Zeta Documentation

Release 0.1.0-alpha

Lucas Peixoto, Rodrigo Peixoto

Aug 27, 2021

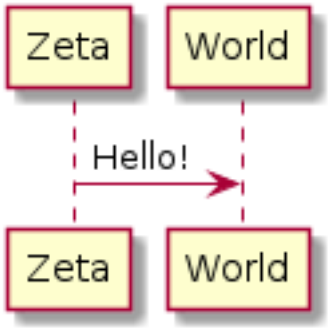
CONTENTS:

CHAPTER

ONE

GETTING STARTED

This is the first step to do.



CHAPTER

TWO

CODE DOCUMENTATION

This is the temporary code documentation.

zeta.zeta.Channel : public object

Represents a channel as written in the YAML file.

Public Functions

__init__()

__repr__(*self*)

Public Members

name

read_only

on_changed

size

persistent

sem

id

initial_value

message

message_obj

pub_services_obj

sub_services_obj

zeta.zeta.Config : public object

Public Functions

__init__()

Public Members

sector_count

storage_partition

storage_period

union data

#include <zeta.template.h>

Public Members

zt_data_int8_t **s8**

zt_data_uint8_t **u8**

zt_data_int16_t **s16**

zt_data_uint16_t **u16**

zt_data_int32_t **s32**

zt_data_uint32_t **u32**

zt_data_int64_t **s64**

zt_data_uint64_t **u64**

zt_data_bytes_t **bytes**

zeta.zeta_cli.FileFactory **public object**

```
Represents a class to generate a file-based
in a template and your respective substitutions.
```

Subclassed by *HeaderFileFactory*, *SourceFileFactory*, *ZetaConf*

Public Functions

__init__()

create_substitutions(*self*)

```
Creates substitutions to be implemented by classes inherited.

:returns: None
:rtype: None
```

generate_file(*self*)

```
Writes output file with the respective substitutions
assigned in create_substitutions function.

:returns: None
:rtype: None
```

run(self)

```
Run the routine responsible for assigns substitutions and
write the output file.

:returns: None
:rtype: None
```

Public Members

destination_file

template_file

zeta

substitutions

union flag_data

#include <*zeta.template.h*> Define pendent options that a channel can have.

Public Members

uint8_t **pend_persistent**
: Active represent that channel must be saved in flash by zeta_thread_nvs

uint8_t **pend_callback**
: Active represent that services callbacks from subscribers must be called by zeta_thread

uint8_t **on_changed**
: Active represent that the service callback will be called on change and not on update

uint8_t **source_serial_isc**
: When active indicates the channel was changed by the serial isc (some host process) instead of a target service

struct *flag_data*::**[anonymous] field**

uint8_t **data**
: Raw data

class zeta.zeta_cli.**HeaderFileFactory**(**public FileFactory**)

```
Responsible for creating header files that will
be used by Zeta.
```

Subclassed by *ZetaHeader*

Public Functions

__init__()

class zeta.zeta_isc.**IPC**

Public Functions

__init__()

create_channels(*self*)

digest_packet()

read_channel(*self*, *cid*)

set_channel(*self*, *cid*, *msg*)

Public Static Attributes

sem

channel_changed_queue

Private Static Attributes

__zeta

__channels

class zeta.zeta_isc.**IPCChannel**

Public Functions

__init__()

__repr__(*self*)

message_union_field(*self*)

create_struct()

Public Members

metadata

data

zeta.zeta_pkt_IPCHeader

```
struct zeta_pkt_IPCHeader{: public LittleEndianStructure
    uint8_t channel : 8; /**!> 256 channels available*/
    uint8_t op : 3; /**!> 0: read, 1: write, 2: read response,
                        3: write response, 4: update */
    uint8_t status : 4;   /**!> 0: ok, 1: failed */
    uint8_t has_data : 1; /**!> 0: no data, 1: contains data */
};
```

Private Static Attributes

pack

fields

zeta.zeta_pkt.IPCHeaderDataInfo : public LittleEndianStructure

```
struct __attribute__((__packed__)) IPCHeaderDataInfo {
    uint8_t crc : 8;  /**!> CCITT 8, polynom 0x07, initial value 0x00 */
    uint8_t size : 8; /**!> data size */
};
```

Private Static Attributes

pack

fields

zeta.zeta_pkt.IPCPacket : public LittleEndianStructure

Public Functions

__init__(*self*, *args*, *kwargs*)

set_data()

set_data_with_struct(*self*, *struct*)

clear_data(*self*)

from_bytes(*cls*, *raw_data*)

to_bytes(*self*)

data(*self*)

set_header()

set_data_info()

set_header_from_bytes()

set_data_info_from_bytes()

header_to_bytes(*self*)

__repr__(*self*)

struct_contents(*self*, *struct*)

struct_contents_set(*self*, *struct*)

Public Static Attributes

OP_READ

OP_WRITE

OP_READ_RESPONSE

OP_WRITE_RESPONSE

OP_UPDATE

OP_DEBUG

DATA_UNAVALABLE

DATA_AVAILABLE

STATUS_OK

STATUS_FAILED

Private Members

__data

Private Static Attributes

pack

fields

class zeta.zeta_isc.**Message**(*object*)

This object represents the Message defined by the Zeta yaml file.

Public Functions

__init__()

__repr__(*self*)

Public Members

name

msg_format

class zeta.zeta_isc.**SerialDataHandler**

Public Functions

__init__(*self*)

append(*self*, *data*)

send_command()

digest(*self*)

run(*self*)

Public Members

iqueue

oqueue

Public Static Attributes

STATE_DIGEST_HEADER_OP

STATE_DIGEST_HEADER_DATA_INFO

STATE_DIGEST_BODY

Private Members

__buffer

__state

__current_pkt

zeta.Service public object

Represents a service built from a YAML file.

Public Functions

__init__()

Public Members

name

priority

stack_size

pub_channels_names

sub_channels_names

pub_channels_obj

sub_channels_obj

zeta.zeta_cli.SourceFileFactory : public FileFactory

Responsible for creating source files that will be used by Zeta.

Subclassed by *ZetaSource*

Public Functions

__init__()

zeta.messages.TestStringMethods : public TestCase

Public Functions

test_primitives(*self*)

test_struct_simple(*self*)

test_union_simple(*self*)

test_bitarray_simple(*self*)

test_array_simple(*self*)

test_complex_message(*self*)

zeta.Zeta : public object

Represents the Zeta object that has access to services, channels and config parameters specified in YAML file.

Public Functions

__init__()

Public Members

config

channels

services

messages

Private Functions

__check_service_channel_relation(*self*)

Checks if the service is registered or is used some nonexistent channel.

:returns: None
:rtype: None
:raise ZetaCLIError: Channel name doesn't exists in channel list.

__check_channel_message_relation(*self*)

```
Checks if the channel_name exists or is used some
nonexistent channel.

:returns: None
:rtype: None
:raise ZetaCLIError: Channel name doesn't exists in channel list.
```

__process_file()

zeta.zeta_cli.ZetaCLI public object

```
Represents the ZetaCLI and has all the callbacks that will be
called when the user type zeta on the terminal.
```

Public Functions

__init__(*self*)

```
ZetaCLI constructor.

:returns: None
:rtype: None
```

init(*self*)

```
Callback when the user type "zeta init" and is responsible for
generates the minimum requirements in order to Zeta works
properly.

:returns: Exit code
:rtype: int
:raise ZetaCLIError: Error in zeta.cmake or zeta.yaml file path
```

version(*self*)

```
Callback when the user type "zeta version" and is responsible
for sends out the ZetaCLI version.

:returns: Exit code
:rtype: int
```

check_files(*self*)

check_code(*self*, *src_dir*)

check(*self*)

```
Callback when the user type "zeta check" and is responsible for
checks if the needed steps were made by user in order to Zeta
works properly.

:returns: Exit code
:rtype: int
```

services(*self*)

```
Callback when the user type "zeta services" and is responsible
for generates files template-based with the services initialized.

:returns: Exit code
:rtype: int
:raise ZetaCLIError: Error opening or reading files
```

generate(*self*)

```
Generate all the internal files that represents Zeta system
like channels, Zeta threads, Zeta API, and others.
```

(continues on next page)

(continued from previous page)

```
        :returns: Exit code
        :rtype: int
```

sniffer(*self*)

```
        Sniffs all the internal files that represents Zeta system
        like channels, Zeta threads, Zeta API, and others.

        :returns: Exit code
        :rtype: int
```

isc(*self*)

```
        Starts a high level ISC on the host. It takes all the yaml file
        details to run the proper environment.

        :returns: Exit code
        :rtype: int
```

zeta.cli.cmdparser.ZetaCLIError **public Exception**

```
    Base class for ZetaCLIError in Zeta
```

Public Functions

__init__(*self*, *message*, *errcode*)

```
        Initializer.

        :param message: Message to be displayed printing error
        :param errcode: Error code
        :returns: None
        :rtype: None
```

handle(*self*)

```
        Responsible for handling the raise call.

        :returns: None
        :rtype: None
```

__str__(*self*)

Public Members

errcode

message

zeta.sniffer.ZetaCLIEventListWalker **public ListWalker**

```
    Sniffer's Zeta CLI Event ListWalker used public ListWalker show
    the events occurred.
```

Public Functions

__init__()

create_table(*self*, *title='[no title]'*)

```
Create a table.

:param title: the table's title
:returns: the table container (LineBox)
:rtype: urwid.LineBox
```

__len__(*self*)

```
Implement len() operator overloading

:returns: the table number of lines
:rtype: int
```

append(*self*, *item*)

```
Implement append method overloading

:param item: dictionary with the data of the new item to be added
:returns: None
:rtype: None
```

create_data_viewer(*self*)

```
Create the data viewer.

:returns: the data viewer
:rtype: urwid.Text
```

__getitem__(*self*, *index*)

```
Implement __getitem__(self, index) overloading

:param index: index of the desired element
:returns: the element at index
:rtype: urwid.Columns
```

set_modified_callback(*self*, *callback*)

```
Not implemented: MonitoredList is not
implemented in SimpleFocusListWalker.
Use connect_signal(list_walker, "modified", ...) instead.
```

set_focus(*self*, *position*)

```
Set the focus position.
```

next_position(*self*, *position*)

```
Return position after start_from.
```

prev_position(*self*, *position*)

```
Return position before start_from.
```

positions(*self*, *reverse=False*)

```
Optional: Method returning an iterable of positions.
```

Public Members

event_list

table_rows

focus

table_title

data_viewer

Private Functions

__create_table_header(*self*)

```
Create table header

:param event_list: list of event items
:returns: table header
:rtype: urwid.AttrMap
```

zeta.zeta_cli.ZetaConf : public FileFactory

Public Functions

__init__()

create_substitutions(*self*)

```
Reads the template file and has the
goal to assigns all the substitutions needed to Zeta works
properly.
```

zeta.zeta_cli.ZetaHeader : public HeaderFileFactory

Public Functions

__init__()

create_substitutions(*self*)

```
Assigns all the needed substitutions to be
written on the output file.

:returns: None
:rtype: None
```

Public Members

services_reference

max_channel_size

class zeta.messages.ZetaMessage

```
Represents a Message. It holds the messages' information for
code generation
```

Public Functions

__init__()

__repr__(*self*)

digest_type(*self*)

```
Method that creates the mtype and mounts the necessary data to the code
generation.

:returns: None
:rtype: None
```

code(*self*)

```
Generates the code for the message.

:returns: the code generated that represents the message
:rtype: str
```

Public Members

name

size

level

description

parent

mtype_obj

mtype_base_type

mtype

fields

zeta.sniffer.ZetaSerialMonitor(*public_protocol*)

```
Class that creates a serial monitor protocol that updates
the interface data related to the income information.
```

Public Functions

__init__()

connection_made(*self*, *transport*)

```
Method called when a connection is made
```

digest_isc_message(*self*, *msg*)

```
Method that digests the encoded message to extract the message
information.
```

data_received(*self*, *data*)

```
Method called when an a new data is available to be read.

:param data: bytes received at the serial.
:returns: None
:rtype: None
```

connection_lost(*self*, *exc*)

```
Method called when the connection is lost
```

pause_writing(*self*)
Event handler when the writing is paused

resume_writing(*self*)
Event handler when the writing is resumed

send(*self*)
Send four newline-terminated messages, one byte at a time.

Public Members

event_list

logging_ui

line_buffer

zeta

transport

class zeta.sniffer.**ZetaSniffer**
Class that encapsulates the sniffer execution function.

Public Functions

run(*self*, *serial_port=''*, *baudrate=115200*)
Function that runs the sniffer. It opens the serial at the specified baud rate passed as argument.

:param serial_port: the serial port description. For example "/dev/ttyUSB0".
:param baudrate: the communication frequency. For example 115200.
:returns: None
:rtype: None

zeta.zeta_cli.**ZetaSource**
Base class that :public SourceFileFactory has the goal to assigns all the substitutions needed to Zeta works properly.

Public Functions

__init__()

gen_sems(*self*)
Helper function for assigns the channel semaphores.

:returns: None
:rtype: None

gen_creation(*self*)
Helper function for creates all the channels that will be used by Zeta.

:returns: None
:rtype: None

gen_nvs_config(*self*)
Helper function for assigns the nvs config.

:returns: None
:rtype: None

create_substitutions(*self*)
Create the needed substitutions to be written on the output file.

:returns: None
:rtype: None

Public Members

channels_creation

channels_sems

sector_size

sector_count

storage_offset

set_publishers

set_subscribers

arrays_init

services_array_init

run_services

storage_partition

zeta.ZetaYamlLoader : **public SafeLoader**
Main class ZetaYamlLoader object to generate a correct dictionary-based in the YAML file.

Public Functions

__init__(*self*, *stream*)
Initialize the YamlReader constructor.

:param stream: default param used for yaml.SafeLoader
:returns: None
:rtype: None

ref(*self*, *node*)
Parse ref statements in YAML file and generate a valid reference

:param node: Ref statement with the reference name
:returns: Reference name
:rtype: str

include(*self*, *node*)
Parse include statements in YAML file and generate a valid reference

:param node: Include statement with the reference name
:returns: Content of the file passed by include
:rtype: dict

struct zt_channel
#include <*zeta.template.h*> Define Zeta channel type.

Public Members

const char ***name**
Channel name

uint8_t ***data**
Channel raw data

uint8_t **read_only**

uint8_t **size**
Channel size

uint8_t **persistent**
Persistent type

zt_channel_e **id**
Channel Id

union *flag_data* **flag**
Options

struct k_sem ***sem**
Preserve shared-memory

zt_service_t ****publishers**
Publishers

zt_service_t ****subscribers**
Subscribers

struct zt_data_bytes_t
#include <zeta.template.h>

Public Members

size_t **size**

uint8_t **value**[]

struct zt_data_int16_t
#include <zeta.template.h>

Public Members

size_t **size**

int16_t **value**

struct zt_data_int32_t
#include <zeta.template.h>

Public Members

size_t **size**

int32_t **value**

struct zt_data_int64_t
#include <zeta.template.h>

Public Members

size_t **size**

int64_t **value**

struct zt_data_int8_t
#include <zeta.template.h>

Public Members

size_t **size**

int8_t **value**

struct zt_data_uint16_t
#include <zeta.template.h>

Public Members

size_t **size**

uint16_t **value**

struct zt_data_uint32_t
#include <zeta.template.h>

Public Members

size_t **size**

uint32_t **value**

struct zt_data_uint64_t
#include <zeta.template.h>

Public Members

size_t **size**

uint64_t **value**

struct zt_data_uint8_t
#include <zeta.template.h>

Public Members

size_t **size**

uint8_t **value**

struct zt_isc_net_pkt

Public Members

struct *zt_isc_net_pkt_header* **header**

struct *zt_isc_net_pkt_header_message_info* **message_info**

uint8_t ***message**

struct zt_isc_net_pkt_header

Public Members

uint8_t **channel**

uint8_t **op**
!> 256 channels available

uint8_t **status**
!> 0: read, 1: write, 2: read response, 3: write response, 4: update

uint8_t **has_data**
!> 0: ok, 1: failed

struct zt_isc_net_pkt_header_message_info

Public Members

uint8_t **crc**

uint8_t **size**
!> CCITT 8, polynom 0x07, initial value 0x00

struct zt_isc_packet
#include <zeta.template.h>

Public Members

uint32_t **id**

uint8_t **service_id**

uint8_t **channel_id**

uint8_t **op**

uint8_t **size**

uint8_t **message**[$max_channel_size]

zeta_isc_ZT_MSG **public Union**

```
Zeta message definition by the config file will be filled up here
```

struct zt_service
#include <zeta.template.h> Define Zeta service type.

Public Members

zt_service_e **id**
Service ID

const char ***name**
Service name

struct k_thread **thread**
Service RTOS thread

zt_callback_f **cb**
Service callback

k_thread_entry_t **entry_point**
Service thread function (entry point)

k_thread_stack_t ***stack**
Service thread stack

size_t **stack_size**
Service thread stack size

module zeta

module __main__

module zeta.**_version**

Variables

str

module zeta.**messages**

Variables

MsgType = namedtuple('MsgCode', "statement separator begin end sizable")

module sniffer

module zeta.**zeta**

Variables

ZETA_MODULE_DIR = "."

ZETA_TEMPLATES_DIR = "."

PROJECT_DIR = "."

ZETA_DIR = "."

ZETA_SRC_DIR = "."

ZETA_INCLUDE_DIR = "."

module zeta.zeta_cli

Functions

run()

Variables

ZETA_MODULE_DIR = "."

ZETA_TEMPLATES_DIR = "."

PROJECT_DIR = str(Path(".").resolve())

ZETA_DIR = "."

ZETA_SRC_DIR = "."

ZETA_INCLUDE_DIR = "."

OK_COLORED = "\033[0;42m \033[1;97mOK \033[0m"

FAIL_COLORED = "\033[0;41m \033[1;97mFAIL \033[0m"

WARNING_COLORED = "\033[1;43m \033[1;97mWARNING \033[0m"

module zeta.zeta_errors

Variables

EZTFILE = 1

EZTFIELD = 2

EZTCHECKFAILED = 3

EZTINVREF = 4

EZTUNEXP = 10

module zeta.zeta_isc

Functions

repr(*self*)

create_base_message()

uart_write_handler()

uart_read_handler()

struct_contents(*struct*)

struct_contents_set(*struct*, *raw_data*)

callback_handler()

pub_read_handler()

isc_run()

Variables

ipc = None

context = Context.instance()

target_attached = False

module zeta_pkt

file **__init__.py**

file **__main__.py**

file **_version.py**

file **messages.py**

file **sniffer.py**

file **zeta.template.c**

#include "zeta.h"#include <drivers/flash.h>#include <fs/nvs.h>#include <logging/log.h>#include <storage/flash_map.h>#include <string.h>#include <zephyr.h>#include "zt_serial_ipc.h"#include "device-tree_fixups.h"

Defines

NVS_SECTOR_COUNT

NVS_STORAGE_PARTITION

Functions

LOG_MODULE_REGISTER(zeta, CONFIG_ZETA_LOG_LEVEL)

static void **__zt_monitor_thread**(void)

K_THREAD_DEFINE (zt_monitor_thread_id, ZT_MONITOR_THREAD_STACK_SIZE, __zt_monitor_thre

K_MSGQ_DEFINE (zt_channels_changed_msgq, sizeof(uint8_t), 30, 4)

const char ***zt_channel_name**(zt_channel_e *id*, int **error*)

Return the channel name.

Parameters

- **id** – Channel Id
- **error** – Handle possible errors

Returns Channel name

size_t **zt_channel_size**(zt_channel_e *id*, int **error*)

Return the channel size.

Parameters

- **id** – Channel Id
- **error** – Handle possible errors

Returns Channel size

int **zt_chan_read**(zt_channel_e *id*, *zt_data_t* **channel_data*)
Read channel value.

Parameters

- **id** – Channel Id
- **channel_data** – pointer to a zt_data_t where the data will be retrieved.

Returns -ENODATA The channel was not found

Returns -EFAULT Channel value is NULL

Returns -EPERM Channel hasn't read function implemented

Returns -EINVAL Size passed is different to channel size

Returns Error code

int **zt_chan_pub**(zt_channel_e *id*, *zt_data_t* **channel_data*)
Publish channel value.

Parameters

- **id** – Channel Id
- **channel_data** – pointer to a zt_data_t where the data is.

Returns -ENODATA The channel was not found

Returns -EACCESS Current thread hasn't permission to publish this channel

Returns -EFAULT Channel value is NULL

Returns -EPERM Channel is read only

Returns -EINVAL Size passed is different to channel size

Returns -EAGAIN Valid function returns false

Returns Error code

file **zeta.template.h**
#include <stddef.h>#include <zephyr.h>#include <zephyr/types.h>

Defines

ZT_MONITOR_THREAD_STACK_SIZE
Stack size that is used in Zeta thread that manages channels callback calls.

ZT_STORAGE_SLEEP_TIME
Storage sleep time.

ZT_MONITOR_THREAD_PRIORITY
Channels thread priority.

ZT_SERVICE_DECLARE(*_name*, *_task*, *_cb*)
Declare a zeta service.

Parameters

- **_name** – Service name
- **_task** – Task pointer function
- **_cb** – Callback to be called when some subscribed channel change

ZT_SERVICE_RUN(*_name*)
Run a zeta service.

Parameters

- **_name** – Service name

ZT_VARIABLE_REF_SIZE(*x*)
Read variable reference and size easily \ to use in Zeta API. \ .

\

Parameters

- **x** – variable name \ \

ZT_CHECK_VAL(*_p*, *_e*, *_err*, ...)
Check if _v value is equal to _c, otherwise _err will be returned and a message will be sent to LOG.

Parameters

- **_v** – Value
- **_c** – Condition
- **_err** – Error code

ZT_CHECK(*_p*, *_err*, ...)
Check if _v is true, otherwise _err will be returned and a message will be sent to LOG.

Parameters

- **_v** – Value
- **_err** – Error code

Returns

ZT_DATA_S8(*data*)

ZT_DATA_U8(*data*)

ZT_DATA_S16(*data*)

ZT_DATA_U16(*data*)

ZT_DATA_S32(*data*)

ZT_DATA_U32(*data*)

ZT_DATA_S64(*data*)

ZT_DATA_U64(*data*)

ZT_DATA_BYTES(*_size*, *data*, ...)

Typedefs

typedef union *data* **zt_data_t**

typedef void (***zt_callback_f**) (zt_channel_e id)
zeta_callback_f define the callback function type of Zeta.

Parameters **id** – Channel Id.

typedef struct *zt_service* **zt_service_t**

typedef struct *zt_isc_packet* **zt_isc_packet_t**

typedef struct *zt_channel* **zt_channel_t**

Functions

struct zt_isc_packet __attribute__ ((packed))

size_t **zt_channel_size** (zt_channel_e *id*, int **error*)
Return the channel size.

Parameters

- **id** – Channel Id
- **error** – Handle possible errors

Returns Channel size

const char ***zt_channel_name** (zt_channel_e *id*, int **error*)
Return the channel name.

Parameters

- **id** – Channel Id
- **error** – Handle possible errors

Returns Channel name

int **zt_chan_read** (zt_channel_e *id*, *zt_data_t* **channel_data*)
Read channel value.

Parameters

- **id** – Channel Id
- **channel_data** – pointer to a zt_data_t where the data will be retrieved.

Returns -ENODATA The channel was not found

Returns -EFAULT Channel value is NULL

Returns -EPERM Channel hasn't read function implemented

Returns -EINVAL Size passed is different to channel size

Returns Error code

int **zt_chan_pub** (zt_channel_e *id*, *zt_data_t* **channel_data*)
Publish channel value.

Parameters

- **id** – Channel Id

- **channel_data** – pointer to a zt_data_t where the data is.

Returns -ENODATA The channel was not found

Returns -EACCESS Current thread hasn't permission to publish this channel

Returns -EFAULT Channel value is NULL

Returns -EPERM Channel is read only

Returns -EINVAL Size passed is different to channel size

Returns -EAGAIN Valid function returns false

Returns Error code

Variables

uint32_t **id**

uint8_t **service_id**

uint8_t **channel_id**

uint8_t **op**

uint8_t **size**

uint8_t **message**[$max_channel_size]

union flag_data __attribute__

file **zt_serial_ipc.h**

#include <zephyr.h>#include "zeta.h"

Defines

ZT_SERIAL_IPC_RX_THREAD_STACK_SIZE

ZT_SERIAL_RX_THREAD_PRIORITY

ZT_SERIAL_IPC_TX_THREAD_STACK_SIZE

ZT_SERIAL_TX_THREAD_PRIORITY

Functions

int **zt_serial_ipc_send_update_to_host**(zt_channel_e *id*)

k_tid_t **zt_serial_ipc_thread**()

file **zt_uart.h**

#include <device.h>#include <zephyr.h>

Functions

int **uart_open**(char *_dev_label_)

int **uart_write**(uint8_t *_data_, size_t _size_)

int **uart_write_str**(char *_str_)

int **uart_write_byte**(uint8_t *_byte_)

struct k_msgq ***uart_get_input_msgq**()

struct k_msgq ***uart_get_output_msgq**()

file **zt_serial_ipc.c**

#include "zt_serial_ipc.h"#include <string.h>#include <sys/crc.h>#include <sys/printk.h>#include <zephyr.h>#include <zt_uart.h>#include "devicetree.h"#include "kernel.h"#include "zeta.h"

Defines

UART_DEVICE_NAME

SPDX-License-Identifier: Apache-2.0

MAX_MESSAGE_LENGTH

ZT_ISC_PKT_ASSERT_EQ(*actual*, *expected*, *ret*)

ZT_ISC_PKT_ASSERT(*cond*, *ret*)

SELF_TEST_INIT()

SELF_TEST_ASSERT(*cond*)

SELF_TEST_FINISH()

NUMBER_OF_CHANNELS

ZT_ISC_NET_PKT_WITH_MSG_SIZE(*_size*)

Enums

enum zt_isc_net_pkt_op_t

Values:

enumerator OP_READ

enumerator OP_WRITE

enumerator OP_READ_RESPONSE

enumerator OP_WRITE_RESPONSE

enumerator OP_UPDATE

enumerator OP_DEBUG

enum pkt_data_availability_t

Values:

enumerator DATA_UNAVALABLE

enumerator DATA_AVAILABLE

enum pkt_status_t

Values:

enumerator STATUS_OK

enumerator STATUS_FAILED

Functions

K_MSGQ_DEFINE (__channel_updated_queue, 1, 16, 1)

void **zt_isc_net_pkt_clear**(**struct** *zt_isc_net_pkt* *self)

void **zt_isc_net_pkt_send**(**struct** *zt_isc_net_pkt* *self)

int **zt_isc_net_pkt_set_message**(**struct** *zt_isc_net_pkt* *self, char *message, size_t message_size)

void **zt_isc_net_pkt_calc_crc**(**struct** *zt_isc_net_pkt* *self)

uint8_t **zt_isc_net_pkt_header_is_valid**(**struct** *zt_isc_net_pkt_header* *header)

uint8_t **zt_isc_net_pkt_message_is_valid**(**struct** *zt_isc_net_pkt* *pkt)

int **digest_byte**(uint8_t *data*, **struct** *zt_isc_net_pkt* *pkt)

uint64_t **digest_byte_test**()

int **zt_serial_ipc_send_update_to_host**(zt_channel_e *id*)

void **__zt_serial_ipc_tx_thread**(void)

k_tid_t **zt_serial_ipc_thread**()

void **__zt_serial_ipc_rx_thread**(void)

K_THREAD_DEFINE (zt_serial_ipc_rx_id, ZT_SERIAL_IPC_RX_THREAD_STACK_SIZE, __zt_serial_

K_THREAD_DEFINE (zt_serial_ipc_tx_id, ZT_SERIAL_IPC_TX_THREAD_STACK_SIZE, __zt_serial_

Variables

static k_tid_t **__current_rx_thread**

file **zt_uart.c**

#include "zt_uart.h"#include <drivers/uart.h>#include <zephyr.h>

Functions

K_MSGQ_DEFINE (__input_msgq, sizeof(uint8_t), 512, 1)

K_MSGQ_DEFINE (__output_msgq, sizeof(uint8_t), 512, 1)

static void **uart_fifo_callback**(**const struct** device *dev, void *user_data)

struct k_msgq ***uart_get_input_msgq**()

struct k_msgq ***uart_get_output_msgq**()

int **uart_open**(char *dev_label)

int **uart_write**(uint8_t *data, size_t size)

int **uart_write_str**(char *str)

int **uart_write_byte**(uint8_t *byte)

Variables

static const struct device ***__uart_dev** = 0

const struct uart_config uart_cfg = {.baudrate = 115200,.parity = UART_CFG_PARITY_

file **zeta.py**

file **zeta_cli.py**

file **zeta_errors.py**

file **zeta_isc.py**

file **zeta_pkt.py**

dir **/home/docs/checkouts/readthedocs.org/user_builds/zeta-middleware/checkouts/latest/zeta/t**

dir **/home/docs/checkouts/readthedocs.org/user_builds/zeta-middleware/checkouts/latest/zeta/t**

dir **/home/docs/checkouts/readthedocs.org/user_builds/zeta-middleware/checkouts/latest/zeta/t**

dir **/home/docs/checkouts/readthedocs.org/user_builds/zeta-middleware/checkouts/latest/zeta/t**

dir **/home/docs/checkouts/readthedocs.org/user_builds/zeta-middleware/checkouts/latest/zeta**

CHAPTER

THREE

LICENSE

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z