
        STEP_FILTER_OP_XOR = 6 }
```

Logical operand used between the current and previous filter values in a filter chain.

Functions

void step_filt_print (struct step_filter_chain *fc)

Prints details of the supplied filter chain using printk.

• int step_filt_evaluate (struct step_filter_chain *fc, struct step_measurement *mes, int *match)

Evaluates the supplied step_measurement against the step_filter_chain to determine if there is a match.

4.2.1 Detailed Description

API header file for the STeP filter engine.

This module implements the evaluation logic to determine if there is a match between a measurment's filter field and the filter chain associated with a processor node.

4.2.2 Enumeration Type Documentation

4.2.2.1 step_filter_op

```
enum step_filter_op
```

Logical operand used between the current and previous filter values in a filter chain.

Note

The first value in a filter chain MUST be either STEP_FILTER_OP_IS of STEP_FILTER_OP_NOT.

4.2 Filter Definitions

Enumerator

STEP_FILTER_OP_IS	Filter evaluation must be logically true. Solely for use as the first operand in a filter chain.
	Note
	This is functionally identical to STEP_FILTER_AND_IS, with the assumption that the previous value is true.
	The first value in a filter chain MUST be either STEP_FILTER_OP_IS of STEP_FILTER_OP_NOT.
STEP_FILTER_OP_NOT	Filter evaluation must be logically false. Solely for use as the first operand in a filter chain.
	Note
	This is functionally identical to STEP_FILTER_AND_IS, with the assumption that the previous value is true.
	The first value in a filter chain MUST be either STEP_FILTER_OP_IS of STEP_FILTER_OP_NOT.
STEP_FILTER_OP_AND	Previous operand AND current operand must resolve to being true, where the current filter evaluation is logically true. Solely for use in non-initial entries in a filter chain.
STEP_FILTER_OP_AND_NOT	Previous operand AND current operand must resolve to being true, where the current filter evaluation is logically false. Solely for use in non-initial entries in a filter chain.
STEP_FILTER_OP_OR	Previous operand OR current operand must resolve to being true, where the current filter evaluation is logically true. Solely for use in non-initial entries in a filter chain.
STEP_FILTER_OP_OR_NOT	Previous operand OR current operand must resolve to being true, where the current filter evaluation is logically false. Solely for use in non-initial entries in a filter chain.
STEP_FILTER_OP_XOR	Exactly one of the previous operand OR current operand must resolve to being true, where the current filter evaluation is logically true. Solely for use in non-initial entries in a filter chain.

4.2.3 Function Documentation

4.2.3.1 step_filt_evaluate()

Evaluates the supplied step_measurement against the step_filter_chain to determine if there is a match.

Parameters

fc	The step_filter_chain to evalute for a match.
mes	The step_measurement to evaluate a match against.
match	1 if the node's filter chain matches, otherwise 0.

Generated by Doxygen

Returns

int Zero on normal execution, otherwise a negative error code.

4.2.3.2 step_filt_print()

```
void step_filt_print ( struct \ step_filter\_chain \ * \ fc \ )
```

Prints details of the supplied filter chain using printk.

Parameters

```
fc The sdsp_filter_chain to print.
```

Referenced by step_node_print().

4.3 Nodes 13

4.3 Nodes

API header file for STeP processor nodes.

Files

· file node.h

Data Structures

struct step_node_callbacks
 Optional callback handlers for nodes.

struct step_node

Node implementation.

Typedefs

- typedef int(* step_node_init_t) (void *cfg, uint32_t handle, uint32_t inst)
 Init callback prototype for node implementations.
- typedef int(* step_node_callback_t) (struct step_measurement *mes, uint32_t handle, uint32_t inst)

 Generic callback prototype for node implementations.
- typedef bool(* step_node_evaluate_t) (struct step_measurement *mes, uint32_t handle, uint32_t inst)

 Callback prototype for node filter evaluation.
- typedef void(* step_node_error_t) (struct step_measurement *mes, uint32_t handle, uint32_t inst, int error)

 Callback prototype when a node fails to successfully run.

Functions

void step_node_print (struct step_node *node)
 Prints details of the supplied processor node using printk.

4.3.1 Detailed Description

API header file for STeP processor nodes.

Nodes are the main building block in STeP, and encapsulate the logic to 'process' incoming step_measurement packets. Thy can be used individually, or connected together in 'processor chains', where the measurment fed into the first node in the chain is handed off to subsequent nodes for further processing.

The first record in a node chain contains a 'filter' or 'filter chain' that determines if a step_measurement should be processed with this node. Assigning NULL to the filter field means that the node will accept all incoming measurements, otherwise the measurement's filter field will be evaluated against the node's filter chain via the filter evaluation engine.

4.3.2 Typedef Documentation

4.3.2.1 step_node_callback_t

step_node_callback_t

Generic callback prototype for node implementations.

Parameters

mes	Pointer to the step_measurement being used.
handle	The handle of the source node this callback.
inst	step_node instance in a node chain (zero-based).

Returns

0 on success, negative error code on failure

4.3.2.2 step_node_error_t

step_node_error_t

Callback prototype when a node fails to successfully run.

Parameters

mes	Pointer to the step_measurement being used.
handle	The handle of the source node this callback.
inst	step_node instance in a node chain (zero-based).
error	Negative error code produced during node execution.

4.3.2.3 step_node_evaluate_t

step_node_evaluate_t

Callback prototype for node filter evaluation.

Parameters

mes	Pointer to the step_measurement being used.
handle	The handle of the source node this callback.
inst	step_node instance in a node chain (zero-based).

Returns

1 on a match, 0 on match failure.

4.3 Nodes 15

4.3.2.4 step_node_init_t

step_node_init_t

Init callback prototype for node implementations.

Parameters

cfg	Pointer to the config struct/value for this node, if any.
handle	The handle of the source node this callback.
inst	step_node instance in a node chain (zero-based).

Returns

0 on success, negative error code on failure

4.3.3 Function Documentation

4.3.3.1 step_node_print()

```
void step_node_print (
          struct step_node * node )
```

Prints details of the supplied processor node using printk.

Parameters

,	The node to display.
node	I he hode to display
11000	i i i o i i o di o di opidy.

4.4 Processor Node Management

API header file for STEP processor node manager.

Files

· file proc_mgr.h

Functions

int step_pm_register (struct step_node *node, uint16_t pri, uint32_t *handle)

Registers a new processor node.

• struct step_node * step_pm_node_get (uint32_t handle, uint32_t inst)

Gets a pointer to the node or node chain associated with the specified handle.

int step pm resume (void)

Initialises the timer thread used to periodically poll for queued measurements.

int step pm suspend (void)

Stops the timer thread used to periodically poll for queued measurements.

• int step_pm_clear (void)

Clears the registry, and resets the manager to it's default state.

• int step_pm_process (struct step_measurement *mes, int *matches, bool free)

Processes the supplied step_measurement using the current processor node registry, consuming the measurement and optionally releasing it from shared memory when completed.

int step_pm_poll (int *mcnt, bool free)

Polls the sample pool for any incoming step_measurement(s) to process, and processes them on a first in, first processed basis.

• int step_pm_disable_node (uint32_t handle)

Disables a registered processor node.

• int step_pm_enable_node (uint32_t handle)

Enables a registered processor node.

int step pm list (void)

Displays a list of registered processor nodes in the order which they are evaluated (highest to lowest priority).

4.4.1 Detailed Description

API header file for STEP processor node manager.

This module manages the processor node registry, the evaluation of measurements against records in the node registry, and if enabled the polling thread used to retrieve and process any queued measurements.

It ensures that measurements are evaluated against the registry in the correct order, based on the node or node chain's 'priority' field, and that the step_measurement is freed from the sample pool's heap memory once processing is complete (if requested).

Nodes can be inserted, enabled or disabled at run time or at build time. The maximum number of nodes stored in the registry is set via KConfig with the CONFIG STEP PROC MGR NODE LIMIT variable.

The sample rate for thee polling thread that checks the sample pool FIFO for queued messages can be configured via CONFIG_STEP_PROC_MGR_POLL_RATE, setting a value in Hertz. Setting this to 0 disables the polling thread, and measurements will have to be manually processing via a call to step_pm_process

4.4.2 Function Documentation

4.4.2.1 step_pm_clear()

Clears the registry, and resets the manager to it's default state.

Returns

int 0 on success, negative error code on failure.

4.4.2.2 step_pm_disable_node()

Disables a registered processor node.

Parameters

handle	The handle the node has been registered under.
--------	--

Returns

int 0 on success, negative error code on failure.

4.4.2.3 step_pm_enable_node()

Enables a registered processor node.

Parameters

h	andle	The handle the node has been registered under.
		The manage and made soon registered and an

Returns

int 0 on success, negative error code on failure.

4.4.2.4 step_pm_list()

```
int step_pm_list (
     void )
```

Displays a list of registered processor nodes in the order which they are evaluated (highest to lowest priority).

Returns

int

4.4.2.5 step_pm_node_get()

Gets a pointer to the node or node chain associated with the specified handle.

Parameters

handle	The handle the node has been registered under.
inst	The instancee number in a node chain, otherwise 0.

Returns

struct* spd_node Pointer to the associated node or node chain, NULL on error.

4.4.2.6 step_pm_poll()

Polls the sample pool for any incoming step_measurement(s) to process, and processes them on a first in, first processed basis.

Parameters

mcnt	Pointer to the number of samples read from the sample pool.	
free	If set to true (1), the measurement will be freed from shared memory when processing is complete.	

Returns

int 0 on success, negative error code on failure.

4.4.2.7 step_pm_process()

Processes the supplied step_measurement using the current processor node registry, consuming the measurement and optionally releasing it from shared memory when completed.

This function evaluates the measurement's filter value against the filter chain of the first node of each active record in the node registry. If a node matches the measurement's filter value, the appropriate callbacks will be fired in the processor node chain, from top to bottom. Only the first node in a processor node chain is evaluated for a filter match.

When this function completes, the supplied step_message can optionally be freed from shared memory in the sample pool via the free argument.

Parameters

mes	Pointer to the step_measurement to parse.
matches	The number of matches that occured during processing.
free	If set to true (1), the measurement will be freed from shared memory when processing is complete.

Returns

int 0 on success, negative error code on failure.

Referenced by step_pm_poll().

4.4.2.8 step_pm_register()

Registers a new processor node.

This function registers a processor node or node chain, such that it will be evaluated when processing incoming measurements.

Since processor nodes can be destructive, and operate on the input measurement directly, they can be assigned a priority value (pri), where a higher number indicates higher priority. Nodes will be inserted into the linked list from highest to lowest priority, and nodes with the same priority level will be inserted sequentionally in the order they are registered.

Non-destructive nodes should be placed first (at a higher priority), before processing the measurement in destructive nodes later in the processing pipeline.

Parameters

node	The processor node to register with the manager.
pri	The priority level for this node (larger = higher priority).
handle	The handle the node has been registered under. Set to -1 (0xFFFFFFF) if node registration limit has
	been reached.

Returns

int 0 on success, negative error code on failure.

4.4.2.9 step_pm_resume()

```
int step_pm_resume (
     void )
```

Initialises the timer thread used to periodically poll for queued measurements.

Returns

int 0 on success, negative error code on failure.

4.4.2.10 step_pm_suspend()

Stops the timer thread used to periodically poll for queued measurements.

Returns

int 0 on success, negative error code on failure.

4.5 Sample Pool Management

API header file for STEP sample pool.

Files

· file sample_pool.h

Functions

void step_sp_put (struct step_measurement *mes)

Adds the specified step_measurement to the pool's FIFO.

struct step measurement * step sp get (void)

Gets an step_measurement from the pool's FIFO, or NULL if nothing is available.

void step_sp_free (struct step_measurement *mes)

Frees the heap memory associated with 'ds'.

void step_sp_flush (void)

Reads the entire sample pool FIFO, flushing any step_measurement(s) found from heap memory. Use with care!

struct step_measurement * step_sp_alloc (uint16_t sz)

Allocates memory for a step_measurement from the sample pool's heap.

int32 t step sp bytes alloc (void)

Returns the number of bytes currently allocated from the sample pool's heap memory.

void step_sp_print_stats (void)

Prints the contents of the statistics struct. Useful for debug purposes to detect memory leaks, etc.

4.5.1 Detailed Description

API header file for STEP sample pool.

This module provides a means to allocate and free step_measurement instances from a shared memory heap, and to queue measurements for processing in a simple FIFO buffer.

Allocating measurements from heap isn't mandatory, but it does provide a number of benefits with asynchronous processing of measurements, such as efficient use of limited memory, and automatic release of the measurement from memory once processing is complete, meaning the processing state doesn't need to be tracked by the data source.

The heap size is set via KConfig using the CONFIG STEP POOL SIZE property.

4.5.2 Function Documentation

4.5.2.1 step sp alloc()

Allocates memory for a step_measurement from the sample pool's heap.

Parameters

SZ

Payload size in bytes. If the payload contents will be modified, make sure to request the maximum required payload size, including the optional timestamp value.

Returns

A pointer to the measurement, or NULL if sufficient memory could not be allocated from the heap.

4.5.2.2 step_sp_bytes_alloc()

Returns the number of bytes currently allocated from the sample pool's heap memory.

Note

Records must be aligned on an 8-byte boundary with Zephyr's heap implementation, so this value may be larger than expected when unaligned records are allocated from the heap memory pool.

This value does not take into account the memory taken up by the k_heap struct, which also comes from the heap memory allocation. Actual memory available for records is limited to 'CONFIG_STEP_POOL_SIZE - sizeof(struct k_heap)'.

Returns

int32_t The number of bytes currently allocated.

4.5.2.3 step_sp_free()

Frees the heap memory associated with 'ds'.

Parameters

mes

Pointer to the step_measurement whose memory should be freed.

Referenced by step_sp_flush().

4.5.2.4 step_sp_get()

Gets an step_measurement from the pool's FIFO, or NULL if nothing is available.

Returns

A pointer to the measurement, or NULL if no measurement could be retrieved.

Referenced by step_pm_poll(), and step_sp_flush().

4.5.2.5 step_sp_put()

Adds the specified step_measurement to the pool's FIFO.

Parameters

mes The step_measurement to add.

4.6 Secure Telemetry Pipeline (STeP) API

Modules

· Filter Match Cache

API header file for a least recently used (LRU) cache.

· Filter Definitions

API header file for the STeP filter engine.

Nodes

API header file for STeP processor nodes.

• Processor Node Management

API header file for STEP processor node manager.

· Sample Pool Management

API header file for STEP sample pool.

Measurements

API Header file for measurements.

Files

· file step.h

4.6.1 Detailed Description

4.7 Measurements

API Header file for measurements.

Files

· file base.h

Base measurement type definitions.

· file ext_color.h

STEP_MES_TYPE_COLOR extended type definitions.

· file ext light.h

STEP_MES_TYPE_LIGHT extended type definitions.

file ext_temperature.h

STEP_MES_TYPE_TEMPERATURE extended type definitions.

· file measurement.h

API header file for STEP measurements.

· file unit.h

SI unit type, ctype and scale definitions.

Data Structures

· struct step_mes_header

Measurement header. All fields in little endian.

· struct step_measurement

Measurement packet wrapper.

Macros

- #define STEP_MES_MASK_FULL_TYPE_POS (0)
- #define STEP_MES_MASK_FULL_TYPE (0xFFFF << STEP_MES_MASK_FULL_TYPE_POS)
- #define STEP MES MASK BASE TYPE POS (0)
- #define STEP_MES_MASK_BASE_TYPE (0xFF << STEP_MES_MASK_BASE_TYPE_POS)
- #define STEP MES MASK EXT TYPE POS (8)
- #define STEP_MES_MASK_EXT_TYPE (0xFF << STEP_MES_MASK_EXT_TYPE_POS)
- #define STEP MES MASK FLAGS POS (16)
- #define STEP_MES_MASK_FLAGS (0xFFFF << STEP_MES_MASK_FLAGS_POS)
- #define STEP MES MASK FORMAT POS (16)
- #define STEP MES MASK FORMAT (0x7 << STEP MES MASK FORMAT POS)
- #define STEP MES MASK ENCODING POS (19)
- #define STEP_MES_MASK_ENCODING (0xF << STEP_MES_MASK_ENCODING_POS)
- #define STEP_MES_MASK_COMPRESSION_POS (23)
- #define STEP MES MASK COMPRESSION (0x7 << STEP MES MASK COMPRESSION POS)
- #define STEP_MES_MASK_TIMESTAMP_POS (26)
- #define STEP_MES_MASK_TIMESTAMP (0x7 << STEP_MES_MASK_TIMESTAMP_POS)

4.7 Measurements 27

Enumerations

```
enum step_mes_format {
 STEP MES FORMAT NONE = 0,
 STEP MES FORMAT CBOR = 1 }
enum step_mes_encoding {
 STEP MES ENCODING NONE = 0,
 STEP_MES_ENCODING_BASE64 = 1,
 STEP_MES_ENCODING_BASE45 = 2 }
• enum step_mes_compression {
 STEP MES COMPRESSION NONE = 0,
 STEP MES COMPRESSION LZ4 = 1 }
enum step_mes_timestamp {
 STEP_MES_TIMESTAMP_NONE = 0,
 STEP_MES_TIMESTAMP_EPOCH_32 = 1,
 STEP_MES_TIMESTAMP_EPOCH_64 = 2,
 STEP\_MES\_TIMESTAMP\_UPTIME\_MS\_32 = 3,
 STEP MES TIMESTAMP UPTIME MS 64 = 4,
 STEP_MES_TIMESTAMP_UPTIME_US_64 = 5 }
enum step mes fragment {
 STEP_MES_FRAGMENT_NONE = 0,
 STEP_MES_FRAGMENT_PARTIAL = 1,
 STEP MES FRAGMENT FINAL = 2 }
enum step_mes_vector_sz {
 STEP MES VECTOR SZ NONE = 0,
 STEP_MES_VECTOR_SZ_2 = 1,
 STEP MES VECTOR SZ 3 = 2,
 STEP MES VECTOR SZ 4 = 3
```

Functions

• int32_t step_mes_sz_payload (struct step_mes_header *hdr)

Calculates the minimum number of bytes required for the measurement payload, taking into account the timestamp, ctype, sample count and encoding scheme indicated in the supplied header.

int32_t step_mes_validate (struct step_measurement *mes)

Checks the populated step_measurement for common errors, such as the payload length being too small for the minimum payload.

void step_mes_print (struct step_measurement *sample)

Helper function to display the contents of the step_measurement.

4.7.1 Detailed Description

API Header file for measurements.

```
Measurement

-----

Measurements consist of a measurement type (Base Type + Extended Type),
represented in a specific SI unit (SI Unit Type), and implemented in a
specific C type in memory (C Type).
```

There is an option to adjust the measurement's scale in +/- 10°n steps (Scale Factor) from the default SI unit and scale indicated by the SI Unit Type. For example, if 'Ampere' is indicated as the SI unit, the measurement could indicate that the value is in uA by setting the scale factor to -6.

The Filter fields, which indicate the measurement type and certain config flag for the payload, is used to allow measurement consumers to 'subscribe' to samples of interest based on the value(s) present in this word.

Measurements have the following representation in memory:

```
3
        2.
               1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
Flags
           | Ext. M Type | Base M Type | <- Filter
| C Type | Scale Factor | SI Unit Type
                      | <- Unit
Source ID | S Cnt | V | F | Payload Length | <- SrcLen
Timestamp (optional)
Payload
```

```
5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
| Res | TSt | CMP | Encod | DF | <- Flags
1
   +---- Data Format (CBOR, etc.)
    +---- Encoding (BASE64, BASE45, etc.)
  +----- Compression (LZ4, etc.)
    +---- Timestamp
   ----- Reserved (version flag?)
```

Filter

Indicates measurement type for payload parsing, and to enable exact-match or selective filtering of measurements by processor nodes.

o Base Measurement Type [0:7] (Mandatory)

The base measurement type of the sample

o Extended Measurement Type [8:15] (Optional)

The extended measurement type of the sample. Exact meaning is specific to the base measurement type indicated.

o Flags [16:31] (Optional)

4.7 Measurements 29

```
o Data Format [0:2]
    Data format used to organise the payload (if any).
    0 = Raw binary data
        Raw, unformatted binary data using the specified C type.
    1 = CBOR (Concise Binary Object Representation, rfc8949)
        The payload is encoded as a CBOR record(s) (rfc8949), which
        optionally allows the use of COSE (rfc8152) to sign and/or
        encrypt the CBOR record(s). For non-trivial data, this is the
        recommended data format to use for complex use cases.
    2...7 Reserved
o Payload Encoding [3:6]
    Encoding format used for the payload:
    0 = None
    1 = BASE64 encoding
                         Data has been BASE64 encoded
    2 = BASE45 encoding
                         Data has been BASE45 encoded
    3..15 = Reserved
o Compression [7:9]
    Compression algorithm used on the payload:
    0 = None
    1 = LZ4
    2..7 = Reserved
o Timestamp
               [10:12]
    Indicates that a timestamp of the specified format is appended at
    the start of the payload. The length of the timestamp is INCLUDED
    in the packet's 'payload length' field.
    0 = None
    1 = Unix Epoch 32-bit
    2 = Unix Epoch 64-bit
    3 = Uptime in milliseconds, 32-bit
    4 = Uptime in milliseconds, 64-bit
    5 = Uptime in microseconds, 64-bit
    6..7 = Reserved
o Reserved [13:15]
```

Must be set to 0.

Unit

Indicates the SI unit, scale factor and underlying C type used to implement the specified base + extended measurement value.

o SI Unit Type [0:15] (Mandatory)

The base, derived or compound SI unit used to represent this measurement.

o Scale Factor [16:23] (Optional)

An optional 10 $^{\circ}$ n scale factor from the default unit scale represented by the SI Unit Type. If the default SI Unit Type is 'Ampere', for example, a Scale Factor of -3 would indicate that this particular measurement represents mA.

o C Type [24:31] (Mandatory)

The underlying C type used to represent the measurement in memory.

SrcLen

Payload length for the measurement, and the source ID to correlate the measurement with a source in the source registry.

o Payload Length [0:15]

Payload length in bytes, minus the header, including optional timestamp if present.

o Fragment [16:17]

Indicates that this is a packet fragment, and the contents should be appended to the previous packets from this source before being parsed.

o Vector Size [18:19]

Indicates that this is a vector (rather than a scalar) value, composed of $\text{vect_sz} + 1$ components. Since this is only two bits, it is limited to representing small vectors or quaternion-type units containing up to 4 distinct components. Leave at 0 to indicate a scalar value.

o Sample Count [20:23]

If more than one sample is present in the payload, this field can be used to represent the number of samples present. Sample count is represented in 2^n format, except for 0xF, where:

4.7 Measurements 31

If the Sample Count field is set to 0xF (15), the payload contains an arbitrary number of samples, where the sample count is placed as a 32-bit word at the very beginning of the payload (after the optional timestamp if present!), as an unsigned integer in little-endian format.

If more than one sample is present, and the timestamp is enabled, the timestamp value corresponds to the time when the first sample was taken, and the interval between samples will need to be communicated out-of-band.

This field only has meaning when 'Data Format' is set to 0, meaning unformatted data is present in the payload. When a specific data format is used (CBOR, etc.), multiple samples should be indicated using an appropriate mechanism in that data format, and this field should be left at 0 to indicate that only one formatted payload is present.

```
o Source ID [24:31]
```

Source registry ID to correlate measurements with. This allows for additional information about the source driver to be retrieved via an out-of-band channel. This may include min/max value range, sample rate, gain settings, precision, model and driver number, etc.

4.7.2 Enumeration Type Documentation

4.7.2.1 step_mes_compression

enum step_mes_compression

Payload compression algorithm used.

Enumerator

STEP_MES_COMPRESSION_NONE	No payload compression used.
STEP_MES_COMPRESSION_LZ4	LZ4 compression.

32 Module Documentation

4.7.2.2 step_mes_encoding

enum step_mes_encoding

Payload encoding used.

Enumerator

STEP_MES_ENCODING_NONE	No encoding used (binary data).
STEP_MES_ENCODING_BASE64	BASE64 Encoding (rfc4648).
STEP_MES_ENCODING_BASE45	BASE45 Encoding (draft-faltstrom-base45-06).

4.7.2.3 step_mes_format

enum step_mes_format

Payload data structure used.

Enumerator

STEP_MES_FORMAT_NONE	No data structure (raw C type data).
STEP_MES_FORMAT_CBOR	CBOR record(s).

4.7.2.4 step_mes_fragment

enum step_mes_fragment

Packet fragments.

Enumerator

STEP_MES_FRAGMENT_NONE	No a fragment (complete payload).
STEP_MES_FRAGMENT_PARTIAL	Non-final fragment in a larger payload.
STEP_MES_FRAGMENT_FINAL	Final fragment in the larger payload.

4.7.2.5 step_mes_timestamp

enum step_mes_timestamp

Optional timestamp format used.

4.7 Measurements 33

Enumerator

STEP_MES_TIMESTAMP_NONE	No timestamp included.
STEP_MES_TIMESTAMP_EPOCH_32 32-bit Unix epoch timestamp pro	
STEP_MES_TIMESTAMP_EPOCH_64	64-bit Unix epoch timestamp present.
STEP_MES_TIMESTAMP_UPTIME_MS_32 32-bit millisecond device uptime count	
STEP_MES_TIMESTAMP_UPTIME_MS_64 64-bit millisecond device uptime counter	
STEP_MES_TIMESTAMP_UPTIME_US_64	64-bit microsecond device uptime counter.

4.7.2.6 step_mes_vector_sz

```
enum step_mes_vector_sz
```

Vector size.

Enumerator

STEP_MES_VECTOR_SZ_NONE	Scalar value.
STEP_MES_VECTOR_SZ_2	Pair, such as an X,Y co-ordinate.
STEP_MES_VECTOR_SZ_3	
STEP_MES_VECTOR_SZ_4	Vector 4, such as a Quaternion.

4.7.3 Function Documentation

4.7.3.1 step_mes_print()

Helper function to display the contents of the step_measurement.

Parameters

```
sample step_measurement to print.
```

4.7.3.2 step_mes_sz_payload()

```
int32_t step_mes_sz_payload ( struct \ step_mes_header * \textit{hdr} \ )
```

34 Module Documentation

Calculates the minimum number of bytes required for the measurement payload, taking into account the timestamp, ctype, sample count and encoding scheme indicated in the supplied header.

Parameters

hdr Pointer to the step_mes_header to parse.

Returns

int32_t The minimum number of bytes required to represent mes in memory, or -1 if a minimum payload can not be determined.

Note

This function does not currently take into account data format (CBOR, etc.) or compression (LZ4, etc.) when determining the minimum payload size. Be sure to adjust the returned minimum size to provide sufficient overhead in the payload for these features if required.

Referenced by step_mes_validate().

4.7.3.3 step_mes_validate()

Checks the populated step_measurement for common errors, such as the payload length being too small for the minimum payload.

Parameters

msg The populated measurement to validate.

Returns

int32 t 0 if the measurement is valid, otherwise a negative error code.

Chapter 5

Data Structure Documentation

5.1 step_cache_rec Struct Reference

Cache record.

#include <cache.h>

Data Fields

• uint32_t input

The input value to be stored in cache.

• uint32_t handle

The handle that the uncached input value was validated against.

uint32_t result

The results of evaluating the uncached input value against the current record's handle.

int64_t last_used

The time that this record was last used. Required to remove the least recently used record from cache on overflow.

5.1.1 Detailed Description

Cache record.

The documentation for this struct was generated from the following file:

• include/step/cache.h

5.2 step_filter Struct Reference

An individual filter entry.

```
#include <filter.h>
```

Data Fields

enum step_filter_op op

The operand to apply between the current and previous step_filters.

uint32_t match

The measurement's filter value must exactly match this value, taking into account any bits excluded via ignore_mask.

uint32_t ignore_mask

Any bits set to 1 in this mask field will be ignored when determining if an exact match was found.

5.2.1 Detailed Description

An individual filter entry.

5.2.2 Field Documentation

5.2.2.1 ignore_mask

```
uint32_t step_filter::ignore_mask
```

Any bits set to 1 in this mask field will be ignored when determining if an exact match was found.

Note

This can be used to perfom and exact match only on the base and/or extended data type fields, for example.

Referenced by step_filt_print().

The documentation for this struct was generated from the following file:

· include/step/filter.h

5.3 step_filter_chain Struct Reference

A filter chain.

```
#include <filter.h>
```

Data Fields

uint32_t count

The number of filters supplied in 'chain'.

• struct step_filter * chain

Pointer to an array of 'count' len of individual filters.

5.3.1 Detailed Description

A filter chain.

Set 'count' to 0 or 'chain' to NULL to indicate that this is a catch-all filter that should match on any valid incoming measurement

Note

Entries in a filter chain are evaluated in a strictly linear left-to-right (or top-to-bottom) manner, where the sum of the previous operands is evaluated against the current filter entry. There is currently no mechanism to override the order of evaluation via parentheses or operator precedence.

5.3.2 Field Documentation

5.3.2.1 chain

```
struct step_filter* step_filter_chain::chain
```

Pointer to an array of 'count' len of individual filters.

If this value is NULL, it will be interpretted as a catch-all indicator, matching on all measurements.

Referenced by step_filt_evaluate(), and step_filt_print().

5.3.2.2 count

```
uint32_t step_filter_chain::count
```

The number of filters supplied in 'chain'.

If this value is 0, it will be interpretted as a catch-all indicator, matching on all measurements.

Referenced by step_filt_evaluate(), and step_filt_print().

The documentation for this struct was generated from the following file:

· include/step/filter.h

5.4 step_measurement Struct Reference

Measurement packet wrapper.

```
#include <measurement.h>
```

Data Fields

- int * reserved
- struct step_mes_header header
- void * payload

5.4.1 Detailed Description

Measurement packet wrapper.

5.4.2 Field Documentation

5.4.2.1 header

```
struct step_mes_header step_measurement::header
```

Packet header containing filter data, SI unit and payload length.

Referenced by step_mes_print(), step_mes_validate(), step_sp_alloc(), and step_sp_free().

5.4.2.2 payload

void* step_measurement::payload

Payload contents.

Referenced by step_mes_print(), and step_sp_alloc().

5.4.2.3 reserved

```
int* step_measurement::reserved
```

Zephyr's FIFO implementation requires that FIFO entries reserve a single word at the start of the record for kernel use (to add the .next pointer). This is wasteful, but is added here until a better alternative can be found, or 'k_fifo—alloc_put' can be made to work properly.

The documentation for this struct was generated from the following file:

• include/step/measurement/measurement.h

5.5 step mes header Struct Reference

Measurement header. All fields in little endian.

```
#include <measurement.h>
```

Data Fields

```
• union {
    struct {
      uint8_t base_type
      uint8_t ext_type
      union {
         struct {
           uint16_t data_format: 3
           uint16_t encoding: 4
           uint16 t compression: 3
           uint16 t timestamp: 3
           uint16_t rsvd: 3
        } flags
         uint16_t flags_bits
    } filter
    uint32_t filter_bits
 };
• union {
    struct {
      uint16 t si unit
         The SI unit and default scale used for this measurement. Must be a member of step_mes_unit_si.
      int8_t scale_factor
         The amount to scale the measurement value up or down (10^{\land}n), starting from the unit and scale indicated by si unit. Ty
      uint8 t ctype
         The data type that this SI unit is represented by in C. Must be a member of step_mes_unit_ctype.
    } unit
    uint32_t unit_bits
 };
• union {
    struct {
      uint16_t len
      struct {
        uint8_t fragment: 2
        uint8_t vec_sz: 2
        uint8_t samples: 4
      uint8_t sourceid
    } srclen
    uint32_t srclen_bits
 };
```

5.5.1 Detailed Description

Measurement header. All fields in little endian.

5.5.2 Field Documentation

```
5.5.2.1 "@1
union { ... }
Filter (header upper word).
5.5.2.2 "@3
union { ... }
Unit (header middle word).
5.5.2.3 "@5
union { ... }
Src/Len (header lower word).
5.5.2.4 _rsvd
uint16_t step_mes_header::_rsvd
Reserved for future use. Must be set to 0.
5.5.2.5 base_type
uint8_t step_mes_header::base_type
Base measurement type.
5.5.2.6 compression
uint16_t step_mes_header::compression
Compression algorithm used (1 = LZ4).
```

5.5.2.7 ctype

uint8_t step_mes_header::ctype

The data type that this SI unit is represented by in C. Must be a member of step_mes_unit_ctype.

This field can be used to determine how many bytes are required to represent this measurement value, and how to interpret the value in memory.

5.5.2.8 data_format

```
uint16_t step_mes_header::data_format
```

Data format used (1 = CBOR).

5.5.2.9 encoding

```
uint16_t step_mes_header::encoding
```

Payload encoding used (1 = BASE64).

5.5.2.10 ext_type

```
uint8_t step_mes_header::ext_type
```

Extended measurement type (meaning depends on base type).

5.5.2.11 filter_bits

```
uint32_t step_mes_header::filter_bits
```

32-bit representation of all Filter bits.

Referenced by step_mes_print().

5.5.2.12 flags_bits

```
uint16_t step_mes_header::flags_bits
```

Flag bits (cbor, timestamp, etc.).

5.5.2.13 fragment

```
uint8_t step_mes_header::fragment
```

Indicates this is a fragment of a larger packet.

5.5.2.14 len

```
uint16_t step_mes_header::len
```

Payload length, excluding the header, including timestamp.

5.5.2.15 samples

```
uint8_t step_mes_header::samples
```

Sample count (2^{n}) , 0 = 1 sample

5.5.2.16 sourceid

```
uint8_t step_mes_header::sourceid
```

Data source registery ID associated with this sample.

5.5.2.17 srclen bits

```
uint32_t step_mes_header::srclen_bits
```

32-bit representation of all Src/Len bits.

Referenced by step_mes_print().

5.5.2.18 timestamp

```
uint16_t step_mes_header::timestamp
```

Timestamp format (1 = epoch32, 2 = epoch64).

5.5.2.19 unit bits

```
uint32_t step_mes_header::unit_bits
```

32-bit representation of si_unit, ctype and scale_factor.

Referenced by step_mes_print().

5.5.2.20 vec_sz

```
uint8_t step_mes_header::vec_sz
```

Indicates the number of vector units. 0 for scalar.

The documentation for this struct was generated from the following file:

• include/step/measurement/measurement.h

5.6 step node Struct Reference

Node implementation.

#include <node.h>

Data Fields

· char * name

An optional display name for this processor node. Must be NULL-terrminated.

· struct step_filter_chain filters

The filter chain use to determine matches for this node.

struct step_node_callbacks callbacks

Callback handlers for this node.

void * config

Config settings for the node. The exact struct or value(s) defined here are node-specific and implementation defined.

struct step node * next

Next node in the node chain. Set to NULL if none.

5.6.1 Detailed Description

Node implementation.

The callback handlers in this struct are used to implement a node or node chain, and to define the node's filter properties to know when it should be executed.

The documentation for this struct was generated from the following file:

• include/step/node.h

5.7 step node callbacks Struct Reference

Optional callback handlers for nodes.

#include <node.h>

Data Fields

· step node init tinit handler

Callback to fire when the node is first registered. This callback allows the node implementation to perform any initiliasation steps required by subsequent callbacks, or to process incoming measurement data.

• step_node_evaluate_t evaluate_handler

Callback to fire when the node is being evaluated based on its filter chain. This callback allows the node implementation to override filter evaluation in the filter engine, and implement the evaluation in the specified callback.

step_node_evaluate_t matched_handler

Callback to fire when the filter engine has indicated that a match occurred. This function allows the node to implement further filtering on secondary info or parameters, and returning true or false from this function will override the previous 'match' value.

· step node callback t start handler

Callback to fire when the node is triggered. This fires before 'run' has been called, but after the filter engine (or override callbacks) has determined that this node should be executed.

· step node callback t exec handler

Callback to implement the node's execution logic.

• step_node_callback_t stop_handler

Callback to fire when the node has successfully finished execution. This fires after 'run' has terminated.

· step_node_error_t error_handler

Callback to fire when the 'run' command fails.

5.7.1 Detailed Description

Optional callback handlers for nodes.

5.7.2 Field Documentation

5.7.2.1 evaluate_handler

```
step_node_evaluate_t step_node_callbacks::evaluate_handler
```

Callback to fire when the node is being evaluated based on its filter chain. This callback allows the node implementation to override filter evaluation in the filter engine, and implement the evaluation in the specified callback.

Note

Set this to NULL to allow the default filter engine to evaluate whether there is a match or not.

Referenced by step_node_print().

5.7.2.2 init handler

```
step_node_init_t step_node_callbacks::init_handler
```

Callback to fire when the node is first registered. This callback allows the node implementation to perform any initiliasation steps required by subsequent callbacks, or to process incoming measurement data.

Note

Set this to NULL to skip the init callback on node registration.

Referenced by step_node_print(), and step_pm_register().

5.7.2.3 matched_handler

```
step_node_evaluate_t step_node_callbacks::matched_handler
```

Callback to fire when the filter engine has indicated that a match occurred. This function allows the node to implement further filtering on secondary info or parameters, and returning true or false from this function will override the previous 'match' value.

Note

Set this to NULL to accept previous filter evaluation.

Referenced by step_node_print().

5.7.2.4 start_handler

```
step_node_callback_t step_node_callbacks::start_handler
```

Callback to fire when the node is triggered. This fires before 'run' has been called, but after the filter engine (or override callbacks) has determined that this node should be executed.

Note

This isn't the same as 'run', since the node hasn't started executing yet. This callback can be used to prepare the data for processing, or implement any statistical tracking required.

Referenced by step_node_print().

The documentation for this struct was generated from the following file:

• include/step/node.h

5.8 step_pm_node_record Struct Reference

Struct used to represent a step_node in the registry.

Data Fields

```
    sys_snode_t snode
        Singly-linked list node reference.
    struct step_node * node
        Pointer to the step_node to register.
    uint32_t handle
        Handle associated with this processor node in the proc. manager.
    uint16_t priority
        Priority level (larger = higher priority).
    struct {
        uint16_t enabled: 1
    } flags
```

Config flags for this node in the linked list.

5.8.1 Detailed Description

Struct used to represent a step_node in the registry.

The documentation for this struct was generated from the following file:

src/proc_mgr.c

5.9 step_sp_stats Struct Reference

Sample pool statistics.

Data Fields

- int32_t bytes_alloc
- uint32_t bytes_alloc_total
- uint32_t bytes_freed_total
- uint32_t fifo_put_calls
- uint32_t fifo_get_calls
- uint32_t pool_free_calls
- uint32_t pool_flush_calls
- uint32_t pool_alloc_calls

5.9.1 Detailed Description

Sample pool statistics.

The documentation for this struct was generated from the following file:

• src/sample_pool.c

Chapter 6

File Documentation

6.1 include/step/cache.h File Reference

```
#include <step/step.h>
```

Data Structures

struct step_cache_rec
 Cache record.

Functions

void step_cache_print (void)

Prints the current cache contents using printk.

void step_cache_print_stats (void)

Prints the current cache statistics using printk.

void step_cache_clear (void)

Clears all existing records from cache memory.

• int step_cache_check (uint32_t filter, uint32_t handle, int *result)

Evaluates the supplied filter and node handle against the cache.

• int step_cache_add (uint32_t filter, uint32_t handle, int result)

Inserts a new record in cache memory. If cache memory is full, the least recently used record will be removed from cache memory to make room for the new record.

6.2 include/step/filter.h File Reference

```
#include <step/step.h>
#include <step/measurement/measurement.h>
```

Data Structures

```
    struct step_filter
        An individual filter entry.

    struct step_filter_chain
        A filter chain.
```

Enumerations

```
    enum step_filter_op {
        STEP_FILTER_OP_IS = 0,
        STEP_FILTER_OP_NOT = 1,
        STEP_FILTER_OP_AND = 2,
        STEP_FILTER_OP_AND_NOT = 3,
        STEP_FILTER_OP_OR = 4,
        STEP_FILTER_OP_OR_NOT = 5,
        STEP_FILTER_OP_XOR = 6 }
```

Logical operand used between the current and previous filter values in a filter chain.

Functions

void step_filt_print (struct step_filter_chain *fc)
 Prints details of the supplied filter chain using printk.

• int step_filt_evaluate (struct step_filter_chain *fc, struct step_measurement *mes, int *match)

Evaluates the supplied step_measurement against the step_filter_chain to determine if there is a match.

6.3 include/step/instrumentation.h File Reference

Instrumentation helper macros.

```
#include <zephyr.h>
#include <sys/printk.h>
#include <step/step.h>
```

Macros

- #define STEP_INSTR_START(t) do { } while(0);
- #define **STEP_INSTR_STOP**(t) do { } while(0);

6.3.1 Detailed Description

Instrumentation helper macros.

6.4 include/step/measurement/base.h File Reference

Base measurement type definitions.

```
#include <step/step.h>
```

Enumerations

```
enum step_mes_type {
 STEP MES TYPE UNDEFINED = 0x00,
 STEP_MES_TYPE_AREA = 0x10,
 STEP_MES_TYPE_ACCELERATION = 0x11,
 STEP_MES_TYPE_AMPLITUDE = 0x12,
 STEP MES TYPE CAPACITANCE = 0x13,
 STEP_MES_TYPE_COLOR = 0x14,
 STEP MES TYPE COORDINATES = 0x15,
 STEP_MES_TYPE_CURRENT = 0x16,
 STEP MES TYPE DIMENSION = 0x17,
 STEP_MES_TYPE_FREQUENCY = 0x18,
 STEP MES TYPE HUMIDITY = 0x19,
 STEP MES TYPE INDUCTANCE = 0x1A,
 STEP MES TYPE LIGHT = 0x1B,
 STEP_MES_TYPE_MAGNETIC_FIELD = 0x1C,
 STEP\_MES\_TYPE\_MASS = 0x1D,
 STEP_MES_TYPE_MOMENTUM = 0x1E,
 STEP_MES_TYPE_ORIENTATION = 0x1F,
 STEP_MES_TYPE_PHASE = 0x20,
 STEP MES TYPE PRESSURE = 0x21,
 STEP MES TYPE RESISTANCE = 0x22,
 STEP MES TYPE SOUND = 0x23,
 STEP_MES_TYPE_TEMPERATURE = 0x24,
 STEP_MES_TYPE_TIME = 0x25,
 STEP_MES_TYPE_VELOCITY = 0x26,
 STEP_MES_TYPE_VOLTAGE = 0x27,
 STEP_MES_TYPE_VOLUME = 0x28,
 STEP_MES_TYPE_ACIDITY = 0x29,
 STEP MES TYPE CONDUCTIVITY = 0x2A,
 STEP MES TYPE FORCE = 0x2B,
 STEP MES TYPE ENERGY = 0x2C,
 STEP MES TYPE USER 1 = 0xF0,
 STEP_MES_TYPE_USER_2 = 0xF1,
 STEP_MES_TYPE_USER_3 = 0xF2,
 STEP_MES_TYPE_USER_4 = 0xF3,
 STEP_MES_TYPE_USER_5 = 0xF4,
 STEP MES TYPE USER 6 = 0xF5,
 STEP_MES_TYPE_USER_7 = 0xF6,
 STEP_MES_TYPE_USER_8 = 0xF7,
 STEP MES TYPE USER 9 = 0xF8,
 STEP_MES_TYPE_USER_10 = 0xF9,
 STEP_MES_TYPE_USER_11 = 0xFA,
 STEP MES TYPE USER 12 = 0xFB,
 STEP MES TYPE USER 13 = 0xFC,
 STEP MES TYPE USER 14 = 0xFD,
 STEP_MES_TYPE_USER_15 = 0xFE,
 STEP_MES_TYPE_LAST = 0xFF }
```

6.4.1 Detailed Description

Base measurement type definitions.

6.4.2 Enumeration Type Documentation

6.4.2.1 step_mes_type

enum step_mes_type

Base measurement types (8-bit)

Enumerator

OTED MEG TYPE INDEEDIED	
STEP_MES_TYPE_UNDEFINED	Undefined
STEP_MES_TYPE_AREA	STEP_MES_UNIT_SI_METERS_2
STEP_MES_TYPE_ACCELERATION	STEP_MES_UNIT_SI_METER_PER_SECOND_2 (linear, gravity)
STEP_MES_TYPE_AMPLITUDE	STEP_MES_UNIT_SI_INTERVAL
STEP_MES_TYPE_CAPACITANCE	STEP_MES_UNIT_SI_FARAD
STEP_MES_TYPE_COLOR	See extended type
STEP_MES_TYPE_COORDINATES	XY vector?
STEP_MES_TYPE_CURRENT	STEP_MES_UNIT_SI_AMPERE
STEP_MES_TYPE_DIMENSION	STEP_MES_UNIT_SI_METER (length, width, radius, distance, etc.)
STEP_MES_TYPE_FREQUENCY	STEP_MES_UNIT_SI_HERTZ
STEP_MES_TYPE_HUMIDITY	STEP_MES_UNIT_SI_RELATIVE_HUMIDITY
STEP_MES_TYPE_INDUCTANCE	STEP_MES_UNIT_SI_HENRY
STEP_MES_TYPE_LIGHT	STEP_MES_UNIT_SI_LUX
STEP_MES_TYPE_MAGNETIC_FIELD	STEP_MES_UNIT_SI_TESLA
STEP_MES_TYPE_MASS	STEP_MES_UNIT_SI_KILOGRAM
STEP_MES_TYPE_MOMENTUM	Angular, Linear, Inertia
STEP_MES_TYPE_ORIENTATION	XYZ vector
STEP_MES_TYPE_PHASE	STEP_MES_UNIT_SI_RADIAN
STEP_MES_TYPE_PRESSURE	STEP_MES_UNIT_SI_PASCAL
STEP_MES_TYPE_RESISTANCE	STEP_MES_UNIT_SI_OHM
STEP_MES_TYPE_SOUND	STEP_MES_UNIT_SI_HERTZ
STEP_MES_TYPE_TEMPERATURE	STEP_MES_UNIT_SI_DEGREE_CELSIUS
STEP_MES_TYPE_TIME	STEP_MES_UNIT_SI_SECOND
STEP_MES_TYPE_VELOCITY	STEP_MES_UNIT_SI_METERS_3_SECOND
STEP_MES_TYPE_VOLTAGE	STEP_MES_UNIT_SI_VOLT
STEP_MES_TYPE_VOLUME	STEP_MES_UNIT_SI_METERS_3
STEP_MES_TYPE_ACIDITY	STEP_MES_UNIT_SI_PH
STEP_MES_TYPE_CONDUCTIVITY	STEP_MES_UNIT_SI_SIEMENS_PER_METER
STEP_MES_TYPE_FORCE	STEP_MES_UNIT_SI_NEWTON
STEP_MES_TYPE_ENERGY	STEP_MES_UNIT_SI_JOULE
STEP_MES_TYPE_USER_1	User defined 1
STEP_MES_TYPE_USER_2	User defined 2
STEP_MES_TYPE_USER_3	User defined 3

Enumerator

STEP_MES_TYPE_USER_4	User defined 4
STEP_MES_TYPE_USER_5	User defined 5
STEP_MES_TYPE_USER_6	User defined 6
STEP_MES_TYPE_USER_7	User defined 7
STEP_MES_TYPE_USER_8	User defined 8
STEP_MES_TYPE_USER_9	User defined 9
STEP_MES_TYPE_USER_10	User defined 10
STEP_MES_TYPE_USER_11	User defined 11
STEP_MES_TYPE_USER_12	User defined 12
STEP_MES_TYPE_USER_13	User defined 13
STEP_MES_TYPE_USER_14	User defined 14
STEP_MES_TYPE_USER_15	User defined 15
STEP_MES_TYPE_LAST	Reserved

6.5 include/step/measurement/ext_color.h File Reference

STEP_MES_TYPE_COLOR extended type definitions.

#include <step/step.h>

Enumerations

```
    enum step_mes_ext_color {
        STEP_MES_EXT_TYPE_COLOR_UNDEFINED = 0,
        STEP_MES_EXT_TYPE_COLOR_RGBA8 = 0x10,
        STEP_MES_EXT_TYPE_COLOR_RGBA16 = 0x11,
        STEP_MES_EXT_TYPE_COLOR_RGBAF = 0x12,
        STEP_MES_EXT_TYPE_COLOR_CIE1931_XYZ = 0x20,
        STEP_MES_EXT_TYPE_COLOR_CIE1931_XYY = 0x21,
        STEP_MES_EXT_TYPE_COLOR_CIE1960_UCS = 0x22,
        STEP_MES_EXT_TYPE_COLOR_CIE1976_UCS = 0x23,
        STEP_MES_EXT_TYPE_COLOR_CIE1960_CCT = 0x24,
        STEP_MES_EXT_TYPE_COLOR_CIE1960_CCT_DUV = 0x25 }
```

6.5.1 Detailed Description

STEP_MES_TYPE_COLOR extended type definitions.

6.5.2 Enumeration Type Documentation

6.5.2.1 step_mes_ext_color

enum step_mes_ext_color

Extended measuremnt types for STEP_MES_TYPE_COLOR (8-bit).

Enumerator

STEP_MES_EXT_TYPE_COLOR_UNDEFINED	Undefined RGBA
STEP_MES_EXT_TYPE_COLOR_RGBA8	8-bit RGBA
STEP_MES_EXT_TYPE_COLOR_RGBA16	16-bit RGBA
STEP_MES_EXT_TYPE_COLOR_RGBAF	01.0 float32 RGBA
STEP_MES_EXT_TYPE_COLOR_CIE1931_XYZ	CIE1931 XYZ tristimulus
STEP_MES_EXT_TYPE_COLOR_CIE1931_XYY	CIE1931 xyY chromaticity
STEP_MES_EXT_TYPE_COLOR_CIE1960_UCS	CIE1960 UCS chromaticity
STEP_MES_EXT_TYPE_COLOR_CIE1976_UCS	CIE1976 UCS chromaticity
STEP_MES_EXT_TYPE_COLOR_CIE1960_CCT	CIE1960 CCT (Duv = 0)
STEP_MES_EXT_TYPE_COLOR_CIE1960_CCT_DUV	CIE1960 CCT and Duv

6.6 include/step/measurement/ext_light.h File Reference

STEP_MES_TYPE_LIGHT extended type definitions.

#include <step/step.h>

Enumerations

```
    enum step_mes_ext_light {
        STEP_MES_EXT_TYPE_LIGHT_UNDEFINED = 0,
        STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANT_FLUX = 0x10,
        STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANT_INTEN = 0x11,
        STEP_MES_EXT_TYPE_LIGHT_RADIO_IRRADIANCE = 0x12,
        STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANCE = 0x13,
        STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUM_FLUX = 0x20,
        STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUM_INTEN = 0x21,
        STEP_MES_EXT_TYPE_LIGHT_PHOTO_ILLUMINANCE = 0x22,
        STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUMINANCE = 0x23 }
        Extended measurement types for STEP_MES_TYPE_LIGHT (8-bit).
```

6.6.1 Detailed Description

STEP_MES_TYPE_LIGHT extended type definitions.

6.6.2 Enumeration Type Documentation

6.6.2.1 step_mes_ext_light

```
enum step_mes_ext_light
```

Extended measurement types for STEP MES TYPE LIGHT (8-bit).

Radiometric Units (STEP_MES_EXT_TYPE_LIGHT_RADIO_*)

Electromagentic radiation is characterised by radiometric units, which describe the physical properties of light (the number of photons, photon energy, or radiant flux). These units have no relation to human vision. Infrared radiation, for example, is not visible to the human eye (>780 nm) but it clearly exists as a radiometric phenomenon and can be accurately measured, described and analysed for scientifically significant purposes.

Photometric Units (STEP_MES_EXT_TYPE_LIGHT_PHOTO_*)

To characterise light relative to the human eye, we need to use photometric units such as luminous intensity, which represents the light intensity of a source as perceived by the human eye, measured in candela (cd).

Since photometric measurements are limited to what the human eye can perceive, these measurements are restricted to the visible spectrum of 380 nm to 780 nm wavelengths.

One of the most common photometric units is luminous flux, which is measured in lumens (lm). It's even more common derivative is illuminance, which is the luminous flux incident per a specific area. Illuminance is measured in lux (which is equal to lm/m^2 2).

Enumerator

STEP_MES_EXT_TYPE_LIGHT_UNDEFINED	lux
STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANT_FLUX	W
STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANT_INTEN	W/sr
STEP_MES_EXT_TYPE_LIGHT_RADIO_IRRADIANCE	W/m^2
STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANCE	W/(sr m^2)
STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUM_FLUX	lm
STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUM_INTEN	lm/sr or cd
STEP_MES_EXT_TYPE_LIGHT_PHOTO_ILLUMINANCE	lm/m^2 or lux
STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUMINANCE	cd/m^2

6.7 include/step/measurement/ext temperature.h File Reference

STEP MES TYPE TEMPERATURE extended type definitions.

```
#include <step/step.h>
```

Enumerations

```
    enum step_mes_ext_temperature {
        STEP_MES_EXT_TYPE_TEMP_UNDEFINED = 0,
        STEP_MES_EXT_TYPE_TEMP_AMBIENT = 1,
        STEP_MES_EXT_TYPE_TEMP_DIE = 2,
        STEP_MES_EXT_TYPE_TEMP_OBJECT = 3 }
```

6.7.1 Detailed Description

STEP_MES_TYPE_TEMPERATURE extended type definitions.

6.7.2 Enumeration Type Documentation

6.7.2.1 step_mes_ext_temperature

```
enum step_mes_ext_temperature
```

Extended measurement types for STEP MES TYPE TEMPERATURE (8-bit).

Enumerator

STEP_MES_EXT_TYPE_TEMP_UNDEFINED	Undefined temperature
STEP_MES_EXT_TYPE_TEMP_AMBIENT	Ambient temperature
STEP_MES_EXT_TYPE_TEMP_DIE	Die temperature
STEP_MES_EXT_TYPE_TEMP_OBJECT	Object temperature

6.8 include/step/measurement/measurement.h File Reference

API header file for STEP measurements.

```
#include <step/step.h>
#include <step/measurement/base.h>
#include <step/measurement/ext_color.h>
#include <step/measurement/ext_light.h>
#include <step/measurement/ext_temperature.h>
#include <step/measurement/unit.h>
```

Data Structures

struct step_mes_header

Measurement header. All fields in little endian.

struct step_measurement

Measurement packet wrapper.

Macros

- #define STEP_MES_MASK_FULL_TYPE_POS (0)
- #define STEP_MES_MASK_FULL_TYPE (0xFFFF << STEP_MES_MASK_FULL_TYPE_POS)
- #define STEP_MES_MASK_BASE_TYPE_POS (0)

#define STEP_MES_MASK_TIMESTAMP_POS (26)

```
#define STEP_MES_MASK_BASE_TYPE (0xFF << STEP_MES_MASK_BASE_TYPE_POS)</li>
#define STEP_MES_MASK_EXT_TYPE_POS (8)
#define STEP_MES_MASK_EXT_TYPE (0xFF << STEP_MES_MASK_EXT_TYPE_POS)</li>
#define STEP_MES_MASK_FLAGS_POS (16)
#define STEP_MES_MASK_FLAGS (0xFFFF << STEP_MES_MASK_FLAGS_POS)</li>
#define STEP_MES_MASK_FORMAT_POS (16)
#define STEP_MES_MASK_FORMAT (0x7 << STEP_MES_MASK_FORMAT_POS)</li>
#define STEP_MES_MASK_ENCODING_POS (19)
#define STEP_MES_MASK_ENCODING (0xF << STEP_MES_MASK_ENCODING_POS)</li>
#define STEP_MES_MASK_COMPRESSION_POS (23)
#define STEP_MES_MASK_COMPRESSION (0x7 << STEP_MES_MASK_COMPRESSION_POS)</li>
```

• #define **STEP_MES_MASK_TIMESTAMP** (0x7 << STEP_MES_MASK_TIMESTAMP_POS)

Enumerations

```
enum step mes format {
 STEP MES FORMAT_NONE = 0,
 STEP MES FORMAT CBOR = 1 }
enum step mes encoding {
 STEP MES ENCODING NONE = 0,
 STEP_MES_ENCODING_BASE64 = 1,
 STEP_MES_ENCODING_BASE45 = 2 }
• enum step_mes_compression {
 STEP MES COMPRESSION NONE = 0,
 STEP MES COMPRESSION LZ4 = 1 }
enum step mes timestamp {
 STEP MES TIMESTAMP NONE = 0,
 STEP_MES_TIMESTAMP_EPOCH_32 = 1,
 STEP_MES_TIMESTAMP_EPOCH_64 = 2,
 STEP_MES_TIMESTAMP_UPTIME_MS_32 = 3,
 STEP_MES_TIMESTAMP_UPTIME_MS_64 = 4,
 STEP_MES_TIMESTAMP_UPTIME_US_64 = 5 }
enum step mes fragment {
 STEP MES FRAGMENT_NONE = 0,
 STEP MES FRAGMENT PARTIAL = 1,
 STEP MES FRAGMENT FINAL = 2 }
• enum step mes vector sz {
 STEP_MES_VECTOR_SZ_NONE = 0,
 STEP_MES_VECTOR_SZ_2 = 1,
 STEP MES VECTOR SZ 3 = 2,
 STEP_MES_VECTOR_SZ_4 = 3 }
```

Functions

int32_t step_mes_sz_payload (struct step_mes_header *hdr)

Calculates the minimum number of bytes required for the measurement payload, taking into account the timestamp, ctype, sample count and encoding scheme indicated in the supplied header.

int32_t step_mes_validate (struct step_measurement *mes)

Checks the populated step_measurement for common errors, such as the payload length being too small for the minimum payload.

void step_mes_print (struct step_measurement *sample)

Helper function to display the contents of the step_measurement.

6.8.1 Detailed Description

API header file for STEP measurements.

6.9 include/step/measurement/unit.h File Reference

SI unit type, ctype and scale definitions.

```
#include <step/step.h>
```

Enumerations

```
• enum step mes unit ctype {
 STEP MES UNIT CTYPE UNDEFINED = 0x00,
 STEP MES UNIT CTYPE IEEE754 FLOAT32 = 0x10,
 STEP MES UNIT CTYPE IEEE754 FLOAT64 = 0x11,
 STEP MES UNIT CTYPE IEEE754 FLOAT128 = 0x12,
 STEP_MES_UNIT_CTYPE_S8 = 0x13,
 STEP_MES_UNIT_CTYPE_S16 = 0x14,
 STEP_MES_UNIT_CTYPE_S32 = 0x15,
 STEP_MES_UNIT_CTYPE_S64 = 0x16,
 STEP_MES_UNIT_CTYPE_S128 = 0x17,
 STEP MES UNIT CTYPE U8 = 0x18,
 STEP MES UNIT CTYPE U16 = 0x19,
 STEP MES UNIT CTYPE U32 = 0x1A,
 STEP MES UNIT CTYPE U64 = 0x1B,
 STEP MES UNIT CTYPE U128 = 0x1C,
 STEP MES UNIT CTYPE BOOL = 0x1D,
 STEP_MES_UNIT_CTYPE_COMPLEX_32 = 0x30,
 STEP_MES_UNIT_CTYPE_COMPLEX_64 = 0x31,
 STEP_MES_UNIT_CTYPE_RANG_UNIT_INTERVAL_32 = 0x80,
 STEP MES UNIT CTYPE RANG UNIT INTERVAL 64 = 0x81,
 STEP MES UNIT CTYPE RANG PERCENT 32 = 0x82,
 STEP MES UNIT CTYPE RANG PERCENT 64 = 0x83,
 STEP_MES_UNIT_CTYPE_USER_1 = 0xF0,
 STEP MES UNIT CTYPE USER 2 = 0xF1,
 STEP_MES_UNIT_CTYPE_USER_3 = 0xF2,
 STEP_MES_UNIT_CTYPE_USER_4 = 0xF3,
 STEP_MES_UNIT_CTYPE_USER_5 = 0xF4,
 STEP MES UNIT CTYPE USER 6 = 0xF5,
 STEP_MES_UNIT_CTYPE_USER_7 = 0xF6,
 STEP_MES_UNIT_CTYPE_USER_8 = 0xF7,
 STEP MES UNIT CTYPE USER 9 = 0xF8,
 STEP MES UNIT CTYPE USER 10 = 0xF9,
 STEP_MES_UNIT_CTYPE_USER_11 = 0xFA,
 STEP MES UNIT CTYPE USER 12 = 0xFB,
 STEP MES UNIT CTYPE USER 13 = 0xFC,
 STEP MES UNIT CTYPE USER 14 = 0xFD,
 STEP_MES_UNIT_CTYPE_USER_15 = 0xFE,
 STEP_MES_UNIT_CTYPE_MAX = 0xFF }
```

C type used to represent a measurement in memory (8-bits).

enum step mes unit si { STEP MES UNIT SI UNDEFINED = 0, STEP MES UNIT SI AMPERE = 0x10, STEP MES UNIT SI CANDELA = 0x11, STEP_MES_UNIT_SI_KELVIN = 0x12, STEP_MES_UNIT_SI_KILOGRAM = 0x13, STEP_MES_UNIT_SI_METER = 0x14, STEP MES UNIT SI MOLE = 0x15, STEP MES UNIT SI SECOND = 0x16. STEP_MES_UNIT_SI_BECQUEREL = 0x20, STEP MES UNIT SI COULOMB = 0x21, STEP MES UNIT SI DEGREE CELSIUS = 0x22, $STEP_MES_UNIT_SI_FARAD = 0x23,$ $STEP_MES_UNIT_SI_GRAY = 0x24,$ STEP MES UNIT SI HENRY = 0x25, STEP MES UNIT SI HERTZ = 0x26, STEP MES UNIT SI JOULE = 0x27, STEP MES UNIT SI KATAL = 0x28, STEP MES UNIT SI LUMEN = 0x29, STEP MES UNIT SI LUX = 0x2A, STEP_MES_UNIT_SI_NEWTON = 0x2B, STEP MES UNIT SI OHM = 0x2C, STEP MES UNIT SI PASCAL = 0x2D, STEP MES UNIT SI RADIAN = 0x2E, STEP_MES_UNIT_SI_SIEMENS = 0x2F, STEP_MES_UNIT_SI_SIEVERT = 0x30, STEP_MES_UNIT_SI_STERADIAN = 0x31, STEP_MES_UNIT_SI_TESLA = 0x32, STEP_MES_UNIT_SI_VOLT = 0x33, STEP MES UNIT SI WATT = 0x34, STEP MES UNIT SI WEBER = 0x35, STEP MES UNIT SI PERCENT = 0x80. STEP_MES_UNIT_SI_INTERVAL = 0x81, STEP MES UNIT SI METERS 2 = 0x1000, STEP MES UNIT SI METER PER SECOND 2 = 0x1100, STEP_MES_UNIT_SI_RELATIVE_HUMIDITY = 0x1900, STEP_MES_UNIT_SI_CANDELA_PER_METER_2 = 0x1B00, STEP MES UNIT SI JOULE PER METER 2 = 0x1B01, STEP MES UNIT SI JOULE PER METER 2 PER HZ = 0x1B02, STEP MES UNIT SI JOULE PER METER 2 PER NM = 0x1B03, STEP MES UNIT SI JOULE PER METER 3 = 0x1B04, STEP MES UNIT SI LUMEN PER METER 2 = 0x1B05, STEP MES UNIT SI LUMEN PER WATT = 0x1B06, STEP_MES_UNIT_SI_LUMEN_SECOND = 0x1B07, STEP_MES_UNIT_SI_LUMEN_SECOND_PER_METER_3 = 0x1B08, $STEP_MES_UNIT_SI_LUX_SECOND = 0x1B09,$ STEP MES UNIT SI WATTS PER HERTZ = 0x1B0A, $STEP_MES_UNIT_SI_WATTS_PER_METER_2 = 0x1B0B,$ STEP_MES_UNIT_SI_WATTS_PER_METER_2_PER_HZ = 0x1B0C, STEP MES UNIT SI WATTS PER METER 2 PER NM = 0x1B0D, STEP MES UNIT SI WATTS PER NM = 0x1B0E, STEP_MES_UNIT_SI_WATTS_PER_STERADIAN = 0x1B0F, STEP MES UNIT SI WATTS PER STERADIAN PER HERTZ = 0x1B10, STEP MES UNIT SI WATTS PER STERADIAN PER METER 2 = 0x1B11, STEP MES UNIT SI WATTS PER STERADIAN PER METER 2 PER HZ = 0x1B12, STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_METER_2_PER_NM = 0x1B13, STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_NM = 0x1B14,

```
STEP_MES_UNIT_SI_MICROTESLA = 0x1C00,
 STEP_MES_UNIT_SI_GRAMS = 0x1D00,
 STEP\_MES\_UNIT\_SI\_HECTOPASCAL = 0x2100,
 STEP_MES_UNIT_SI_METERS_3_SECOND = 0x2601,
 STEP_MES_UNIT_SI_MILLIVOLTS = 0x2700,
 STEP MES UNIT SI METERS 3 = 0x2800,
 STEP MES UNIT SI PH = 0x2900,
 STEP MES UNIT SI SIEMENS PER METER = 0x2A00,
 STEP MES UNIT SI USER DEFINED 1 = 0xFF00,
 STEP MES UNIT SI USER DEFINED 255 = 0xFFFE,
 STEP_MES_UNIT_SI_MAX = 0xFFFF }
    Standard SI units (16-bits).
• enum step mes si scale {
 STEP MES SI SCALE YOTTA = 24,
 STEP MES SI SCALE ZETTA = 21,
 STEP MES SI SCALE EXA = 18,
 STEP_MES_SI_SCALE_PETA = 15,
 STEP_MES_SI_SCALE_TERA = 12,
 STEP_MES_SI_SCALE_GIGA = 9,
 STEP_MES_SI_SCALE_MEGA = 6,
 STEP_MES_SI_SCALE_KILO = 3,
 STEP_MES_SI_SCALE_HECTO = 2,
 STEP_MES_SI_SCALE_DECA = 1,
 STEP MES SI SCALE NONE = 0,
 STEP_MES_SI_SCALE_DECI = -1,
 STEP_MES_SI_SCALE_CENTI = -2,
 STEP MES SI SCALE MILLI = -3,
 STEP MES SI SCALE MICRO = -6,
 STEP_MES_SI_SCALE_NANO = -9,
 STEP_MES_SI_SCALE_PICO = -12,
 STEP MES SI SCALE FEMTO = -15,
 STEP_MES_SI_SCALE_ATTO = -18,
 STEP_MES_SI_SCALE_ZEPTO = -21,
 STEP MES SI SCALE YOCTO = -24 }
```

Standard SI scales/powers.

6.9.1 Detailed Description

SI unit type, ctype and scale definitions.

6.9.2 Enumeration Type Documentation

6.9.2.1 step_mes_si_scale

enum step_mes_si_scale

Standard SI scales/powers.

Enumerator

STEP_MES_SI_SCALE_YOTTA	Y: Septillion
STEP_MES_SI_SCALE_ZETTA	Z: Sextillion
STEP_MES_SI_SCALE_EXA	E: Quintillion
STEP_MES_SI_SCALE_PETA	P: Quadrillion
STEP_MES_SI_SCALE_TERA	T: Trillion
STEP_MES_SI_SCALE_GIGA	G: Billion
STEP_MES_SI_SCALE_MEGA	M: Million
STEP_MES_SI_SCALE_KILO	k: Thousand
STEP_MES_SI_SCALE_HECTO	h: Hundred
STEP_MES_SI_SCALE_DECA	da: Ten
STEP_MES_SI_SCALE_NONE	One
STEP_MES_SI_SCALE_DECI	d: Tenth
STEP_MES_SI_SCALE_CENTI	c: Hundredth
STEP_MES_SI_SCALE_MILLI	m: Thousandth
STEP_MES_SI_SCALE_MICRO	u: Millionth
STEP_MES_SI_SCALE_NANO	n: Billionth
STEP_MES_SI_SCALE_PICO	p: Trillionth
STEP_MES_SI_SCALE_FEMTO	f: Quadrillionth
STEP_MES_SI_SCALE_ATTO	a: Quintillionth
STEP_MES_SI_SCALE_ZEPTO	z: Sextillionth
STEP_MES_SI_SCALE_YOCTO	y: Septillionth

6.9.2.2 step_mes_unit_ctype

```
\verb"enum step_mes_unit_ctype"
```

C type used to represent a measurement in memory (8-bits).

Note

All types are little-endian.

Memory map:

- 0 = Undefined
- 0x10..0x4F = Standard C types
- 0x50..0x7F = Reserved
- 0x80..0x8F = Range types (unit interval, percent, etc.)
- 0x90..0xEF = Reserved
- 0xF0..0xFE = User-defined types

Enumerator

STEP_MES_UNIT_CTYPE_RANG_UNIT_INTER↔	01.0 inclusive,
VAL_32	STEP_MES_UNIT_CTYPE_FLOAT32.
STEP_MES_UNIT_CTYPE_RANG_UNIT_INTER↔	01.0 inclusive,
VAL_64	STEP_MES_UNIT_CTYPE_FLOAT64.
STEP_MES_UNIT_CTYPE_RANG_PERCENT_32	0.0100.0 inclusive,
	STEP_MES_UNIT_CTYPE_FLOAT32.
STEP_MES_UNIT_CTYPE_RANG_PERCENT_64	0.0100.0 inclusive,
	STEP_MES_UNIT_CTYPE_FLOAT64.

6.9.2.3 step_mes_unit_si

enum step_mes_unit_si

Standard SI units (16-bits).

Note

Base and derived SI units can be represented in 8-bits, while combined units require 16-bits to represent.

Memory map:

- 0 = Undefined unit
- 0x0001..0x000F = Reserved
- 0x0010..0x001F = SI base units
- 0x0020..0x003F = SI derived units
- 0x0040..0x007F = Reserved
- 0x0080..0x008F = Unitless values (percent, interval, etc.)
- 0x0090..0x00FF = Reserved
- 0x0100..0x0FFF = SI combined units (generic)
- 0x1000..0w7FFF = Base type specific units, in groups of 0xFF.
- 0x8000..0xFEFF = Reserved
- 0xFF00..0xFFFE = User defined units

Enumerator

STEP_MES_UNIT_SI_AMPERE	A, electric current.
STEP_MES_UNIT_SI_CANDELA	cd, Luminous intensity
STEP_MES_UNIT_SI_KELVIN	K, thermodynamic temp.
STEP_MES_UNIT_SI_KILOGRAM	kg, mass
STEP_MES_UNIT_SI_METER	m, length
STEP_MES_UNIT_SI_MOLE	mol, Amount of substance
STEP_MES_UNIT_SI_SECOND	s, time

Enumerator

STEP_MES_UNIT_SI_BECQUEREL	Bq, radionucl. activity, 1/s.
STEP_MES_UNIT_SI_COULOMB	C, electric charge, A*s.
STEP_MES_UNIT_SI_DEGREE_CELSIUS	Degrees C, Celsius temp, K.
STEP_MES_UNIT_SI_FARAD	F, elec. capaticance, C/V or s^4*A^2/m^2*kg .
STEP_MES_UNIT_SI_GRAY	Gy, Absorbed dose, J/kg or m^2/s^2.
STEP_MES_UNIT_SI_HENRY	H, electric inductance, Wb/A or kg*m ² /s ² *A ² .
STEP_MES_UNIT_SI_HERTZ	Hz, frequency, 1/s.
STEP_MES_UNIT_SI_JOULE	J, energy, work, N*m or kg*m^2/s^2.
STEP_MES_UNIT_SI_KATAL	kat, Catamytic activ., mol/s
STEP_MES_UNIT_SI_LUMEN	Im, Luminous flux, cd*sr
STEP_MES_UNIT_SI_LUX	Ix, illuminance, Im/m^2
STEP_MES_UNIT_SI_NEWTON	N, force, m∗kg/s ² .
STEP_MES_UNIT_SI_OHM	electric resistence, V/A or kg*m^2/s^3*A^2
STEP_MES_UNIT_SI_PASCAL	Pa, pressure, stress, N/m ² or kg/m*s ² .
STEP_MES_UNIT_SI_RADIAN	rad, plane angle, m/m
STEP_MES_UNIT_SI_SIEMENS	S, electric conductance, A/V.
STEP_MES_UNIT_SI_SIEVERT	Sv, Dose equivalent, J/kg or m ² /s ² .
STEP_MES_UNIT_SI_STERADIAN	sr, solid angle, m^2/m^2
STEP_MES_UNIT_SI_TESLA	T, magn. flux dens., Wb/m^2 or kg/A*s^2.
STEP_MES_UNIT_SI_VOLT	V, elec. poten. diff, W/A or kg*m^2/A*s^3.
STEP_MES_UNIT_SI_WATT	W, power, radiant flux, J/s or kg*m^2/s^3.
STEP MES UNIT SI WEBER	Wb, magnetic flux, V*s or kg*m^2/s^2*A.
STEP MES UNIT SI PERCENT	%, 0.0 100.0 inclusive.
STEP_MES_UNIT_SI_INTERVAL	0.0 1.0 inclusive.
STEP_MES_UNIT_SI_METERS_2	m^2
STEP MES UNIT SI METER PER SECOND 2	m/s^2
STEP_MES_UNIT_SI_RELATIVE_HUMIDITY	Percent.
STEP_MES_UNIT_SI_CANDELA_PER_METER_2	cd/m^2
STEP_MES_UNIT_SI_JOULE_PER_METER_2	J/m^2.
STEP_MES_UNIT_SI_JOULE_PER_METER_2_P↔ ER_HZ	J/m^2/Hz.
STEP_MES_UNIT_SI_JOULE_PER_METER_2_P↔ ER_NM	J/m^2/nm.
STEP_MES_UNIT_SI_JOULE_PER_METER_3	Energy density, J/m^3.
STEP_MES_UNIT_SI_LUMEN_PER_METER_2	lm/m [^] 2
STEP_MES_UNIT_SI_LUMEN_PER_WATT	lm/W
STEP_MES_UNIT_SI_LUMEN_SECOND	Im s, AKA talbot
STEP_MES_UNIT_SI_LUMEN_SECOND_PER_M↔ ETER_3	Im s/m^3
STEP_MES_UNIT_SI_LUX_SECOND	lx s
STEP_MES_UNIT_SI_WATTS_PER_HERTZ	W/Hz.
STEP_MES_UNIT_SI_WATTS_PER_METER_2	W/m^2.
STEP_MES_UNIT_SI_WATTS_PER_METER_2_ PER_HZ	W/m^2/Hz.
STEP_MES_UNIT_SI_WATTS_PER_METER_2_ PER_NM	W/m^2/nm.
STEP_MES_UNIT_SI_WATTS_PER_NM	W/nm.

Enumerator

STEP_MES_UNIT_SI_WATTS_PER_STERADIAN	W/sr.
STEP_MES_UNIT_SI_WATTS_PER_STERADIA↔ N_PER_HERTZ	W/sr/Hz.
STEP_MES_UNIT_SI_WATTS_PER_STERADIA↔ N_PER_METER_2	W/sr/m^2.
STEP_MES_UNIT_SI_WATTS_PER_STERADIA↔ N_PER_METER_2_PER_HZ	W/sr/m^2/Hz.
STEP_MES_UNIT_SI_WATTS_PER_STERADIA↔ N_PER_METER_2_PER_NM	W/sr/m^2/nm.
STEP_MES_UNIT_SI_WATTS_PER_STERADIA↔ N_PER_NM	W/sr/nm.
STEP_MES_UNIT_SI_MICROTESLA	uT
STEP_MES_UNIT_SI_GRAMS	g
STEP_MES_UNIT_SI_HECTOPASCAL	hPA
STEP_MES_UNIT_SI_METERS_3_SECOND	m^3/s, flow rate.
STEP_MES_UNIT_SI_MILLIVOLTS	mV
STEP_MES_UNIT_SI_METERS_3	m^3
STEP_MES_UNIT_SI_PH	pH level (not actually a unit, shhh!)
STEP_MES_UNIT_SI_SIEMENS_PER_METER	S/m.

6.10 include/step/node.h File Reference

```
#include <step/step.h>
#include <step/filter.h>
#include <step/measurement/measurement.h>
```

Data Structures

• struct step_node_callbacks

Optional callback handlers for nodes.

struct step_node

Node implementation.

Typedefs

- typedef int(* step_node_init_t) (void *cfg, uint32_t handle, uint32_t inst)

 Init callback prototype for node implementations.
- typedef int(* step_node_callback_t) (struct step_measurement *mes, uint32_t handle, uint32_t inst)

 Generic callback prototype for node implementations.
- typedef bool(* step_node_evaluate_t) (struct step_measurement *mes, uint32_t handle, uint32_t inst)

 Callback prototype for node filter evaluation.
- typedef void(* step_node_error_t) (struct step_measurement *mes, uint32_t handle, uint32_t inst, int error)

 Callback prototype when a node fails to successfully run.

Functions

void step_node_print (struct step_node *node)

Prints details of the supplied processor node using printk.

6.11 include/step/proc_mgr.h File Reference

```
#include <step/step.h>
#include <step/node.h>
```

Functions

int step_pm_register (struct step_node *node, uint16_t pri, uint32_t *handle)

Registers a new processor node.

struct step_node * step_pm_node_get (uint32_t handle, uint32_t inst)

Gets a pointer to the node or node chain associated with the specified handle.

int step pm resume (void)

Initialises the timer thread used to periodically poll for queued measurements.

int step_pm_suspend (void)

Stops the timer thread used to periodically poll for queued measurements.

int step_pm_clear (void)

Clears the registry, and resets the manager to it's default state.

• int step_pm_process (struct step_measurement *mes, int *matches, bool free)

Processes the supplied step_measurement using the current processor node registry, consuming the measurement and optionally releasing it from shared memory when completed.

• int step_pm_poll (int *mcnt, bool free)

Polls the sample pool for any incoming step_measurement(s) to process, and processes them on a first in, first processed basis.

• int step_pm_disable_node (uint32_t handle)

Disables a registered processor node.

• int step_pm_enable_node (uint32_t handle)

Enables a registered processor node.

• int step_pm_list (void)

Displays a list of registered processor nodes in the order which they are evaluated (highest to lowest priority).

6.12 include/step/sample_pool.h File Reference

```
#include <step/step.h>
#include <step/measurement/measurement.h>
```

Functions

void step_sp_put (struct step_measurement *mes)

Adds the specified step measurement to the pool's FIFO.

struct step_measurement * step_sp_get (void)

Gets an step_measurement from the pool's FIFO, or NULL if nothing is available.

void step_sp_free (struct step_measurement *mes)

Frees the heap memory associated with 'ds'.

void step_sp_flush (void)

Reads the entire sample pool FIFO, flushing any step_measurement(s) found from heap memory. Use with care!

struct step_measurement * step_sp_alloc (uint16_t sz)

Allocates memory for a step_measurement from the sample pool's heap.

int32_t step_sp_bytes_alloc (void)

Returns the number of bytes currently allocated from the sample pool's heap memory.

void step_sp_print_stats (void)

Prints the contents of the statistics struct. Useful for debug purposes to detect memory leaks, etc.

6.13 include/step/step.h File Reference

#include <stdint.h>

Index

```
_rsvd
                                                STEP_MES_TYPE_VOLUME, 50
   step_mes_header, 40
                                            base type
                                                step mes header, 40
base.h
   step mes type, 50
                                            chain
    STEP MES TYPE ACCELERATION, 50
                                                step_filter_chain, 37
   STEP_MES_TYPE_ACIDITY, 50
                                            compression
   STEP_MES_TYPE_AMPLITUDE, 50
                                                step mes header, 40
   STEP_MES_TYPE_AREA, 50
                                            count
   STEP MES TYPE CAPACITANCE, 50
                                                step_filter_chain, 37
   STEP_MES_TYPE_COLOR, 50
                                            ctype
   STEP MES TYPE CONDUCTIVITY, 50
                                                step_mes_header, 40
    STEP MES TYPE COORDINATES, 50
   STEP_MES_TYPE_CURRENT, 50
                                            data_format
   STEP_MES_TYPE_DIMENSION, 50
                                                step_mes_header, 40
   STEP_MES_TYPE_ENERGY, 50
   STEP MES TYPE FORCE, 50
                                            encoding
   STEP MES TYPE FREQUENCY, 50
                                                step mes header, 41
   STEP_MES_TYPE_HUMIDITY, 50
                                            evaluate_handler
   STEP_MES_TYPE_INDUCTANCE, 50
                                                step node callbacks, 44
   STEP MES TYPE LAST, 51
                                            ext_color.h
   STEP MES TYPE LIGHT, 50
                                                step_mes_ext_color, 51
   STEP MES TYPE MAGNETIC FIELD, 50
                                                STEP_MES_EXT_TYPE_COLOR_CIE1931_XYY,
   STEP MES TYPE MASS, 50
   STEP MES TYPE MOMENTUM, 50
                                                STEP_MES_EXT_TYPE_COLOR_CIE1931_XYZ,
   STEP_MES_TYPE_ORIENTATION, 50
                                                    52
    STEP MES TYPE PHASE, 50
                                                STEP MES EXT TYPE COLOR CIE1960 CCT,
    STEP MES TYPE PRESSURE, 50
   STEP_MES_TYPE_RESISTANCE, 50
                                                STEP_MES_EXT_TYPE_COLOR_CIE1960_CCT_DUV,
   STEP_MES_TYPE_SOUND, 50
   STEP_MES_TYPE_TEMPERATURE, 50
                                                STEP_MES_EXT_TYPE_COLOR_CIE1960_UCS,
   STEP_MES_TYPE_TIME, 50
   STEP_MES_TYPE_UNDEFINED, 50
                                                    52
                                                STEP_MES_EXT_TYPE_COLOR_CIE1976_UCS,
   STEP_MES_TYPE_USER_1, 50
   STEP_MES_TYPE_USER_10, 51
                                                STEP_MES_EXT_TYPE_COLOR_RGBA16, 52
   STEP MES TYPE USER 11,51
                                                STEP_MES_EXT_TYPE_COLOR_RGBA8, 52
   STEP_MES_TYPE_USER_12, 51
                                                STEP_MES_EXT_TYPE_COLOR_RGBAF, 52
   STEP_MES_TYPE_USER_13, 51
                                                STEP_MES_EXT_TYPE_COLOR_UNDEFINED,
   STEP_MES_TYPE_USER_14, 51
   STEP MES TYPE USER 15, 51
                                                    52
                                            ext light.h
   STEP_MES_TYPE_USER_2, 50
    STEP_MES_TYPE_USER_3, 50
                                                step_mes_ext_light, 52
                                                STEP_MES_EXT_TYPE_LIGHT_PHOTO_ILLUMINANCE,
    STEP MES TYPE USER 4, 51
   STEP MES_TYPE_USER_5, 51
                                                STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUM_FLUX,
   STEP_MES_TYPE_USER_6, 51
   STEP MES TYPE USER 7, 51
   STEP MES TYPE USER 8, 51
                                                STEP MES EXT TYPE LIGHT PHOTO LUM INTEN,
   STEP MES TYPE USER 9, 51
                                                    53
   STEP_MES_TYPE_VELOCITY, 50
                                                STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUMINANCE,
    STEP_MES_TYPE_VOLTAGE, 50
                                                    53
```

STEP_MES_EXT_TYPE_LIGHT_RADIO_IRRADIAN	IQ E .
53	step_mes_header, 41
STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANC	
53	matched_handler
STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANT	_FLU x ţep_node_callbacks, 44
53	Measurements, 26
STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANT_	_INTE \$l ep_mes_compression, 31
53	STEP_MES_COMPRESSION_LZ4, 31
STEP_MES_EXT_TYPE_LIGHT_UNDEFINED, 53	STEP_MES_COMPRESSION_NONE, 31
ext_temperature.h	step_mes_encoding, 31
step_mes_ext_temperature, 54	STEP_MES_ENCODING_BASE45, 32
STEP_MES_EXT_TYPE_TEMP_AMBIENT, 54	STEP_MES_ENCODING_BASE64, 32
STEP_MES_EXT_TYPE_TEMP_DIE, 54	STEP_MES_ENCODING_NONE, 32
STEP_MES_EXT_TYPE_TEMP_OBJECT, 54	step_mes_format, 32
STEP_MES_EXT_TYPE_TEMP_UNDEFINED, 54	STEP_MES_FORMAT_CBOR, 32
ext_type	STEP_MES_FORMAT_NONE, 32
step_mes_header, 41	step_mes_fragment, 32
	STEP_MES_FRAGMENT_FINAL, 32
Filter Definitions, 10	STEP_MES_FRAGMENT_NONE, 32
step_filt_evaluate, 11	STEP_MES_FRAGMENT_PARTIAL, 32
step_filt_print, 12	step_mes_print, 33
step_filter_op, 10	step_mes_sz_payload, 33
STEP_FILTER_OP_AND, 11	step_mes_timestamp, 32
STEP_FILTER_OP_AND_NOT, 11	STEP_MES_TIMESTAMP_EPOCH_32, 33
STEP_FILTER_OP_IS, 11	STEP_MES_TIMESTAMP_EPOCH_64, 33
STEP_FILTER_OP_NOT, 11	STEP_MES_TIMESTAMP_NONE, 33
STEP_FILTER_OP_OR, 11	STEP_MES_TIMESTAMP_UPTIME_MS_32, 33
STEP_FILTER_OP_OR_NOT, 11	STEP_MES_TIMESTAMP_UPTIME_MS_64, 33
STEP_FILTER_OP_XOR, 11	STEP_MES_TIMESTAMP_UPTIME_US_64, 33
Filter Match Cache, 7	step_mes_validate, 34
step_cache_add, 8	step_mes_vector_sz, 33
step_cache_check, 8	STEP_MES_VECTOR_SZ_2, 33
filter_bits	STEP_MES_VECTOR_SZ_3, 33
step_mes_header, 41	STEP_MES_VECTOR_SZ_4, 33
flags_bits	STEP_MES_VECTOR_SZ_NONE, 33
step_mes_header, 41	Nodes, 13
fragment step mes header, 41	step_node_callback_t, 13
Step_mes_neader, 41	step_node_error_t, 14
header	step_node_evaluate_t, 14
step measurement, 38	step_node_init_t, 14
otop_mododromont, oo	step_node_print, 16
ignore_mask	step_node_print, ro
step_filter, 36	payload
include/step/cache.h, 47	step_measurement, 38
include/step/filter.h, 47	Processor Node Management, 17
include/step/instrumentation.h, 48	step_pm_clear, 18
include/step/measurement/base.h, 49	step_pm_disable_node, 18
include/step/measurement/ext_color.h, 51	step_pm_enable_node, 18
include/step/measurement/ext_light.h, 52	step_pm_list, 19
include/step/measurement/ext_temperature.h, 53	step_pm_node_get, 19
include/step/measurement/measurement.h, 54	step_pm_poll, 19
include/step/measurement/unit.h, 56	step_pm_process, 20
include/step/node.h, 62	step_pm_register, 20
include/step/proc_mgr.h, 63	step_pm_resume, 21
include/step/sample_pool.h, 63	step_pm_suspend, 21
include/step/step.h, 64	,
init_handler	reserved
step_node_callbacks, 44	step_measurement, 38

Sample Pool Management, 22	Measurements, 32
step_sp_alloc, 22	STEP_MES_ENCODING_BASE64
step_sp_bytes_alloc, 23	Measurements, 32
step_sp_free, 23	STEP_MES_ENCODING_NONE
step_sp_get, 23	Measurements, 32
step_sp_put, 24	step_mes_ext_color
samples	ext_color.h, 51
step_mes_header, 41	step_mes_ext_light
Secure Telemetry Pipeline (STeP) API, 25	ext_light.h, 52
sourceid	step_mes_ext_temperature
step_mes_header, 42	ext_temperature.h, 54
srclen_bits	STEP_MES_EXT_TYPE_COLOR_CIE1931_XYY
step_mes_header, 42	ext_color.h, 52
start_handler	STEP_MES_EXT_TYPE_COLOR_CIE1931_XYZ
step_node_callbacks, 44	ext_color.h, 52
step_cache_add	STEP_MES_EXT_TYPE_COLOR_CIE1960_CCT
Filter Match Cache, 8	ext_color.h, 52
step_cache_check	STEP_MES_EXT_TYPE_COLOR_CIE1960_CCT_DUV
Filter Match Cache, 8	ext_color.h, 52
step_cache_rec, 35	STEP_MES_EXT_TYPE_COLOR_CIE1960_UCS
step_filt_evaluate	ext_color.h, 52
Filter Definitions, 11	STEP_MES_EXT_TYPE_COLOR_CIE1976_UCS
step_filt_print	ext_color.h, 52
Filter Definitions, 12	STEP_MES_EXT_TYPE_COLOR_RGBA16
step_filter, 35	ext_color.h, 52
ignore_mask, 36	STEP_MES_EXT_TYPE_COLOR_RGBA8
step_filter_chain, 36	ext color.h, 52
chain, 37	STEP_MES_EXT_TYPE_COLOR_RGBAF
count, 37	ext_color.h, 52
step_filter_op	STEP_MES_EXT_TYPE_COLOR_UNDEFINED
Filter Definitions, 10	ext_color.h, 52
STEP_FILTER_OP_AND	STEP_MES_EXT_TYPE_LIGHT_PHOTO_ILLUMINANCE
Filter Definitions, 11	ext_light.h, 53
STEP_FILTER_OP_AND_NOT	STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUM_FLUX
Filter Definitions, 11	ext_light.h, 53
STEP_FILTER_OP_IS	STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUM_INTEN
Filter Definitions, 11	ext_light.h, 53
STEP_FILTER_OP_NOT	STEP_MES_EXT_TYPE_LIGHT_PHOTO_LUMINANCE
Filter Definitions, 11	ext_light.h, 53
STEP_FILTER_OP_OR	STEP_MES_EXT_TYPE_LIGHT_RADIO_IRRADIANCE
Filter Definitions, 11	ext_light.h, 53
STEP_FILTER_OP_OR_NOT	STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANCE
Filter Definitions, 11	ext_light.h, 53
STEP_FILTER_OP_XOR	STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANT_FLUX
Filter Definitions, 11	ext_light.h, 53
step_measurement, 37	STEP_MES_EXT_TYPE_LIGHT_RADIO_RADIANT_INTEN
header, 38	ext_light.h, 53
payload, 38	STEP_MES_EXT_TYPE_LIGHT_UNDEFINED
reserved, 38	ext_light.h, 53
step_mes_compression	STEP_MES_EXT_TYPE_TEMP_AMBIENT
Measurements, 31	ext_temperature.h, 54
STEP_MES_COMPRESSION_LZ4	STEP_MES_EXT_TYPE_TEMP_DIE
Measurements, 31	ext_temperature.h, 54
STEP_MES_COMPRESSION_NONE	STEP_MES_EXT_TYPE_TEMP_OBJECT
Measurements, 31	ext_temperature.h, 54
step_mes_encoding	STEP_MES_EXT_TYPE_TEMP_UNDEFINED
Measurements, 31	ext_temperature.h, 54
STEP_MES_ENCODING_BASE45	step_mes_format
-	

Measurements, 32	unit.h, 59
STEP_MES_FORMAT_CBOR	STEP_MES_SI_SCALE_NANO
Measurements, 32	unit.h, 59
STEP_MES_FORMAT_NONE	STEP_MES_SI_SCALE_NONE
Measurements, 32	unit.h, 59
step_mes_fragment	STEP_MES_SI_SCALE_PETA
Measurements, 32	unit.h, 59
STEP_MES_FRAGMENT_FINAL	STEP_MES_SI_SCALE_PICO
Measurements, 32	unit.h, 59
STEP_MES_FRAGMENT_NONE	STEP_MES_SI_SCALE_TERA
Measurements, 32	unit.h, 59
STEP_MES_FRAGMENT_PARTIAL	STEP_MES_SI_SCALE_YOCTO
Measurements, 32	unit.h, 59
step_mes_header, 39	STEP_MES_SI_SCALE_YOTTA
_rsvd, 40	unit.h, 59
base_type, 40	STEP_MES_SI_SCALE_ZEPTO
compression, 40	unit.h, 59
ctype, 40	STEP MES SI SCALE ZETTA
data_format, 40	unit.h, 59
encoding, 41	step_mes_sz_payload
ext type, 41	Measurements, 33
-2.	
filter_bits, 41	step_mes_timestamp
flags_bits, 41	Measurements, 32
fragment, 41	STEP_MES_TIMESTAMP_EPOCH_32
len, 41	Measurements, 33
samples, 41	STEP_MES_TIMESTAMP_EPOCH_64
sourceid, 42	Measurements, 33
srclen_bits, 42	STEP_MES_TIMESTAMP_NONE
timestamp, 42	Measurements, 33
unit_bits, 42	STEP_MES_TIMESTAMP_UPTIME_MS_32
vec_sz, 42	Measurements, 33
step_mes_print	STEP_MES_TIMESTAMP_UPTIME_MS_64
Measurements, 33	Measurements, 33
step_mes_si_scale	STEP_MES_TIMESTAMP_UPTIME_US_64
unit.h, 58	Measurements, 33
STEP_MES_SI_SCALE_ATTO	step_mes_type
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_CENTI	STEP_MES_TYPE_ACCELERATION
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_DECA	STEP_MES_TYPE_ACIDITY
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_DECI	STEP MES TYPE AMPLITUDE
 unit.h, 59	base.h, 50
STEP MES SI SCALE EXA	STEP MES TYPE AREA
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_FEMTO	STEP MES TYPE CAPACITANCE
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_GIGA	STEP MES TYPE COLOR
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_HECTO	STEP_MES_TYPE_CONDUCTIVITY
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_KILO	STEP_MES_TYPE_COORDINATES
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_MEGA	STEP_MES_TYPE_CURRENT
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_MICRO	STEP_MES_TYPE_DIMENSION
unit.h, 59	base.h, 50
STEP_MES_SI_SCALE_MILLI	STEP_MES_TYPE_ENERGY

base.h, 50	base.h, 51
STEP_MES_TYPE_FORCE	STEP_MES_TYPE_USER_7
base.h, 50	base.h, 51
STEP_MES_TYPE_FREQUENCY	STEP_MES_TYPE_USER_8
base.h, 50	base.h, 51
STEP_MES_TYPE_HUMIDITY	STEP_MES_TYPE_USER_9
base.h, 50	base.h, 51
STEP_MES_TYPE_INDUCTANCE	STEP_MES_TYPE_VELOCITY
base.h, 50	base.h, 50
STEP_MES_TYPE_LAST	STEP_MES_TYPE_VOLTAGE
base.h, 51	base.h, 50
STEP_MES_TYPE_LIGHT	STEP_MES_TYPE_VOLUME
base.h, 50	base.h, 50
STEP_MES_TYPE_MAGNETIC_FIELD	step_mes_unit_ctype
base.h, 50	unit.h, 59
STEP_MES_TYPE_MASS	STEP_MES_UNIT_CTYPE_RANG_PERCENT_32
base.h, 50	unit.h, 60
STEP_MES_TYPE_MOMENTUM	STEP_MES_UNIT_CTYPE_RANG_PERCENT_64
base.h, 50	unit.h, 60
STEP_MES_TYPE_ORIENTATION	STEP_MES_UNIT_CTYPE_RANG_UNIT_INTERVAL_32
base.h, 50	unit.h, 60
STEP_MES_TYPE_PHASE	STEP_MES_UNIT_CTYPE_RANG_UNIT_INTERVAL_64
base.h, 50	unit.h, 60
STEP_MES_TYPE_PRESSURE	step_mes_unit_si
base.h, 50	unit.h, 60
STEP_MES_TYPE_RESISTANCE	STEP_MES_UNIT_SI_AMPERE
base.h, 50	unit.h, 60
STEP_MES_TYPE_SOUND	STEP_MES_UNIT_SI_BECQUEREL
base.h, 50	unit.h, 61
STEP_MES_TYPE_TEMPERATURE	STEP_MES_UNIT_SI_CANDELA
base.h, 50	unit.h, 60
STEP_MES_TYPE_TIME	STEP_MES_UNIT_SI_CANDELA_PER_METER_2
base.h, 50	unit.h, 61
STEP_MES_TYPE_UNDEFINED	STEP_MES_UNIT_SI_COULOMB
base.h, 50	unit.h, 61
STEP_MES_TYPE_USER_1	STEP_MES_UNIT_SI_DEGREE_CELSIUS
base.h, 50	unit.h, 61
STEP_MES_TYPE_USER_10	STEP_MES_UNIT_SI_FARAD
base.h, 51	unit.h, 61
STEP_MES_TYPE_USER_11	STEP_MES_UNIT_SI_GRAMS
base.h, 51	unit.h, 62
STEP_MES_TYPE_USER_12	STEP_MES_UNIT_SI_GRAY
base.h, 51	unit.h, 61
STEP_MES_TYPE_USER_13	STEP_MES_UNIT_SI_HECTOPASCAL
base.h, 51	unit.h, 62
STEP_MES_TYPE_USER_14	STEP_MES_UNIT_SI_HENRY
base.h, 51	unit.h, 61
STEP_MES_TYPE_USER_15	STEP_MES_UNIT_SI_HERTZ
base.h, 51	unit.h, <mark>61</mark>
STEP_MES_TYPE_USER_2	STEP_MES_UNIT_SI_INTERVAL
base.h, 50	unit.h, <mark>61</mark>
STEP_MES_TYPE_USER_3	STEP_MES_UNIT_SI_JOULE
base.h, 50	unit.h, 61
STEP_MES_TYPE_USER_4	STEP_MES_UNIT_SI_JOULE_PER_METER_2
base.h, 51	unit.h, 61
STEP_MES_TYPE_USER_5	STEP_MES_UNIT_SI_JOULE_PER_METER_2_PER_HZ
base.h, 51	unit.h, 61
STEP_MES_TYPE_USER_6	STEP_MES_UNIT_SI_JOULE_PER_METER_2_PER_NM

unit.h, 61	unit.h, 62
STEP_MES_UNIT_SI_JOULE_PER_METER_3	STEP_MES_UNIT_SI_SIEVERT
unit.h, 61	unit.h, 61
STEP_MES_UNIT_SI_KATAL unit.h, 61	STEP_MES_UNIT_SI_STERADIAN unit.h, 61
STEP_MES_UNIT_SI_KELVIN	STEP_MES_UNIT_SI_TESLA
unit.h, 60	unit.h, 61
STEP_MES_UNIT_SI_KILOGRAM	STEP_MES_UNIT_SI_VOLT
unit.h, 60	unit.h, 61
STEP_MES_UNIT_SI_LUMEN	STEP_MES_UNIT_SI_WATT
unit.h, 61	unit.h, 61
STEP_MES_UNIT_SI_LUMEN_PER_METER_2	STEP_MES_UNIT_SI_WATTS_PER_HERTZ
unit.h, 61	unit.h, 61
STEP_MES_UNIT_SI_LUMEN_PER_WATT	STEP_MES_UNIT_SI_WATTS_PER_METER_2
unit.h, 61	unit.h, 61
STEP_MES_UNIT_SI_LUMEN_SECOND	STEP_MES_UNIT_SI_WATTS_PER_METER_2_PER_HZ
unit.h, 61	unit.h, 61
	STEP_MES_UNIT_SI_WATTS_PER_METER_2_PER_NM
unit.h, 61 STEP_MES_UNIT_SI_LUX	unit.h, 61 STEP_MES_UNIT_SI_WATTS_PER_NM
unit.h, 61	unit.h, 61
STEP_MES_UNIT_SI_LUX_SECOND	STEP_MES_UNIT_SI_WATTS_PER_STERADIAN
unit.h, 61	unit.h, 62
STEP_MES_UNIT_SI_METER	STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_HERTZ
unit.h, 60	unit.h, 62
STEP_MES_UNIT_SI_METER_PER_SECOND_2	STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_METER_2
unit.h, 61	unit.h, 62
STEP_MES_UNIT_SI_METERS_2	STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_METER_2_PER
unit.h, 61	unit.h, 62
STEP_MES_UNIT_SI_METERS_3	STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_METER_2_PEF
unit.h, 62	unit.h, 62
STEP_MES_UNIT_SI_METERS_3_SECOND unit.h, 62	STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_NM unit.h, 62
STEP_MES_UNIT_SI_MICROTESLA	STEP_MES_UNIT_SI_WEBER
unit.h, 62	unit.h, 61
STEP_MES_UNIT_SI_MILLIVOLTS	step_mes_validate
unit.h, 62	Measurements, 34
STEP_MES_UNIT_SI_MOLE	step_mes_vector_sz
unit.h, 60	Measurements, 33
STEP_MES_UNIT_SI_NEWTON	STEP_MES_VECTOR_SZ_2
unit.h, 61	Measurements, 33
STEP_MES_UNIT_SI_OHM	STEP_MES_VECTOR_SZ_3
unit.h, 61	Measurements, 33
STEP_MES_UNIT_SI_PASCAL	STEP_MES_VECTOR_SZ_4
unit.h, 61 STEP_MES_UNIT_SI_PERCENT	Measurements, 33 STEP_MES_VECTOR_SZ_NONE
unit.h, 61	Measurements, 33
STEP_MES_UNIT_SI_PH	step_node, 43
unit.h, 62	step_node_callback_t
STEP_MES_UNIT_SI_RADIAN	Nodes, 13
unit.h, 61	step_node_callbacks, 43
STEP_MES_UNIT_SI_RELATIVE_HUMIDITY	evaluate_handler, 44
unit.h, 61	init_handler, 44
STEP_MES_UNIT_SI_SECOND	matched_handler, 44
unit.h, 60	start_handler, 44
STEP_MES_UNIT_SI_SIEMENS	step_node_error_t
unit.h, 61	Nodes, 14
STEP MES UNIT SI SIEMENS PER METER	STED NOCE EVALUATE T

	Nodes, 14	STEP_MES_SI_SCALE_TERA, 59
	node_init_t	STEP_MES_SI_SCALE_YOCTO, 59
	Nodes, 14	STEP_MES_SI_SCALE_YOTTA, 59
	node_print	STEP_MES_SI_SCALE_ZEPTO, 59
	Nodes, 16	STEP_MES_SI_SCALE_ZETTA, 59
step_	pm_clear	step_mes_unit_ctype, 59
F	Processor Node Management, 18	STEP_MES_UNIT_CTYPE_RANG_PERCENT_32,
step_	pm_disable_node	60
F	Processor Node Management, 18	STEP_MES_UNIT_CTYPE_RANG_PERCENT_64,
step_	pm_enable_node	60
F	Processor Node Management, 18	STEP_MES_UNIT_CTYPE_RANG_UNIT_INTERVAL_32,
step_	pm_list	60
F	Processor Node Management, 19	STEP_MES_UNIT_CTYPE_RANG_UNIT_INTERVAL_64,
step_	pm_node_get	60
F	Processor Node Management, 19	step_mes_unit_si, 60
step_	pm_node_record, 45	STEP_MES_UNIT_SI_AMPERE, 60
step_	pm_poll	STEP_MES_UNIT_SI_BECQUEREL, 61
F	Processor Node Management, 19	STEP MES UNIT SI CANDELA, 60
step_	pm_process	STEP_MES_UNIT_SI_CANDELA_PER_METER_2,
F	Processor Node Management, 20	61
step_	pm_register	STEP_MES_UNIT_SI_COULOMB, 61
F	Processor Node Management, 20	STEP_MES_UNIT_SI_DEGREE_CELSIUS, 61
step_	pm_resume	STEP MES UNIT SI FARAD, 61
F	Processor Node Management, 21	STEP_MES_UNIT_SI_GRAMS, 62
step_	pm_suspend	STEP_MES_UNIT_SI_GRAY, 61
F	Processor Node Management, 21	STEP_MES_UNIT_SI_HECTOPASCAL, 62
step_	sp_alloc	STEP MES UNIT SI HENRY, 61
5	Sample Pool Management, 22	STEP_MES_UNIT_SI_HERTZ, 61
step_	sp_bytes_alloc	STEP_MES_UNIT_SI_INTERVAL, 61
5	Sample Pool Management, 23	STEP_MES_UNIT_SI_JOULE, 61
step_	sp_free	STEP_MES_UNIT_SI_JOULE_PER_METER_2,
	Sample Pool Management, 23	61
step_	sp_get	STEP MES UNIT SI JOULE PER METER 2 PER HZ,
	Sample Pool Management, 23	61
step_	sp_put	STEP MES UNIT SI JOULE PER METER 2 PER NM,
	Sample Pool Management, 24	61
step_	sp_stats, 46	
. –		STEP_MES_UNIT_SI_JOULE_PER_METER_3,
timest	tamp	61
5	step_mes_header, 42	STEP_MES_UNIT_SI_KATAL, 61
		STEP_MES_UNIT_SI_KELVIN, 60
unit.h		STEP_MES_UNIT_SI_KILOGRAM, 60
	step_mes_si_scale, 58	STEP_MES_UNIT_SI_LUMEN, 61
	STEP_MES_SI_SCALE_ATTO, 59	STEP_MES_UNIT_SI_LUMEN_PER_METER_2,
	STEP_MES_SI_SCALE_CENTI, 59	61
	STEP_MES_SI_SCALE_DECA, 59	STEP_MES_UNIT_SI_LUMEN_PER_WATT, 61
	STEP_MES_SI_SCALE_DECI, 59	STEP_MES_UNIT_SI_LUMEN_SECOND, 61
	STEP_MES_SI_SCALE_EXA, 59	STEP_MES_UNIT_SI_LUMEN_SECOND_PER_METER_3,
	STEP_MES_SI_SCALE_FEMTO, 59	61
	STEP_MES_SI_SCALE_GIGA, 59	STEP_MES_UNIT_SI_LUX, 61
	STEP_MES_SI_SCALE_HECTO, 59	STEP_MES_UNIT_SI_LUX_SECOND, 61
	STEP_MES_SI_SCALE_KILO, 59	STEP_MES_UNIT_SI_METER, 60
	STEP_MES_SI_SCALE_MEGA, 59	STEP_MES_UNIT_SI_METER_PER_SECOND_2,
	STEP_MES_SI_SCALE_MICRO, 59	61
	STEP_MES_SI_SCALE_MILLI, 59	STEP_MES_UNIT_SI_METERS_2, 61
	STEP_MES_SI_SCALE_NANO, 59	STEP_MES_UNIT_SI_METERS_3, 62
	STEP_MES_SI_SCALE_NONE, 59	STEP_MES_UNIT_SI_METERS_3_SECOND, 62
5	STEP_MES_SI_SCALE_PETA, 59	STEP_MES_UNIT_SI_MICROTESLA, 62
5	STEP_MES_SI_SCALE_PICO, 59	STEP_MES_UNIT_SI_MILLIVOLTS, 62

```
STEP_MES_UNIT_SI_MOLE, 60
   STEP MES UNIT SI NEWTON, 61
   STEP_MES_UNIT_SI_OHM, 61
   STEP_MES_UNIT_SI_PASCAL, 61
   STEP_MES_UNIT_SI_PERCENT, 61
   STEP MES UNIT SI PH, 62
   STEP MES UNIT SI RADIAN, 61
   STEP MES UNIT SI RELATIVE HUMIDITY, 61
   STEP MES UNIT SI SECOND, 60
   STEP_MES_UNIT_SI_SIEMENS, 61
   STEP_MES_UNIT_SI_SIEMENS_PER_METER,
       62
   STEP_MES_UNIT_SI_SIEVERT, 61
   STEP MES UNIT SI STERADIAN, 61
   STEP_MES_UNIT_SI_TESLA, 61
   STEP_MES_UNIT_SI_VOLT, 61
   STEP MES UNIT SI WATT, 61
   STEP_MES_UNIT_SI_WATTS_PER_HERTZ, 61
   STEP_MES_UNIT_SI_WATTS_PER_METER_2,
    STEP_MES_UNIT_SI_WATTS_PER_METER_2_PER_HZ,
   STEP_MES_UNIT_SI_WATTS_PER_METER_2_PER_NM,
    STEP MES UNIT SI WATTS PER NM, 61
   STEP_MES_UNIT_SI_WATTS_PER_STERADIAN,
       62
   STEP MES UNIT SI WATTS PER STERADIAN PER HERTZ,
   STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_METER_2,
   STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_METER_2_PER_HZ,
   STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_METER_2_PER_NM,
    STEP_MES_UNIT_SI_WATTS_PER_STERADIAN_PER_NM,
   STEP_MES_UNIT_SI_WEBER, 61
unit bits
    step mes header, 42
vec sz
    step mes header, 42
```