Ubisense DIMENSION4™

DIMENSION4 trace messages

Wednesday, May 10, 2023

Ubisense Limited, St Andrew's House, St Andrew's Road, Cambridge CB4 1DL, United Kingdom.

Telephone: +44 (0)1223 535170. Website: *https://www.ubisense.com*

Ubisense Limited
St Andrew's House
St Andrew's Road
Cambridge CB4 1DL
United Kingdom

Tel: +44 (0)1223 535170

WWW: *https://www.ubisense.com*

Contents

DIMENSION4 trace messages

This section lists the trace messages that might be logged by a DIMENSION4 system, organized as follows:

- *The 'sensor_health' trace message*. An explanation of the health data periodically sent by sensors.
- *Location system information messages enabled by default*. Details of the 'boot', 'ls_sink_stats', 'ls_sink_info', 'ls_sink_time', 'logging_server_stats' and 'tftp_report' messages which are enabled by default in this version of DIMENSION4.
- *Location system information messages disabled by default*. Details of message streams that might be useful on occasion but are disabled by default.
- *Sensor information messages enabled by default*. Details of the 'sensor_init', 'sensor_info', and 'sensor_warning' messages which are enabled by default in this version of DIMENSION4.
- *Sensor fatal error messages*. A list of all fatal errors that might be sent by sensors.
- *Location system low-level debugging messages*. A list of low-level debugging messages sent by the location system, but disabled by default.
- *Sensor low-level debugging messages*. A list of low-level debugging messages sent by sensors, but disabled by default.
- *Location platform warning messages*. A list of some of the important warning messages that are logged by the underlying location platform support.

Message format

The format of every message is shown, using the following conventions:

- Constant strings printed in the message are shown as `fixed font`
- Variables, filled in with some value, are shown as *`italic`* (in the health section, they are also underlined).

For example, the format of one of the sensor_init messages is shown as:

```
sensor_init: Components OK, will request config from address (our
protocol address is address )
```

And in the log itself this might correspond to a displayed message like this:

```
sensor_init: Components OK, will request config from 10.1.2.4 (our
protocol address is 10.1.2.26)
```

Traces enabled by default

From DIMENSION4 version 1.0.2 commonly-required traces are enabled by default, so that no extra user configuration (of the platform_monitor parameter) is required. If these traces are explicitly enabled, no harm will be done, but the configuration step is no longer required.

The ‘sensor_health’ trace message

ARM section

The ARM section contains health information about the sensor control system, sensor-sensor and sensor-server protocols. It has several counters that show activity since the previous health report. These can be used to understand the size of data flows in the system (e.g. bandwidth to server, tag bandwidth at the sensor, sensor to sensor communications). It also includes more advanced performance data that may be useful if analyzing a subsequent sensor error, but is not valuable for normal field analysis.

This message is sent once a minute.

The health message is broken down as follows:

```
ARM
```

Health data from the sensor CPU (ARM)

```
|RECV s_arrays/scans/valid/meas/late
ScanArrays/Scans/ValidScans/Measurements/Late
```

The RECV section gives details about individual measurements the ARM has received over the network or from the digital signal processor (DSP).

ScanArrays: the DSP sends measurements to the ARM in bulk messages called ScanArrays. This value is the number of messages the ARM has received from the DSP.

Scans: the number of measurements the ARM has received from the DSP. For example, if there are four measurements per full ScanArray and the ARM has received six half-full ScanArrays, then the ARM has received twelve measurements.

Valid Scans: the number of measurements which had a valid scan. Approximately half the measurements will not have been scanned because the sensor has a high-resolution decoder which can perform accurate scans to get fine TDOA data and angle data, and a low-resolution decoder which does not. The measurements from the low-resolution decoder are currently not used.

Measurements: the number of measurements received from the network.

Late: the number of measurements (local or received via the network) that were too late to be included in processing for that beacon.

```
|SENT rem_meas/fltr_sets/fltr_meas
Measurements/FilterInputSets/FilterMeasurements
```

This SENT section gives details about what the ARM has sent over the network or to the DSP.

Measurements: the number of measurements the ARM has sent to other sensors over the network.

FilterInputSets: the ARM sends measurements to the DSP to be filtered in sets, with each set containing all the measurements gathered for a particular tag beacon event. This value is the number of sets sent to the DSP.

FilterMeasurements: the total number of measurements sent to the DSP to be filtered. For example, if every tag is seen by exactly five sensors, this value should be equal to FilterInputSets multiplied by five.

```
|FILTER lp_sets/res/seq/drpd_sets
SetsDroppedLowPower/FilterResults/FilterSequenceHints/DiscardedFilterSets
```

The FILTER section describes the data flows in/out of the filter (from the ARM's perspective). This is for tags where this sensor is acting as the 'master'.

SetsDroppedLowPower: the number of sets of messages which were dropped due to power thresholding.

FilterResults: the number of filter results received by the ARM.

FilterSequenceHints: the number of messages which tells the ARM which filter sets have been filtered.

DiscardedFilterSets: the number of filter sets which have been discarded (for example, if the filter queue is too big).

```
|SENT loc_v2/loc_v3 SentLocationMessagesV2/SentLocationMessagesV3
```

This SENT section shows counts of data messages sent to the server.

SentLocationMessagesV2: the number of location messages sent to the location cell manager.

SentLocationMessagesV3: the number of location messages (these include the measurement data) sent to the logging server.

```
|FAILED meas/loc_v2/loc_v3
SendMeasurementFailures/SendV2Failures/SendV3Failures
```

The FAILED section shows counts of low-level network send failures.

SendMeasurementFailures: the number of measurements sent to other sensors which failed to send.

SendV2Failures: the number of location messages sent to the location cell manager which failed to send.

SendV3Failures: the number of location messages sent to the logging server which failed to send.

```
|STATE alloc_meas/fq_sets/fq_meas
AllocatedMeasurements/FilterQueueSets/FilterQueueMeasurements
```

The STATE section shows the current state of various buffers.

AllocatedMeasurements: the number of measurements currently allocated.

FilterQueueSets: the number of filter sets which have been sent to the DSP to be filtered for which the ARM is still waiting for a filter result.

FilterQueueMeasurements: the number of measurements in the queued filter sets.

```
|HWM AllocatedMeasurements/FilterQueueSets/FilterQueueMeasurements
```

The HWM section lists high-water-marks (i.e. the highest value seen since the last message) for the values in the STATE message (see above).

```
|MSGQALLOC: for each channel i with alloc errors: #i: AllocErrors[i]
(if there are no errors) (no errors)
```

The MSGQALLOC section gives the total number of errors encountered whilst allocating messages to communicate with the DSP. For each message type, this section gives the number of allocation errors (if a message type has no errors then it is omitted). If there have been no errors then this section has the text "(no errors)".

```
MEM HWM/RSS Memory HWM/MemoryResidentSetSize
or if not able to retrieve memory monitor statistics: (get use err)
```

The MEM section gives information on the ARM's memory usage. If unable to retrieve the usage statistics this section prints an error message.

HWM: the high-water-mark for the memory used by the ARM application.

MemoryResidentSetSize: the current memory used by the ARM application.

```
| LOOP free/sent/loop_errs/lb_errs
SlotsFree/SlotsSent/LoopErrors/LoopBufferErrors
```

The LOOP section shows the current state of the message loop between the ARM and DSP. The message loop is used to send some messages to/from the DSP. If there are no slots free, then this will cause problems.

SlotsFree: the current number of slots in the message loop which are free (to be allocated).

SlotsSent: the current number of slots in the message loop which have been sent to the DSP.

LoopErrors: the number of errors in the message loop.

LoopBufferErrors: the number of errors in the message loop buffer.

DSP section

The DSP section contains information about the DSP and the underlying hardware. In general the data is for advanced analysis only. However the 'HEALTH load:' percentage will give an idea of the remaining processing capacity, and the TS section will give an idea of the performance of the timing subsystem.

This message is sent once a minute.

```
| DSP
```

Health data from the DSP.

```
|TASKS for each task: TaskNameActivityCount
```

The TASKS section shows the number of times each task has run. This can be useful to debug unusual situations.

```
| UPP for each stage in the data pipeline: StageCount
```

The UPP section shows the message counts for each stage in the data pipeline from the receiver through to sending the measurement to the ARM.

```
| FILTER for each stage in the filter pipeline: StageCount
```

The FILTER section shows the message counts for each stage in the filter pipeline.

```
| MSGALLOC: for each channel i with alloc errors: #i: AllocErrors[i]
(if there are no errors) (no errors)
```

The MSGALLOC section gives the total number of errors encountered whilst allocating messages to communicate with the DSP. For each message type, this section gives the number of allocation errors (if a message type has no errors then it is omitted). If there have been no errors then this section has the text "(no errors)".

```
| HEALTH load: DSPProcessorLoad %
```

The HEALTH section shows the current load on the DSP in percent.

```
LOOP free/sent MessageQueueLoopSlotsFree/MessageQueueLoopSlotsSent
```

The LOOP section shows the current state of the message loop between the DSP and ARM. The message loop is used to send some messages to/from the ARM. If there are no slots free, then this will cause problems.

MessageQueueLoopSlotsFree: the current number of slots in the message loop which are free (to be allocated).

MessageQueueLoopSlotsSent: the current number of slots in the message loop which have been sent to the ARM.

```
MEM used/size/blkfree  MemoryUsed/MemorySize/LargestFreeBlock
```

The MEM section shows the current state of the DSP's memory.

```
| DRIFT ClockDriftMicroseconds
```

The DRIFT section shows the accumulated clock drift in microseconds.

```
| TS ok/sht/lng NormalTimeslotCount/ShortTimeslotCount/LongTimeslotCount
```

The TS section shows statistics about the timing subsystem. If ShortTimeslotCount or LongTimeslotCount are increasing, this may indicate a problem.

```
| CFAR min-max threshold/raw/stripped/real w1-w2/x1-x2/y1-y2/z1-z2
```

The CFAR section shows the minimum and maximum CFAR values since the last DSP health message for four ranges of values: CFAR threshold *w1-w2*, raw bits *x1-x2*, stripped bits *y1-y2*, and real bits *z1-z2*.

```
| RADIO installed
| RADIO absent
```

If the trace contains the `| RADIO installed` section then the sensor has a working radio; if the trace has the `| RADIO absent` section then no radio is installed.

Additional Periodic DSP health information

An additional message is sent out once every three minutes with DSP health information:

```
|TEMP nrDAC/nrRX TempNearDAC/TempNearReceiver
```

TempNearDAC: temperature near the DAC in Celsius

TempNearReceiver: temperature near the receiver in Celsius

```
|DSP|glbl_stack GlobalStackBytesUsed
```

GlobalStackBytesUsed: number of bytes used in the global stack

```
|<Task> mode/sp/used/handle
TaskMode/TaskStackPtr/TaskUsedStackBytes/TaskHandleAddr
```

For each task:

TaskMode: the current mode of the task

TaskStackPtr: the stack pointer address

TaskUsedStackBytes: the number of bytes used in the task's stack

TaskHandleAddr: the address of the task handle

```
|PPM cnt/no_f/no_l/insuf
PacketCount/NoPulsesFirstPart/NoPulsesLastPart/NotEnoughPulses
```

PacketCount: the count of packets considered

NoPulsesFirstPart: the count of packets with no pulse in the first part of the packet

NoPulsesLastPart: the count of packets with no pulse in the last part of the packet

NotEnoughPulses: the count of packets with not enough pulses

```
|CABLE unsync/ok/1/>1
UnsyncedBits/ErrorFreeBytes/SingleErrorBytes/MultipleErrorBytes
```

UnsyncedBits: (unused)

ErrorFreeBytes: bytes which were error free

SingleErrorBytes: bytes which had a single bit error

MultipleErrorBytes: bytes which had multiple bit errors

```
|LOOP_ERRORS buf/loop TotalLoopBufferErrors/TotalLoopErrors
```

TotalLoopBufferErrors: the number of errors in the message loop buffer.

TotalLoopErrors: the number of errors in the message loop.

Location system information messages enabled by default

Trace option 'boot'

This option traces operation of the boot server. The message formats are:

```
boot: Boot configuration server listening on address
```

```
boot: Boot configuration server setting intended versions for
kernel/filesystem/firmware to KernelVersion/FSVersion/FirmwareVersion
```

```
boot: Sensor mac requested configuration (v ProtocolVersion) at
address, returning INVALID (server in test mode)
```

```
boot: Sensor mac requested configuration (v ProtocolVersion) at
address, returning OK, kernel: KernelVersion/KernelLength/KernelCRC,
fs: FSVersion/FSLength/FSCRC, firmware:
FirmwareVersion/FirmwareLength/FirmwareCRC, nextaddr:
BootServerAddress
```

```
boot: Sensor mac requested configuration (v ProtocolVersion) at
address, returning OK/kernel_KernelVersion/fs_FSVersion/addr/kcrc_
KernelCRC/fcrc_FSCRC
```

```
boot: Sensor mac requested filename at address
```

```
boot: Sensor mac completed request for filename at address
```

Trace option ‘ls_sink_info’

This option traces location sink configuration. It can be used to check for whether the persistent or nonpersistent sensor status storage is enabled. The message formats are:

```
ls_sink_info: Location sink for cell cell not storing status
persistently.
```

```
ls_sink_info: Location sink for cell cell storing status
persistently.
```

Trace option ‘ls_sink_stats’

This option traces the messages received by the location sink server. It prints a single trace format that can contain counts for different protocol messages:

```
ls_sink_stats: cell_ received message_counts
```

Trace option ‘ls_sink_time’

This option traces the time synchronization protocol that is controlled by the location sink. It has a single message format:

```
ls_sink_time: TIME SYNC SERVER: client time = tsm.client_time_ (
tsm.client_time_.seconds() s tsm.client_time_.remainder_nanoseconds()
ns) server = tsm.server_time_ ( tsm.server_time_.seconds() s
tsm.server_time_.remainder_nanoseconds() ns) send to address
```

Trace option ‘logging_server_stats’

This option traces the messages received by the logging server (excluding trace messages). It prints a single trace format that can contain counts for different sensor messages. In the current version the only message type supported is ULocationSystem::Messages::LocationMessageV3-Sensor (i.e. a location message from a sensor).

```
logging_server_stats: cell received message_counts
```

Trace option ‘tftp_report’

This option gives a report of the TFTP server embedded within the boot server. It has a single message format:

```
boot_server: XFERS ok/failed: ok / failed CONNS lwm/curr/hwm: lwm /
curr / hwm RRQ drpd: drpd LOSS_INDICATORS rrq_dup/retries/ack_
old/ack_fut: rrq_dup / retries / ack_old / ack_fut
```

XFERS ok/failed: running total of counts of transfers which successfully completed or failed.

CONNS lwm/curr/hwm: the low-water-mark, current and high-water-mark for the number of active connections. By default the boot server has a maximum of 32 concurrent connections. The lwm and hwm counts are measured since the previous tftp_report message.

RRQ drpd: the number of file transfer requests dropped due to reaching the limit on concurrent connections.

LOSS_INDICATORS rrq_dup/retries/ack_old/ack_fut: various statistics that might indicate lost packets: rrq_dup = duplicate file requests, retries = number of retried server sends, ack_old = number of stale acknowledgement packets, ack_fut = number of extremely old acknowledgement packets.

Location system information messages disabled by default

Trace option ‘ls_upstream_info’

This option traces the timing cable topology discovery process. The message formats are:

```
ls_upstream_info: cell_ got upstream_info.upstream_mac_ port
upstream_info.upstream_port_ -> upstream_info.mac_ (found downstream
sensor ? . : (did not find downstream sensor))
```

```
ls_upstream_info: cell_ got upstream_info.upstream_mac_ port
upstream_info.upstream_port_ -> upstream_info.mac_ (found upstream
sensor ? . : (did not find upstream sensor))
```

```
ls_upstream_info: cell_ got upstream_info.upstream_mac_ port
upstream_info.upstream_port_ -> upstream_info.mac_(update needed ? .
: (skipping server update))
```

```
ls_upstream_info: parameter_name changed for downstream_mac to trace_
value
```

```
ls_upstream_info: Reported parameter_name changed for downstream_mac
to trace_value
```

Trace option 'ls_sink_tag_data'

This option traces the receipt of telemetry data from tags. It has a single message format:

```
ls_sink_tag_data: Location sink for cell cell_ received tag data for
msg.tag_id_ at msg.time_ with values TagDataPrinter(msg.values_)
```

Trace option 'tftp_info'

This option traces the transfer of boot files using TFTP. It emits about five messages per transfer, and has the following formats:

```
tftp_info: boot_server: creating connection for client_address
requesting filename with blocksize blocksize.
```

```
tftp_info: boot_server: sending OACK for client_address requesting
filename with blocksize blocksize.
```

```
tftp_info: boot_server: connection for client_address (requested
filename) transitioning to state WAITING_FOR_OACK_ACK.
```

```
tftp_info: boot_server: connection for client_address (requested
filename) transitioning to state WAITING_FOR_DATA_ACK.
```

```
tftp_info: boot_server: destroying connection for client_address
(requested filename) as sending OACK exceeded maximum retries.
```

```
tftp_info: boot_server: destroying connection for client_address
(requested filename) as sending DATA exceeded maximum retries.
```

```
tftp_info: boot_server: destroying connection for client_address
(requested filename) as error packet received.
```

```
tftp_info: boot_server: destroying connection for client_address
(requested filename) as transfer has completed.
```

```
tftp_info: boot_server: destroying old connection for client_address
(requested filename) as client has new request.
```

```
tftp_info: boot_server: destroying connection for client_address
(requested filename) as server unable to get file data.
```

Sensor information messages enabled by default

Trace option ‘sensor_init’

This option traces sensor initialization. The message formats are:

```
sensor_init: Components OK, will request config from address (our
protocol address is address )
```

```
sensor_init: Config Manager OK
```

```
sensor_init: Configuration received OK
```

```
sensor_init: Detected receiver type receiver_type
```

```
sensor_init: DSP config OK

sensor_init: DSP ready, waiting for time sync

sensor_init: DSP time sync OK

sensor_init: DSP-controlled hardware OK

sensor_init: GPP-side IPC OK

sensor_init: Initial IPC OK

sensor_init: Initialisation complete

sensor_init: IPC Loop OK

sensor_init: LEDs, Resetter OK

sensor_init: Opening sensor routing channel

sensor_init: Opening UWB channel

sensor_init: Sending config to DSP

sensor_init: Time synchronisation OK

sensor_init: Waiting for DSP hardware initialisation
```

```
sensor_init: Waiting for DSP to be ready
```

```
sensor_init: Watchdog OK
```

Trace option ‘sensor_info’

This option traces various sensor behaviors. Normal message formats are:

```
sensor_info: Clearing local CNC as it is invalid.
```

```
sensor_info: Requested firmware upgrade not necessary: version
flashed_firmware_version_ is already flashed
```

```
sensor_info: Requested software upgrade not necessary: version
flashed_kernel_version_ / flashed_fs_version_ is already flashed
```

```
sensor_info: Successfully re-assigned MAC from
settings_.instruction_.old_mac_ to settings_.instruction_.new_mac_ .
Will now reboot.
```

```
sensor_info: timing is stable
```

```
sensor_info: Updating CNC status from remote_cnc_state.status_ to
local_cnc_status .
```

```
sensor_info: Updating local CNC from local_cnc to requested remote_
cnc_state.cnc_ .
```

```
sensor_info: Upgrading firmware from flashed_firmware_version_ to
desired_version successful, will now reboot.
```

```
sensor_info: Upgrading firmware from flashed_firmware_version_ to
desired_version
```

```
sensor_info: Upgrading flashed software from flashed_kernel_version_
/ flashed_fs_version_ to desired_kernel / desired_fs successful, will
now reboot.
```

```
sensor_info: Upgrading flashed software from flashed_kernel_version_
/ flashed_fs_version_ to desired_kernel / desired_fs
```

Message formats that probably indicate problems are:

```
sensor_info: timing is UNSTABLE
```

```
sensor_info: DSP health timeout, resetting sensor
```

Trace option ‘sensor_warning’

This option traces is used to deliver non-serious warnings about time synchronization. If network delays are very high, then the network-based time synchronization protocol will lose accuracy. This is not a problem in this version of the software because a sensor doesn’t use timestamps from other sensors as part of its location protocol, but it is included in case we wish to use timestamps in this way in future versions. Normal message formats are:

```
sensor_warning: Time synchronisation: estimated network delay is
value .
```

Sensor fatal error messages

These messages are sent by a sensor in response to an unrecoverable error; the sensor will then reboot. Message formats are:

```
fatal: >1 AD9783 instance
```

fatal: >1 AMChannelReference instance

fatal: >1 CCxx10 instance

fatal: >1 ClockGeneration instance

fatal: >1 FPGA instance

fatal: >1 hamming error but decoded nibble not 255.

fatal: >2 ADF4002 instances

fatal: >2 ADT7302 instances

fatal: AD9783 BIST failed (data1 = *ad9783_result*)

fatal: AD9783 initialisation timeout (data1 = *reg*)

fatal: AD9783 SH calibration failed (data1 = *ad9783_result*)

fatal: AD9783 SH calibration outside normal range (data1 = *ad9783_result* , data2 = *settings_.sh_calibration_max_ x 1000 + settings_.sh_calibration_min_*)

fatal: Assert CNC status failed

fatal: Assert EEPROM info failed: query processor EEPROM failure

fatal: Assert EEPROM info failed: query receiver EEPROM failure

fatal: Assert EEPROM info failed: remote operation error

fatal: Boot file CRC error. (data1 = *calculated_crc* , data2 = *expected_crc*)

fatal: CC2510 bad chip ID

fatal: CC2510 version error

fatal: CCxx10 SPI xfer size too big (data1 = *size_in_16_bit_words*)

fatal: CCxx10: bad chip id (data1 = *chip_id_*)

fatal: CCxx10: unknown chip id (data1 = *chip_id_*)

fatal: CCxx10: verification error

fatal: Clock count assertion (data1 = *CLK_countspms()*)

fatal: Config component registration too late

fatal: Config establish timeout. Sensor will now reboot.

fatal: Config registration request failed (server at *config_server_address*) response: incompatible configuration request. Sensor will now reboot.

fatal: Config registration request failed (server at *config_server_address*) response: MAC *mac()* is not known. Sensor will now reboot.

fatal: Config registration request failed (server at *config_server_address*) unable to read response (error *invoke_result->get_error()*). Sensor will now reboot.

fatal: Config registration request failed (server at *config_server_address*) unknown response *op_result* . Sensor will now reboot.

fatal: Config registration request failed (server at *config_server_address*). Sensor will now reboot.

fatal: data pipeline select fail (data1 = *result*)

fatal: EDMA3 initialise failed

fatal: emergency

fatal: error *result* reading active firmware page

fatal: Error committing active firmware page (data1 = *eeprom_result*)

fatal: Error detecting active firmware page (data1 = *spi_result.first* , data2 = *spi_result.second*)

fatal: Error writing active firmware page (data1 = *eeprom_result*)

fatal: Failed firmware upgrade: downloaded kernel size mismatch, expected *available_length* but file is *firmware_buffer.written_size()* . Sensor will now reboot.

fatal: Failed software upgrade: downloaded filesystem size mismatch, expected *available_fs_size* but file is *fs_buffer.written_size()* . Sensor will not reboot.

fatal: Failed software upgrade: downloaded kernel size mismatch, expected *available_kernel_size* but file is *kernel_buffer.written_size()* . Sensor will now reboot.

fatal: Failed to allocate IPCCheck message

fatal: Failed to assert *name* CRC (result= *eeprom_result*)

fatal: Failed to assert *name* length (result= *eeprom_result*)

fatal: Failed to assert *name* version (result= *eeprom_result*)

fatal: Failed to assert firmware page (data1 = *eeprom_result*)

fatal: failed to attach DSP: (data1 = *status*)

fatal: Failed to commit flashed software details to EEPROM (data1 = *eeprom_result*)

fatal: Failed to commit phase 1 to EEPROM (data1 = *eeprom_result*)

fatal: Failed to commit phase 2 to EEPROM (data1 = *eeprom_result*)

fatal: Failed to download file *filename* from *boot_server_address* for *operation* upgrade. Sensor will now reboot.

fatal: failed to load DSP: (data1 = *status*)

fatal: Failed to retrieve boot configuration from *boot_config_address* for *operation* upgrade. Sensor will now reboot.

fatal: failed to setup DSP: (data1 = *status*)

fatal: failed to start DSP: (data1 = *status*)

fatal: Failed to switch active firmware page. (data1 = *spi_result.first* , data2 = *spi_result.second*)

fatal: Failed to unset firmware page (data1 = *eeprom_result*)

fatal: Failed to unset fs version (data1 = *eeprom_result*)

fatal: Failed to unset kernel version (data1 = *eeprom_result*)

fatal: Failed to verify written firmware CRC. (data1 = *spi_result.first* , data2 = *spi_result.second*)

fatal: Failed to verify written fs CRC. (data1 = *spi_result.first* , data2 = *spi_result.second*)

fatal: Failed to verify written kernel CRC. (data1 = *spi_result.first* , data2 = *spi_result.second*)

fatal: Failed to write firmware upgrade. (data1 = *spi_result.first* , data2 = *spi_result.second*)

fatal: Failed to write fs upgrade. (data1 = *spi_result.first* , data2 = *spi_result.second*)

fatal: Failed to write kernel upgrade. (data1 = *spi_result.first* , data2 = *spi_result.second*)

fatal: fatal log init 1

fatal: fatal log init 2

fatal: fatal log init 3

fatal: fatal log init 4

fatal: fatal log init 5

fatal: Filesystem CRC mismatch. (data1 = *spi_crc* , data2 = *available_fs_crc*)

fatal: Firmware CRC mismatch. (data1 = *spi_crc* , data2 = *available_crc*)

fatal: Firmware version mismatch: intended version is *desired_version* , but server at *boot_server_address* reports available version is *available_version* . Sensor will now reboot.

fatal: FPGA INIT_B ready timeout

fatal: FPGA initialise failed

fatal: FPGA MCB error (data1 = *(ones_cnt_rxbuf_p_->subsequent_reset_flag_ 9) | (ones_cnt_rxbuf_p_->mcb_calibration_done_ 8) | (ones_cnt_rxbuf_p_->p2_mcb_rd_error_ 7) | (ones_cnt_rxbuf_p_->p3_mcb_rd_error_ 6) | (ones_cnt_rxbuf_p_->p4_mcb_wr_error_ 5) | (ones_cnt_rxbuf_p_->p5_mcb_wr_error_ 4) | (ones_cnt_rxbuf_p_->p2_mcb_rd_overflow_ 3) | (ones_cnt_rxbuf_p_>p3_mcb_rd_overflow_ 2) | (ones_cnt_rxbuf_p_->p4_mcb_wr_underrun_ 1) | ones_cnt_rxbuf_p_->p5_mcb_wr_underrun_)*

fatal: FPGA ones counter test pattern error (data1 = *test_pattern*)

fatal: FPGA PLL/DCM unlocked (data1 = *0*)

fatal: FPGA PLLs and DCM failed to lock (data1 = *0*)

fatal: FPGA programming failed

fatal: FPGA subsequent reset error (data1 = *ones_cnt_rxbuf_p_->subsequent_reset_flag_*)

fatal: Illegal config listener address

fatal: InitStatus alloc failure (data1 = *state*)

fatal: invalid init check

fatal: IPC add handler check failed (data1 = *offset* , data2 = *msg_max_count_*)

fatal: IPC add handler duplication (data1 = *(free_message ? 1000 : 0) + offset* , data2 = *msg_max_count_*)

fatal: IPC code sanity check failed (data1 = *offset* , data2 = *msg_max_count_*)

fatal: IPC null handler (data1 = *offset* , data2 = *msg_max_count_*)

fatal: Kernel CRC mismatch. (data1 = *spi_crc* , data2 = *available_kernel_crc*)

fatal: MAC assignment completion notification failed

fatal: No Location System Configuration Server registered. Sensor will now reboot.

fatal: PRD assertion (data1 = *CLK_getprd()*)

fatal: Processor Task: task_loop exited (data1 = *result*)

fatal: Receiver PLL unlocked (data1 = *0*)

fatal: sample proc not configured

fatal: Sample Processor: coarse clock overflow (data1 = *protocol_data_packet.first_pulse_clock_count_* , data2 = *protocol_data_packet.first_pulse_ppe_number_*)

fatal: Sanity checks failed.

fatal: Software version mismatch: intended version is *desired_kernel / desired_fs* , but server at *boot_server_address* reports available version is *available_kernel_version / available_fs_version* . Sensor will now reboot.

fatal: SPI CRC failed (data1 = *0*)

fatal: SPI CRC failed (data1 = *1*)

fatal: SPI CRC failed (data1 = *2*)

fatal: SPI CRC failed (data1 = *3*)

fatal: SPI create channel failed

fatal: SPI xfer failed (data1 = *device* , data2 = *length*)

fatal: Timesync alloc error 1

fatal: Timesync alloc error 2

fatal: TimeSync info alloc failed

fatal: TimeSync init alloc failed

fatal: Timesync latency error

fatal: unable to cache network configuration from EEPROM

fatal: unable to cache Processor EEPROM values before re-assigning MAC.

fatal: unable to cache processor EEPROM values: *result*

fatal: unable to cache receiver EEPROM values: *result*

fatal: unable to commit cleared CNC to EEPROM. Error *result* .

fatal: unable to commit CNC changes to EEPROM. Error *result* .

fatal: Unable to comprehend boot configuration (data1 = *source.get_error()* , data2 = *buffer.written_size()*)

fatal: unable to detect flashed firmware version (data1 = *result* , data2 = *firmware_page0_active_ ? 0 : 1*)

fatal: unable to detect flashed fs version (data1 = *result*)

fatal: unable to detect flashed kernel version (data1 = *result*)

fatal: Unable to perform *operation* upgrade due to missing boot config server parameter in configuration. Sensor will now reboot.

fatal: unable to query board identifier from Processor EEPROM

fatal: unable to query board identifier from Receiver EEPROM

fatal: unable to re-assign MAC address, error code *result* .

fatal: unable to reset gracefully. (data1 = *1* , data2 = *result*)

fatal: unable to reset gracefully. (data1 = *2* , data2 = *result*)

fatal: unable to write CNC (DNS IP # *i*) to EEPROM. Error *result* .

fatal: unable to write CNC (DNS Suffix # *i*) to EEPROM. Error *result* .

fatal: unable to write CNC (Gateway IP) to EEPROM. Error *result* .

fatal: unable to write CNC (IP) to EEPROM. Error *result* .

fatal: unable to write CNC (Search method # *i*) to EEPROM. Error *result* .

fatal: unable to write CNC (Subnet mask) to EEPROM. Error *result* .

fatal: Unexpected FPGA programming result

fatal: Unknown AM channel threshold method (data1 = *settings_.method_*)

fatal: Unknown AM channel threshold state (data1 = *state_.machine_state_*)

fatal: unknown receiver type (data1 = *receiver_type*)

fatal: Unknown timing cable OTW decoder state (data1 = *state_*)

fatal: Unknown timing cable OTW encoder state (data1 = *state_*)

fatal: unrecoverable dsp fatal error

```
fatal: UPP Ifc Task exited (data1 = result )

fatal: UPP initialise failed

fatal: UPP: error during read (data1 = result )

fatal: UPP: invalid packet source (data1 = pusher->protocol_data_
packet_.source_ )

fatal: urgent handler emergency init 1

fatal: urgent handler emergency init 2

fatal: urgent handler not initialised.

fatal: Writer failed to locate ARM MSGQ reader

fatal: written results overflow (data1 = loop_state_.written_results_
)
```

Location system low-level debugging messages

There are many low-level tracing options available whose main purpose is debugging system software operation. They should not normally be enabled but are included here to give an idea of what could be traced in principle.

Trace option ‘boot_d’

This does extra tracing on the boot protocol.

```
boot_d: Adding boot file file version version
```

```
boot_d: Sensor mac registering at address.get_name_local
```

```
boot_d: Sensor mac: sending filename (bytes bytes ) to address
```

Trace option 'ls_sink_liveness_d'

This traces the 'server liveness timeout' behavior.

```
ls_sink_liveness_d: cell is not a location cell
```

```
ls_sink_liveness_d: increasing timeout to minimum allowed (min
timeout) for geometry cell (was timeout)
```

```
ls_sink_liveness_d: no logging address associated with cell location
cell
```

```
ls_sink_liveness_d: parent cell is not a geometry cell for location
cell
```

```
ls_sink_liveness_d: parent cell not found for location cell
```

```
ls_sink_liveness_d: removing expiry time of expiry for location cell
because timeout is now set to zero
```

```
ls_sink_liveness_d: setting expiry time of time for location cell
```

```
ls_sink_liveness_d: setting location sink of address for location
cell
```

```
ls_sink_liveness_d: setting logging server of address for location
cell
```

```
ls_sink_liveness_d: setting time server of address for location cell
```

Trace option ‘ls_timing_graph_d’

This traces the calculation of timing graphs.

```
ls_timing_graph_d: Finished timing graph for root
```

```
ls_timing_graph_d: Found route from root to v with delay delay,
variance variance
```

```
ls_timing_graph_d: Shortest distance from root to sensor is distance
```

```
ls_timing_graph_t: Calculated N delays
```

```
ls_timing_graph_t: Calculating delays with N timing root sensors
```

```
ls_timing_graph_t: Delay for sensor is delay (variance variance)
```

```
ls_timing_graph_t: Timing root sensor root
```

```
ls_timing_graph_t: Updated delay for sensor to delay (variance
variance)
```

Trace option ‘ls_timing_delay_checker’

This traces server-side solving of sensor orientations and cable delays.

```
ls_timing_delay_checker: check_cable_swaps
```

```
ls_timing_delay_checker: check_delays
```

```
ls_timing_delay_checker: check_descriptors

ls_timing_delay_checker: check_estimated_positions

ls_timing_delay_checker: check_orientation_result_sensors

ls_timing_delay_checker: check_orientation_results_invalid

ls_timing_delay_checker: check_orientation_results_invalidated

ls_timing_delay_checker: check_orientation_results

ls_timing_delay_checker: check_orientation_solved

ls_timing_delay_checker: check_orientations

ls_timing_delay_checker: check_overrides

ls_timing_delay_checker: check_sensor_moves

ls_timing_delay_checker: check_sensor_swaps

ls_timing_delay_checker: check_timing_result_routes

ls_timing_delay_checker: check_timing_result_sensors

ls_timing_delay_checker: check_timing_results_invalid
```

```
ls_timing_delay_checker: check_timing_results_invalidated

ls_timing_delay_checker: check_timing_results

ls_timing_delay_checker: check_timing_routes

ls_timing_delay_checker: check_timing_solved

ls_timing_delay_checker: check_valid_flags

ls_timing_delay_checker: Starting timing delay checker

ls_timing_delay_checker: sync_delays

ls_timing_delay_checker: sync_installation_properties

ls_timing_delay_checker: sync_positions

ls_timing_delay_checker: sync_valid_flags
```

Trace option ‘ls_referential_integrity’

This traces the object referential integrity checker.

```
ls_referential_integrity: Pruning all parameters that are not in a
set of N objects

ls_referential_integrity: Pruning all sensor-group pairs that are not
in a set of N objects
```

```
ls_referential_integrity: Pruning parameters from a set of N objects

ls_referential_integrity: Pruning sensor-group pairs from a set of N
objects

ls_referential_integrity: Removing parameter for object

ls_referential_integrity: Removing sensor-group pair sensor / group

ls_referential_integrity: Starting location system referential
integrity checker
```

Trace option ‘ls_child_has_timing_issue_checker

This traces computation of the ChildHasTimingIssue flag.

```
ls_child_has_timing_issue_checker: asserting flag for mac value

ls_child_has_timing_issue_checker: on_current_status_changed(sensor,
status)

ls_child_has_timing_issue_checker: on_current_status_removed(sensor)

ls_child_has_timing_issue_checker: on_location_cell_changed(sensor,
location cell)

ls_child_has_timing_issue_checker: on_location_cell_removed(sensor)

ls_child_has_timing_issue_checker: on_upstream_sensor_changed(parent,
child)
```

```
ls_child_has_timing_issue_checker: on_upstream_sensor_removed(parent,
child)
```

```
ls_child_has_timing_issue_checker: Sensor/Location Cell descriptor
not found
```

```
ls_child_has_timing_issue_checker: Sensor/Status/Error Flags
descriptor not found
```

Trace option ‘tftp_report_d’

This option contains all the data in the tftp_report (which is enabled by default) and some additional information for debugging. It has one format:

tftp_report_d: boot_server: *for each statistic statistic_name: count/current state*

Configuration distribution

There are several distinct trace options for tracing the configuration distribution protocol.

N.B. The config distribution server checks the value of the platform_monitor variable approximately every minute, and updates what trace streams are enabled. This means that the config distribution servers do not need to be restarted to change which trace streams are enabled. Additionally, the config distribution streams (i.e. streams starting with ‘ls_cfgdist’) all output the server’s cell at the start of the message.

```
ls_cfgdist_actions: execute failed, but no state for action
```

```
ls_cfgdist_actions: execute failed, but old session for ( old_session
/ new_session ) action
```

```
ls_cfgdist_actions: execute failed, will retry for action
```

```
ls_cfgdist_actions: execute successful for action
```

```
ls_cfgdist_actions: execute successful, but no state for action
```

ls_cfgdist_actions: execute successful, but old session (*old_session / new_session*) for *action*

ls_cfgdist_interests: operation : changed reqs/sgrps/ns/ds/locs/geoms/macs *reqs/sgrps/ns/ds/locs/geoms/macs*

ls_cfgdist_interests_d: *mac_count* MACs interested in *descriptor obj : value*

ls_cfgdist_interests_d: Descriptor changed: *descriptor*

ls_cfgdist_interests_d: getting establish for *mac* .

ls_cfgdist_interests_d: handling *namespace_count* changed namespaces and *descriptor_count* changed descriptors.

ls_cfgdist_interests_d: handling changed requests for *mac_count* macs.

ls_cfgdist_interests_d: handling changed sensor groups for *changed_ sensors_count* sensors, *changed_groups_count* groups.

ls_cfgdist_interests_d: MAC requests don’t match group *group* : *differences*

ls_cfgdist_interests_d: MAC requests match group *group*

ls_cfgdist_interests_d: MACHasRequest changed for *mac*

ls_cfgdist_interests_d: Namespace updated: *namespace*

ls_cfgdist_interests_d: No MACs interested in *descriptor obj*

ls_cfgdist_interests_d: on_commit

ls_cfgdist_interests_d: on_establish

ls_cfgdist_interests_d: on_macs_changed

ls_cfgdist_interests_d: reload request for *mac* : created new group *new_mac_group* with *request_count* requests (previous group: *old_mac_group*).

ls_cfgdist_interests_d: reload request for *mac* : re-using group *new_mac_group* (previous group: *old_mac_group*).

ls_cfgdist_interests_d: reload request: mac *mac* has no request, removed from mac group *group* . New group size: *new_size* .

ls_cfgdist_interests_d: Sensor geometry cell changed: sensor *sensor*

ls_cfgdist_interests_d: Sensor location cell changed: sensor *sensor*

ls_cfgdist_interests_d: SensorHasMAC changed for *mac*

ls_cfgdist_interests_d: SensorInGroup change for sensor *sensor* group *group*

ls_cfgdist_interests_d: unable to find group interests for *mac* (group= *group*).

ls_cfgdist_interests_d: unable to find mac group for *mac* .

ls_cfgdist_macs: *operation [continuation]* : #changed_macs= *changed_macs_count* #changed_sensors= *changed_sensors_count* #added= *added_macs_count* #removed= *removed_macs_count* #callbacks= *callbacks_count* added= *added_macs_set* removed= *removed_macs_set*

ls_cfgdist_state: *operation* : macs_with_changed_interests= *changed_macs_count* added_macs= *added_macs_count* removed_macs= *removed_macs_count* changed_params= *changed_params_count* removed_params= *removed_params_count* actions est/upd= *establish_action_count / update_action_count*

ls_cfgdist_state_d: new state for *mac* resetting last service time

ls_cfgdist_state_d: Parameter *param* removed for object: *macs_count* MACs interested

ls_cfgdist_state_d: Parameter *param* changed for object: *macs_count* MACs interested

ls_cfgdist_state_dd: on_commit: deleting changes for *mac* as no request found.

ls_cfgdist_state_dd: on_commit: ignoring changes for *mac* as no management state.

ls_cfgdist_state_dd: on_commit: pushing update action for *mac* , with *updated_parameters_count* updates, *removed_parameters_count* removals.

ls_cfg_server_cell_checker: checking cells: cell extents *[not]* changed cell config *[not]* changed Sensor/Location Cell rows to check: *sensor_loc_cell_changed_count* Sensor/Geometry Cell rows to check:

sensor_geom_cell_changed_count Sensor/Status/Error Flags rows to check: *sensor_error_flags_changed_count* Location Cell/Named rows to check: *loc_cell_named_changed_count* Geometry Cell/Named rows to check: *geom_cell_named_changed_count*

ls_cfg_server_cell_checker_d: *cell* named changed

ls_cfg_server_cell_checker_d: *sensor* error flags changed

ls_cfg_server_cell_checker_d: *sensor* geometry cell changed

ls_cfg_server_cell_checker_d: *sensor* location cell changed

ls_cfg_server_cell_checker_d: *sensor* position changed

ls_cfg_server_cell_checker_d: asserting *required_cell* for *sensor* as *required_cell_static_type*

ls_cfg_server_cell_checker_d: cell config changed

ls_cfg_server_cell_checker_d: cell extents changed

ls_cfg_server_cell_checker_d: checking cells

ls_cfg_server_cell_checker_d: deleting *required_cell_static_type* for *sensor*

ls_cfg_server_timing_root_d: adding sensor – *sensor*

ls_cfg_server_timing_root_d: adding sensor – *upstream_sensor*

ls_cfg_server_timing_root_d: Build sensor upstream map

ls_cfg_server_timing_root_d: Calculating timing roots

ls_cfg_server_timing_root_d: considering *sensor*

ls_cfg_server_timing_root_d: followed timing tree but found no acting TS, giving up

ls_cfg_server_timing_root_d: upstream sensor *it->second* is acting TS

ls_cfg_server_timing_root_d: upstream sensor *it->second* is not acting TS

ls_config_server_reg_debug: SensorRegistrationServer::execute: (register_sensor) result= *(int)result*

ls_config_server_reg_debug: SensorRegistrationServer::execute: (unknown op= *op*)

ls_config_server_reg_debug: SensorRegistrationServer::execute: assert_eeprom_values(*EEPROMValuesPrinter(processor_values)* , *EEPROMValuesPrinter(receiver_values)*).

ls_config_server_reg_debug: SensorRegistrationServer::execute: enter

ls_config_server_reg_debug: SensorRegistrationServer::execute: mac_assignment_complete(*instruction.old_mac_ -> instruction.new_mac_* processor *instruction.processor_id_.manufacturer_info1_ / instruction.processor_id_.manufacturer_info2_ / instruction.processor_id_.manufacturer_info3_* receiver *instruction.receiver_id_.manufacturer_info1_ / instruction.receiver_*

```
id_.manufacturer_info2_ / instruction.receiver_id_.manufacturer_
info3_ ).
```

```
ls_config_server_reg_debug: SensorRegistrationServer::execute:
request.get_error()= request.get_error()
```

```
ls_config_server_reg_debug: SensorRegistrationServer::execute: set_
cnc_status( mac , cnc_seq , valid ? true : false )
```

Sensor low-level debugging messages

There are several low-level tracing options available whose main purpose is debugging sensor operation. They should not normally be enabled but are included here to give an idea of what could be traced in principle.

Trace option ‘sensor_cnc’

This is used to trace the custom network configuration protocol.

```
sensor_cnc: config_client_log: value
```

```
sensor_cnc: Ignoring invalid remote CNC = remote_cnc_state.cnc_ using
local = local_cnc
```

```
sensor_cnc: Local CNC == Remote CNC = remote_cnc_state.cnc_ .
```

```
sensor_cnc: Local CNC == Remote CNC == empty.
```

Trace option ‘sensor_config’

This is used to trace the configuration protocol.

```
sensor_config: change_message: changed= changed_params.size() ,
components= components.size()
```

```
sensor_config: establish_message: params= message.parameters_.size()
, components= component_properties_.size() , notifier_comps=
notifier_components.size()
```

```
sensor_config: establish_message: prefix= message.prefix_.prefix_ ,
our_prefix= prefix_ , request.get_error()= request.get_error()
```

```
sensor_config: notifier: components= components.size()
```

```
sensor_config: unknown_message: request.get_error()= request.get_
error() , message_code= message_code
```

```
sensor_config: update_message: prefix= message.prefix_.prefix_ , our_
prefix= prefix_ , request.get_error()= request.get_error() , updated=
message.updated_parameters_.size() , removed= message.removed_
parameters_.size()
```

Trace option ‘sensor_sw’

This is used to trace the firmware and software flash update process.

```
sensor_sw: Flashed firmware/kernel/fs: flashed_firmware_version_ /
flashed_kernel_version_ / flashed_fs_version_
```

```
sensor_sw: No firmware upgrade requested.
```

```
sensor_sw: No software upgrade requested.
```

```
sensor_sw: Storing filesystem details in EEPROM (version/size/crc):
available_fs_version / available_fs_size / available_fs_crc
```

```
sensor_sw: Storing kernel details in EEPROM (version/size/crc):
available_kernel_version / available_kernel_size / available_kernel_
crc
```

```
sensor_sw: Verifying filesystem CRC from SPI flash
```

```
sensor_sw: Verifying kernel CRC from SPI flash
```

```
sensor_sw: Writing filesystem to SPI flash
```

```
sensor_sw: Writing kernel to SPI flash
```

Trace option ‘tftp_sender’

This option traces the TFTP requests that the ARM makes to download new firmware or software to write into flash.

```
tftp_sender: Requesting filename from address max_attempts send_
attempts timeout_period_ms timeout_in_ms adaptive_timeout adaptive_
timeout
```

```
tftp_sender: Request for filename from address has failed, received
block_count blocks.
```

```
tftp_sender: Request for filename from address has completed.
```

Location platform warning messages

Some important warning messages are provided by the underlying location platform libraries and are common to all Ubisense services, including DIMENSION4 services.

Thread scheduling delays

These events will be reported if a process detects that it is not being scheduled promptly. For example, if it requests a sleep of a certain length and is in fact woken up some time after the

sleep time has expired, then this may cause a warning if the delay in waking the process up is too large.

```
warning: slow thread scheduling in last interval s; count events;
mean mean_delay ms; max max_delay ms at time_of_max_delay
```

Disk write latency

Disk write latencies are reported if a process detects that the write operation took too long to complete.

```
warning: immediate disk write latency report for 'file_name':
detected latency of latency_ms milliseconds doing operation (handle:
file_handle ).
```

```
warning: periodic disk write latency report for 'file_name': highest
latency was stats_.max_interval_latency_ms milliseconds doing max_
interval_latency_operation in the last interval seconds (handle:
file_handle ).
```