IgH Master userspace program in ROS Documentation

Release 0.3.1

Mike Karamousadakis

Sep 06, 2018

Contents

Welcome to the documentation of IgH Master userspace real time program in ROS. If you would like to see an example of usage and installation instructions, see the Guide section. For help or new features or bugs, see information in the Support subsection of the Guide section.

CHAPTER 1

Contents

1.1 Files / Classes

1.1.1 Header Files

EtherCAT Communicator header file

Header file for the *EthercatCommunicator* class.

class EthercatCommunicator
#include <ethercat_communicator.h> The Ethercat Communicator class.

Basic class for implementing realtime pure communication purposes, from our application to the Ethercat slaves, via IgH Master module. The class uses the POSIX API for gaining realtime attributes.

Public Functions

void **init**(ros::NodeHandle &*n*)
Initializes the main thread.

Mostly makes ready the attributes of the realtime thread, before running.

Parameters

- n: The ROS Node Handle

void **start**()
Starts the main thread.

The function that actually starts the realtime thread. The realtime attributes have been set from *init*. Implements the basic realtime communication (Tx/Rx) with the EtherCAT slaves. Doesn't change the output PDOs. Basic state machine:

- Receive the new PDOs in domain1_pd from the IgH Master Module (and therefore from the EtherCAT slaves)
- Move to the domain_pd the output data of process_data_buf, safely
- Publish the "raw" data (not linked to EtherCAT variables) in PDOs received from the domain1_pd, to the /ethercat_data_raw topic
- Synchronize the DC of every slave (every *count'nth* cycle)
- Send the new PDOs from domain1_pd to the IgH Master Module (and then to EtherCAT slaves)

  See void *init(ros::NodeHandle &n)*

void **stop** ()
Stops the main thread.

This function stops the execution of the realtime thread. The mechanism for stopping it, is provided by the POSIX API. Search for *pthread_testcancel()* and other related functions.

Public Static Functions

static bool **has_running_thread** ()
A getter for knowing if there is a running thread.

It's used from the EthercatCommd service, to know if a user stops/starts an already stopped/started *EthercatCommunicator*.

Private Members

pthread_attr_t **current_thattr_**

struct sched_param **sched_param_**

Private Static Functions

void ***run** (void **arg*)

void **cleanup_handler** (void **arg*)

void **copy_data_to_domain_buf** ()

void **publish_raw_data** ()

Private Static Attributes

int **cleanup_pop_arg_** = 0

pthread_t **communicator_thread_** = {}

ros::Publisher **data_raw_pub_**

bool **running_thread_** = false

EtherCAT Input Data Handler header file

Header file for the *EthercatInputDataHandler* class.

class EthercatInputDataHandler
#include <ethercat_input_data_handler.h> The Ethercat Input Data Handler class.

Used for trasforming the "raw" indexed data from the */ethercat_data_raw* topic, sent by the Ethercat Communicator, to values of variables, and stream them to the */ethercat_data_slave_{slave_id}* topic.

Public Functions

void **init** (ros::NodeHandle &*n*)
Initialization Method.

Used for initializing the *EthercatInputDataHandler* object. It's basically the main method in the class, which initializes the listener to the afore mentioned topic.

Parameters

- `n`: The ROS Node Handle

void **raw_data_callback** (**const** ighm_ros::EthercatRawData::ConstPtr &*ethercat_data_raw*)
Raw Data Callback.

This method, is called when there are data in the */ethercat_data_raw* topic. Should the EtherCAT application change, this callback must change also. Implements the basic functionality of the class, to transform the "raw" data into variable values and pipe them into another topic.

Parameters

- `ethercat_data_raw`: A copy of the actual data sent to the topic */ethercat_data_raw*.

Private Members

ros::Subscriber **data_raw_sub_**

ros::Publisher ***data_pub_**

EtherCAT Output Data Handler header file

Header file for the *EthercatOutputDataHandler* class.

class EthercatOutputDataHandler
#include <ethercat_output_data_handler.h> The Ethercat Output Data Handler class.

Used for trasforming the "raw" indexed data from the */ethercat_data_raw* topic, sent by the Ethercat Communicator, to values of variables, and stream them to the */ethercat_data_out* topic.

Public Functions

void **init** (ros::NodeHandle &*n*)
Initialization Method.

Used for initializing the *EthercatInputDataHandler* object. It's basically the main method in the class, which initializes the listener to the afore mentioned topic.

Parameters

- `n`: The ROS Node Handle

void **raw_data_callback**(**const** ighm_ros::EthercatRawData::ConstPtr &*ethercat_data_raw*)
Raw Data Callback.

This method, is called when there are data in the */ethercat_data_raw* topic. Should the EtherCAT application change, this callback must change also. Implements the basic functionality of the class, to transform the "raw" data into variable values and pipe them into another topic.

Parameters

- `ethercat_data_raw`: A copy of the actual data sent to the topic */ethercat_data_raw*.

Private Members

ros::Subscriber **data_raw_sub_**

ros::Publisher **output_data_pub_**

EtherCAT Slave header file

Header file for the *EthercatSlave* class.

class EthercatSlave
#include <ethercat_slave.h> The Ethercat Slave class.

Used for having all the ethercat slave related variables, fetched from the correspondent yaml file, in a single entity.

Public Functions

void **init**(std::string *slave*, ros::NodeHandle &*n*)
Initialization Method.

Used for initializing the *EthercatSlave* entity. It's basically the main method in the class.

int **get_pdo_out**()
Getter Method.

Used for getting the number of bytes of the output PDO of the single slave.

int **get_pdo_in**()
Getter Method.

Used for getting the number of bytes of the input PDO of the single slave.

Private Members

int **vendor_id_**

std::string **slave_id_**

int **product_code_**

int **assign_activate_**

int **position_**

int **alias_**

int **input_port_**

int **output_port_**

ec_slave_config_t ***ighm_slave_**

int **pdo_in_**

int **pdo_out_**

Main header file

Main header file.

Typedefs

typedef struct *slave_struct* **slave_struct**
A shorthand definition of *slave_struct*.

Variables

slave_struct ***ethercat_slaves**
The main slave struct.

Used by our program to contain all the useful info of every slave.

uint8_t ***domain1_pd**
Global buffer for the actual communication with the IgH Master Module.

uint8_t ***process_data_buf**
Global buffer for safe concurrent accesses from the output PDOs services and the EtherCAT Communicator.

See *ethercat_comm*

size_t **total_process_data**
Total number of process data (PD) (bytes).

size_t **num_process_data_in**
Number of input PD per slave (bytes).

Assumes that the EtherCAT application is the same for every slave.

size_t **num_process_data_out**
Number of output PD per slave (bytes).

Assumes that the EtherCAT application is the same for every slave.

int **log_fd**
File descriptor used for logging, provided that measure_timing is enabled.

Could be deprecated in a next version (see kernelshark).

ec_master_t ***master**
The main master struct.

Used for communication with the IgH Master Module.

ec_master_state_t **master_state**
The master state struct.

Used to examine the current state (Links Up/Down, AL states) of the Master.

ec_master_info_t **master_info**
The master info struct.

Used to know the slaves responding to the Master.

ec_domain_t ***domain1**
The main domain struct variable.

Used to send and receive the datagrams.

ec_domain_state_t **domain1_state**
The domain state struct.

Used to examine the current state (Working counter, DL states) of the domain.

See *ethercat_comm*

pthread_spinlock_t **lock**
The shared spinlock.

Used by every thread whick modifies the process_data_buf.

See *process_data_buf*

EthercatCommunicator **ethercat_comm**
The barebone object of our application.

Used for realtime communication (Tx/Rx) with the EtherCAT slaves. Doesn't change the output PDOs. Basic state machine:

- Receive the new PDOs in domain1_pd from the IgH Master Module (and then to EtherCAT slaves)
- Move to the domain_pd the output data of process_data_buf, safely
- Publish the "raw" data (not linked to EtherCAT variables) in PDOs from the domain1_pd
- Send the new PDOs from domain1_pd to the IgH Master Module (and then to EtherCAT slaves)

EthercatInputDataHandler **ethercat_input_data_handler**
Main object for publishing to the /ethercat_data_slave_x the values of the EtherCAT input variables.

Maps indeces to variables.

EthercatOutputDataHandler **ethercat_output_data_handler**
Main object for publishing to the /ethercat_data_out the values of the EtherCAT output variables.

Maps indeces to variables.

int **PERIOD_NS**
Handy variable induced from the Frequency variable.

See *FREQUENCY*

int **FREQUENCY**
Frequency of the realtime thread: EtherCAT Communicator.

int **RUN_TIME**
Total run time of the realtime thread: EtherCAT Communicator.

struct slave_struct
#include <ighm_ros.h> The basic slave struct.

Public Members

slave_struct::slave_name
The slave's name (for convienience, the names are the positions of the legs)

slave_struct::id
The slave's id (in a four legged robot, that will be from 0 to 3)

slave_struct::slave
The *EthercatSlave* object, used to store every other useful information.

Services header file

Services header file.

Includes:

- EtherCAT Communicator Daemon
- Services for modifying output PDOs

Functions

bool **modify_output_bit**(ighm_ros::ModifyOutputBit::Request &*req*, ighm_ros::ModifyOutputBit::Response &*res*)
ROS Service Callback.

Sets the value of a bit in an index of the *process_data_buf*

bool **modify_output_sbyte**(ighm_ros::ModifyOutputBit::Request &*req*, ighm_ros::ModifyOutputBit::Response &*res*)
ROS Service Callback.

Sets the value of a signed byte in an index of the *process_data_buf*

bool **modify_output_uint16**(ighm_ros::ModifyOutputUInt16::Request &*req*, ighm_ros::ModifyOutputUInt16::Response &*res*)
ROS Service Callback.

Sets the value of an unsigned 16-bit integer in an index of the *process_data_buf*

bool **modify_output_sint16**(ighm_ros::ModifyOutputSInt16::Request &*req*, ighm_ros::ModifyOutputSInt16::Response &*res*)
ROS Service Callback.

Sets the value of a signed 16-bit integer in an index of the *process_data_buf*

bool **modify_output_sint32**(ighm_ros::ModifyOutputSInt32::Request &*req*, ighm_ros::ModifyOutputSInt32::Response &*res*)
ROS Service Callback.

Sets the value of a signed 32-bit integer in an index of the *process_data_buf*

bool **ethercat_communicatord**(ighm_ros::EthercatCommd::Request &*req*, ighm_ros::EthercatCommd::Response &*res*)
ROS Service Callback.

Controls the Ethercat Communicator. The basic functionality is:

- Start
- Stop

- Restart (Remember that a Service Callback must always return a boolean.)

bool **start_ethercat_communicator**()
Helper function for the *ethercat_communicatord* callback.

Used from the callback in order to actualy send the start command to the Ethercat Communicator.

bool **stop_ethercat_communicator**()
Helper function for the *ethercat_communicatord* callback.

Used from the callback in order to actualy send the stop command to the Ethercat Communicator.

Utilities header file

Utilities header file.

Includes:

- Functions for processing EtherCAT PDOs
- Function for insisting write to file
- Function for safe ascii to integer conversion
- Function for adding two timespec structs
- Functions for checking domain and master states

Functions

bool **process_input_bit**(uint8_t **data_ptr*, uint8_t *index*, uint8_t *subindex*)
Returns bit indexed with *index*,*subindex* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The byte index in the buffer
- `subindex`: The bit index inside the byte, indexed by *index*

uint8_t **process_input_uint8**(uint8_t **data_ptr*, uint8_t *index*)
Returns unsigned byte indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The unsigned byte index in the buffer

int8_t **process_input_sint8**(uint8_t **data_ptr*, uint8_t *index*)
Returns signed byte indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The signed byte index in the buffer

uint16_t **process_input_uint16**(uint8_t **data_ptr*, uint8_t *index*)
Returns unsigned 16-bit integer indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The unsigned 16-bit integer index in the buffer

int16_t **process_input_sint16** (uint8_t *_data_ptr_, uint8_t _index_)
Returns signed 16-bit integer indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The signed 16-bit integer index in the buffer

int32_t **process_input_sint32** (uint8_t *_data_ptr_, uint8_t _index_)
Returns signed 32-bit integer indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The signed 32-bit integer index in the buffer

ssize_t **insist_write** (int *fd*, **const** char *_buf_, size_t *count*)
Writes to a *file* descriptor, persistently.

The main difference between *insist_write()* and write() from <unistd.h>, is that write() could write less than *count* bytes to the fd, for various reasons (see man 3 write). Therefore the need for a persistent write to the fd was apparent, hence the *insist_write()*.

Parameters

- `fd`: The file descriptor to write to
- `buf`: The buffer to read from
- `count`: Number of bytes to write

int **safe_atoi** (**const** char *_s_, int *_val_)
Converts an ascii sequence (NULL terminated or "escaped") to integer, safely.

Parameters

- `s`: The ascii sequence (string)
- `val`: The value to return

struct timespec **timespec_add** (**struct** timespec *time1*, **struct** timespec *time2*)
Adds the timespec of the *time2* to the *time1* timespec struct.

The sum is returned in the *time1* timespec struct.

Parameters

- `time1`: The first timespec struct.
- `time2`: The second timespec struct, to be added to the first one.

void **check_domain1_state** (void)
Checks the domain1 state variable.

The checks are for the working counter states and values.

void **check_master_state**(void)
Checks the master state variable.

The checks are for AL states and slaves responding to the master.

Deadline Scheduler header file

Deadline scheduler header file.

Should be added to the project in order the SCHED_DEADLINE option could be used. I'm not the author of the header file, nor do I claim any copyrights for it. This file is distributed under the GPLv2 licence. See the licence above to get an idea.

Defines

SCHED_DEADLINE

Functions

int **sched_setattr**(pid_t *pid*, **const struct** *sched_attr* **attr*, unsigned int *flags*)

int **sched_getattr**(pid_t *pid*, **struct** *sched_attr* **attr*, unsigned int *size*, unsigned int *flags*)

struct sched_attr
#include <deadline_scheduler.h>

Public Members

__u32 **size**

__u32 **sched_policy**

__u64 **sched_flags**

__s32 **sched_nice**

__u32 **sched_priority**

__u64 **sched_runtime**

__u64 **sched_deadline**

__u64 **sched_period**

1.1.2 Source Files

EtherCAT Communicator source file

Implementation of *EthercatCommunicator* class.

Used for real-time communication with the EtherCAT slaves, via the IgH Master module. The new PD are sent to the */ethercat_data_raw* topic.

EtherCAT Input Data Handler source file

Implementation of *EthercatInputDataHandler* class.

Used for handling the "raw" input data, received from EtherCAT Communicator and transforming them into useful, human-readable format, consisted of the EtherCAT variables used by our application. Transforms the indeces to variables.

EtherCAT Output Data Handler source file

Implementation of *EthercatOutputDataHandler* class.

Used for handling the "raw" output data, received from EtherCAT Communicator and transforming them into useful, human-readable format, consisted of the EtherCAT variables used by our application. Transforms the indeces to variables.

EtherCAT Slave source file

Implementation of *EthercatSlave* class.

Used for containing all the useful information of an EtherCAT slave, from the userspace program perspective. Receives all the useful information via the ROS Parameter Server (after they are loaded from ethercat_slaves.yaml).

Main source file

Main source file.

IgH Master EtherCAT module main for realtime communication with EtherCAT slaves. This interface is designed for realtime modules, running in the ROS environment and want to use EtherCAT. There are classes and functions to get the Input and Output PDOs in a realtime context. Before trying to understand the source code of the project, please consider to read and understand the IgH Master Documentation located at: https://www.etherlab.org/download/ethercat/ethercat-1.5.2.pdf. Finally try to understand the API provided in the /opt/etherlab/include/ecrt.h file. The author admits that the C++ language, is not his strong suit. Therefore feel free to refactore the code given with use of the new C++ (11/14/17) helpful tools (unique/shared pointers and other cool stuff).

Changes in version 0.3:

- Created a frontend UI client in Python, for interacting with the Services API. Features include: Start/Stop function of the EtherCAT Communicator Change the Output PDOs on the run Run script with the aproppriate Service API calls

Changes in version 0.2:

- Added features and bug fixes including: First realtime characteristics added (debate: FIFO vs DEADLINE scheduling?) Handling of the EtherCAT communicator module: Start/Stop/Restart API
- Added processing for the /ethercat_data_raw topic and created: Service API for Output PDO handling and topic streaming: /ethercat_output_data Service API for Input PDO handling and topic streaming: /ethercat_data_slave_{slave_id}
- Added synchronization primitives (spinlocks) for the concurrent threads accessing the EtherCAT buffer.
- Added more source files, ethercat communicator, ethercat_slave, ethercat_input_data_handler and ethercat_output_data_handler. Created external objects to use the appropriate classes and functions.

Version 0.1:

- Created the first bare communication layer in the ROS environment. Many bugs and deficiencies including: Non realtime characteristics, no API for Output PDO handling and topic streaming, no handling of the EtherCAT communicator module, no Service API for Input PDO topic streaming.
- One topic streaming: /ethercat_data_raw

Unnamed Group

uint8_t ***domain1_pd**
Global buffer for the actual communication with the IgH Master Module.

uint8_t ***process_data_buf**
Global buffer for safe concurrent accesses from the output PDOs services and the EtherCAT Communicator.

See *ethercat_comm*

size_t **total_process_data**
Total number of process data (PD) (bytes).

size_t **num_process_data_in**
Number of input PD per slave (bytes).

Assumes that the EtherCAT application is the same for every slave.

size_t **num_process_data_out**
Number of output PD per slave (bytes).

Assumes that the EtherCAT application is the same for every slave.

int **log_fd**
File descriptor used for logging, provided that measure_timing is enabled.

Could be deprecated in a next version (see kernelshark).

ec_master_t ***master**
The main master struct.

Used for communication with the IgH Master Module.

ec_master_state_t **master_state**
The master state struct.

Used to examine the current state (Links Up/Down, AL states) of the Master.

ec_master_info_t **master_info**
The master info struct.

Used to know the slaves responding to the Master.

ec_domain_t ***domain1**
The main domain struct variable.

Used to send and receive the datagrams.

ec_domain_state_t **domain1_state**
The domain state struct.

Used to examine the current state (Working counter, DL states) of the domain.

See *ethercat_comm*

slave_struct ***ethercat_slaves**
The main slave struct.

Used by our program to contain all the useful info of every slave.

pthread_spinlock_t **lock**
The shared spinlock.

Used by every thread whick modifies the process_data_buf.

See *process_data_buf*

EthercatCommunicator **ethercat_comm**
The barebone object of our application.

Used for realtime communication (Tx/Rx) with the EtherCAT slaves. Doesn't change the output PDOs. Basic state machine:

- Receive the new PDOs in domain1_pd from the IgH Master Module (and then to EtherCAT slaves)
- Move to the domain_pd the output data of process_data_buf, safely
- Publish the "raw" data (not linked to EtherCAT variables) in PDOs from the domain1_pd
- Send the new PDOs from domain1_pd to the IgH Master Module (and then to EtherCAT slaves)

EthercatInputDataHandler **ethercat_input_data_handler**
Main object for publishing to the /ethercat_data_slave_x the values of the EtherCAT input variables.

Maps indeces to variables.

EthercatOutputDataHandler **ethercat_output_data_handler**
Main object for publishing to the /ethercat_data_out the values of the EtherCAT output variables.

Maps indeces to variables.

int **FREQUENCY**
Frequency of the realtime thread: EtherCAT Communicator.

int **RUN_TIME**
Total run time of the realtime thread: EtherCAT Communicator.

int **PERIOD_NS**
Handy variable induced from the Frequency variable.

See *FREQUENCY*

int **main** (int *argc*, char ***argv*)

Services source file

Implements the services used.

Provides services for:

- Interacting with the EtherCAT Communicator
- Changing the EtherCAT output PDOs

Functions

std::string &**ltrim**(std::string &*str*, **const** std::string &*chars* = "tnvfr ")
Left trims a string.

This function trims any character specified in *chars*, which is left of the input *str*.

Parameters

- `str`: The input untrimmed string.
- `chars`: The characters to trim.

std::string &**rtrim**(std::string &*str*, **const** std::string &*chars* = "tnvfr ")
Right trims a string.

This function trims any character specified in *chars*, which is right of the input *str*.

Parameters

- `str`: The input untrimmed string.
- `chars`: The characters to trim.

std::string &**trim**(std::string &*str*, **const** std::string &*chars* = "tnvfr ")
Trims a string from the left and right.

This function trims any character specified in *chars*, which is right or left of the input *str*. Calls internally the *ltrim* and *rtrim* functions. See more at: http://www.martinbroadhurst.com/how-to-trim-a-stdstring.html

See *ltrim*

See *rtrim*

bool **modify_output_bit**(ighm_ros::ModifyOutputBit::Request &*req*, ighm_ros::ModifyOutputBit::Response &*res*)
ROS Service Callback.

Sets the value of a bit in an index of the *process_data_buf*

bool **modify_output_sbyte**(ighm_ros::ModifyOutputBit::Request &*req*, ighm_ros::ModifyOutputBit::Response &*res*)
ROS Service Callback.

Sets the value of a signed byte in an index of the *process_data_buf*

bool **modify_output_uint16**(ighm_ros::ModifyOutputUInt16::Request &*req*, ighm_ros::ModifyOutputUInt16::Response &*res*)
ROS Service Callback.

Sets the value of an unsigned 16-bit integer in an index of the *process_data_buf*

bool **modify_output_sint16**(ighm_ros::ModifyOutputSInt16::Request &*req*, ighm_ros::ModifyOutputSInt16::Response &*res*)
ROS Service Callback.

Sets the value of a signed 16-bit integer in an index of the *process_data_buf*

bool **modify_output_sint32**(ighm_ros::ModifyOutputSInt32::Request &*req*, ighm_ros::ModifyOutputSInt32::Response &*res*)
ROS Service Callback.

Sets the value of a signed 32-bit integer in an index of the *process_data_buf*

ethercat_communicatord(ighm_ros::EthercatCommd::Request &*req*, ighm_ros::EthercatCommd::Response &*res*)
ROS Service Callback.

Controls the Ethercat Communicator. The basic functionality is:

- Start
- Stop
- Restart (Remember that a Service Callback must always return a boolean.)

start_ethercat_communicator()
Helper function for the *ethercat_communicatord* callback.

Used from the callback in order to actualy send the start command to the Ethercat Communicator.

stop_ethercat_communicator()
Helper function for the *ethercat_communicatord* callback.

Used from the callback in order to actualy send the stop command to the Ethercat Communicator.

Utilities source file

A library with useful functions for handling EtherCAT PDOs and other utilities.

Functions

int **safe_atoi**(**const** char **s*, int **val*)
Converts an ascii sequence (NULL terminated or "escaped") to integer, safely.

Parameters

- `s`: The ascii sequence (string)
- `val`: The value to return

bool **process_input_bit**(uint8_t **data_ptr*, uint8_t *index*, uint8_t *subindex*)
Returns bit indexed with *index*,*subindex* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The byte index in the buffer
- `subindex`: The bit index inside the byte, indexed by *index*

int8_t **process_input_sint8**(uint8_t **data_ptr*, uint8_t *index*)
Returns signed byte indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The signed byte index in the buffer

uint8_t **process_input_uint8**(uint8_t **data_ptr*, uint8_t *index*)
Returns unsigned byte indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The unsigned byte index in the buffer

uint16_t **process_input_uint16** (uint8_t **data_ptr*, uint8_t *index*)
Returns unsigned 16-bit integer indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The unsigned 16-bit integer index in the buffer

int16_t **process_input_sint16** (uint8_t **data_ptr*, uint8_t *index*)
Returns signed 16-bit integer indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The signed 16-bit integer index in the buffer

int32_t **process_input_sint32** (uint8_t **data_ptr*, uint8_t *index*)
Returns signed 32-bit integer indexed with *index* of the *data_ptr* buffer.

Parameters

- `data_ptr`: The buffer to get the data
- `index`: The signed 32-bit integer index in the buffer

struct timespec **timespec_add** (**struct** timespec *time1*, **struct** timespec *time2*)
Adds the timespec of the *time2* to the *time1* timespec struct.

The sum is returned in the *time1* timespec struct.

Parameters

- `time1`: The first timespec struct.
- `time2`: The second timespec struct, to be added to the first one.

void **check_domain1_state** (void)
Checks the domain1 state variable.

The checks are for the working counter states and values.

check_master_state (void)
Checks the master state variable.

The checks are for AL states and slaves responding to the master.

ssize_t **insist_write** (int *fd*, **const** char **buf*, size_t *count*)
Writes to a *file* descriptor, persistently.

The main difference between *insist_write()* and write() from <unistd.h>, is that write() could write less than *count* bytes to the fd, for various reasons (see man 3 write). Therefore the need for a persistent write to the fd was apparent, hence the *insist_write()*.

Parameters

- `fd`: The file descriptor to write to
- `buf`: The buffer to read from
- `count`: Number of bytes to write

EtherCAT Keyboard Controller python file

namespace ethercat_keyboard_controller

Variables

################################## ## Ethercat Keyboard controller ## ################################## Changes elliptic trajectory parameters and controlls the ethercat communicator. All the terminal commands must beggin with an exclamation mark "!". If you want to find the current application variables, see the EthercatOutputData.msg. The current supported commands are: !start : starts the ethercat communicator !stop : stops the ethercat communicator !restart : restarts the ethercat communicator !variable [slave_id | 'all'] [variable_name] [value] : change the value of a variable in the ethercat output data !run [script_to_run] : run the script specified, inside the ighm_ros/scripts directory !help : shows this help message !q : exit the terminal Type !q to quit """]

Welcome to the Ethercat Keyboard Controller! By Mike Karamousadakis Contact: mkaramousadakis@zoho.eu """]

prompt = *ethercat_controller*()

intro

class ethercat_controller

Base Ethercat Controller class.

Inherits from the base class Cmd. Serves as a frontend to the C++ code, and specifically to the services implemented.

Public Functions

call_modify_service(*self self*, *slave_id slave_id*, *variable_name variable_name*, *value value*)

modify_service wrapper.

Used for creating the arguments for calling the *Modify Services*.

Parameters

- `self`: The current object.
- `slave_id`: The slave's index.
- `variable_name`: The variable's name to modify.
- `value`: The value to set.

call_service_mux(*self self*, *name name*, *data data*)

call service mux.

Used for muxing the calls to the *Modify Services*.

Parameters

- `self`: The current object.
- `name`: The function's name
- `data`: The arguments for the function

modify_output_bit_client(*self self*, *slave_id slave_id*, *index index*, *subindex subindex*, *value value*)
Frontend client for the modify_output_bit service.

Parameters
- `self`: The current object.
- `slave_id`: The slave's index.
- `index`: The index to change inside the buffer.
- `subindex`: The subindex inside the index.
- `value`: The value to set.

modify_output_sbyte_client(*self self*, *slave_id slave_id*, *index index*, *value value*)
Frontend client for the modify_output_sbyte service.

Parameters
- `self`: The current object.
- `slave_id`: The slave's index.
- `index`: The index to change inside the buffer.
- `value`: The value to set.

modify_output_sint16_client(*self self*, *slave_id slave_id*, *index index*, *value value*)
Frontend client for the modify_output_sint16 service.

Parameters
- `self`: The current object.
- `slave_id`: The slave's index.
- `index`: The index to change inside the buffer.
- `value`: The value to set.

modify_output_uint16_client(*self self*, *slave_id slave_id*, *index index*, *value value*)
Frontend client for the modify_output_uint16 service.

Parameters
- `self`: The current object.
- `slave_id`: The slave's index.
- `index`: The index to change inside the buffer.
- `value`: The value to set.

modify_output_sint32_client(*self self*, *slave_id slave_id*, *index index*, *value value*)
Frontend client for the modify_output_sint32 service.

Parameters
- `self`: The current object.
- `slave_id`: The slave's index.
- `index`: The index to change inside the buffer.
- `value`: The value to set.

ethercat_communicator_client(*self self*, *mode mode*)
Frontend client for the ethercat_communicatord service.

Parameters
- `self`: The current object.
- `mode`: The mode to change to (start/stop/restart).

do_shell(*self self*, *args args*)
Main method of *ethercat_controller* class.

Accepts commnads, decomposes them and acts.
Parameters
- `self`: The current object.
- `args`: The arguments given from the cmd line.

do_help (*self self*, *line line*)
Helper method.

Parameters
- `self`: The current object.
- `line`: Unrelated argument.Just follow the API

default (*self self*, *line line*)
Unrecognized command method.

Parameters
- `self`: The current object.
- `line`: Unrelated argument.Just follow the API

Public Static Attributes

"state_machine" : [[0, 0], "bit"], "initialize_clock": [[0, 1], "bit"], "initialize_angles": [[0, 2], "bit"], "inverse_kinematics": [[0, 3], "bit"], "blue_led": [[0, 4], "bit"], "red_led": [[0, 5], "bit"], "button_1": [[0, 6], "bit"], "button_2": [[0, 7], "bit"], "sync": [[1], "sbyte"], "desired_x_value": [[2], "sint32"], "filter_bandwidth": [[6], "uint16"], "desired_y_value": [[8], "sint32"], "kp_100_knee": [[12], "sint16"], "kd_1000_knee": [[14], "sint16"], "ki_100_knee": [[16], "sint16"], "kp_100_hip": [[18], "sint16"], "kd_1000_hip": [[20], "sint16"], "ki_100_hip": [[22], "sint16"], "x_cntr_traj1000": [[24], "sint16"], "y_cntr_traj1000": [[26], "sint16"], "a_ellipse100": [[28], "sint16"], "b_ellipse100": [[30], "sint16"], "traj_freq100": [[32], "sint16"], "phase_deg": [[34], "sint16"], "flatness_param100": [[36], "sint16"] }] variables2indeces dictionary. Used for transforming the [index,subindex] -> variable

"sint16": modify_output_sint16_client, "uint16": modify_output_uint16_client, "sint32": modify_output_sint32_client, "sbyte" : modify_output_sbyte_client, "ethercat_communicator": ethercat_communicator_client}] Function dictionary. It's keys are the names of the methods, and their values are the references to the actual methods.

1.2 License

1.2.1 Note

The license mentioned above concerns the source code only. Using the EtherCAT technology and brand is only permitted in compliance with the industrial property and similar rights of Beckhoff Automation GmbH. Contact information: mkaramousadakis@zoho.eu

1.3 Guide

This guide will solve your problem of where to start with documentation, by providing a basic explanation of how to do it easily. Look how easy it is to use:

- After you *catkin_make* the project, in one terminal run:

$ roslaunch ighm_ros ighm_ros.launch

- After that, and while the process is running, you run in another terminal:

$ rosrun ighm_ros ethercat_keyboard_controller.py

- Now you can give orders to the EtherCAT Communicator via a custom terminal.

Have fun playing around!

- *Tip: You could run a bash script in the custom terminal by running:*

[ethercat_controller] > !r my_awesome_bash_script.sh

Notice that your script must be under the *scripts* directory. You could also check some example scripts there.

1.3.1 Features

- Real time characteristics, using PREEMPT_RT patch
- EtherCAT technology adaptation in Linux
- Works in a recent version of ROS (kinetic)
- Utilizes the main development framework for EtherCAT applications, IgH Master kernel module

1.3.2 Installation

SUPER SOS : The following manual will be extremely helpful for understanding the instructions given in this section. You should **definitely** read it before procceding. Link.

0. Preempt_RT Patch

First step to utilize the repository given, is to install the preempt_rt patch in the kernel. Note that in order to install the IgHM, the kernel should be up to 4.9. A proper guide for the installation procedure can be found in the following link. After you patch the kernel (say 4.9), you will want to use some other configuration parameters in the build of the kernel. Therefore after step 3 and in the line of *make menuconfig* of the previous link, you will want to specify some extra configuration parameters, derived from the chapter 3 of the manual mentioned in the beginning, namely:

- CONFIG_PREEMPT_RT_FULL
- CONFIG_CPU_FREQ=n
- CONFIG_CPU_IDLE=n
- CONFIG_NO_HZ_FULL=y
- CONFIG_RCU_NOCB_CPU=y

The author has used only the first configuration parameter and has seen great boost in the performance of the real-time tasks specified. Future work requires more tweaks to be done, derived from the afore mentioned manual.

Tip: While you are in the menuconfig, type "/" and you can type to find the place of a configuration parameter. Exit with ESC

1. Installation of IgHM

1. cd into the etherlab-mercurial folder
2. Run *make ethercatMasterInstallWithAutoStart*. Note that root access is needed. If you use another native driver from e1000e, open the Makefile and change the configure option with your driver version. You can find here and there (Chapter 9), the supported hardware and the options the command *configure* takes, respectively. If your hardware is not supported or if you don't want the native driver support fuzz, then you should change the *configure* command to enable generic driver support (although I think it defaults to that).

2. Run the scripts

1. Because we want to have a process in realtime context, we should change it's priority (done in the code -FIFO policy, 80 priority-). Besides that, the interrupt handler which handles the interrupts generated by the network driver, should have higher priority than the process we develop, so that the EtherCAT datagrams are ready to be sent/received before we process them. For that cause I have written a script, as a sample script, to change the priority of the irq process of the network card.This should be used accordingly to change **your** process's priority. You could check if the priority has changed with the *chrt* command. How-to can be found in this link.
2. Aside from the enhancements proposed by the manual, we should also change the throttling of our network driver to 0. This is done in the script also in the *testbench* directory. It is based on my e1000e driver, so use it as a sample script. Documentation for the insertion of the module of the e1000e network driver can be found in here.
3. Run the script for changing the permissions of ighm_ros. We set the suid of ighm_ros to be root, so that the ighm_ros can be launched without *sudo*. This will be useful **after** you *catkin_make* the project.

1.3.3 Contribute

- Issue Tracker: https://github.com/mikekaram/IgHMaster-userspace-program-in-ROS/issues
- Source Code: https://github.com/mikekaram/IgHMaster-userspace-program-in-ROS

1.3.4 Limitations / Steps Forward

This program assumes that the actual control code of the robot is running in the EtherCAT slaves. Therefore there is no connection between this program and ros_control, although the intention of the author is to make this connection happen, for robots that do have a control api inside ROS. This of course means that the ros_control module should communicate afterwards with this program, to send new data to the EtherCAT slaves. Needless to say, the EtherCAT slaves will have a much more passive role in this configuration.

1.3.5 Support

If you are having issues, please let us know. We don't have a mailing list yet, so the default way is by communicating with: mkaramousadakis@zoho.eu

1.3.6 License

The project is licensed under the GPLv2 licence. See more details in the source files of the project or in the Lincence section.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

F

I

L

M

N

P

R

S

T

V