Implementation of emProtocol

This document describes the implementation of emProtocol under embOBJ environment, for use on the EMS boards and on the PC104 host.

Approval History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Version | Author | | Date | Approved | | Date |
| 1.0 | Accame | iCub Unit |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 24 Jul 13 | M. Accame | First edit of the document |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 The implementation of emProtocol 1](#_Toc362965407)

[2 The communication between the host and the board 2](#_Toc362965408)

[2.1 The object EOtheBOARDtransceiver 3](#_Toc362965409)

[2.1.1 The object EOnvSet 3](#_Toc362965410)

[2.1.2 The object EOtransceiver 7](#_Toc362965411)

[2.2 The object EOhostTransceiver 9](#_Toc362965412)

[2.2.1 The object EOnvSet 9](#_Toc362965413)

[2.2.2 The object EOtransceiver 10](#_Toc362965414)

[2.2.3 Example of use of the EOconfirmationManager 12](#_Toc362965415)

[3 Most common operations 13](#_Toc362965416)

[3.1 Operations on the host 13](#_Toc362965417)

[3.1.1 Add an occasional ROP set<> to the transceiver 13](#_Toc362965418)

[3.1.2 Send a ROPframe 14](#_Toc362965419)

[3.1.3 Add an occasional ROP ask<> to the transceiver 14](#_Toc362965420)

[3.1.4 Manage the reception of a ROP say<> 15](#_Toc362965421)

[3.1.5 Managing a ask<> and say<> 16](#_Toc362965422)

[3.1.6 Have access of a given variable from its ID 17](#_Toc362965423)

[3.1.7 Build IDs 18](#_Toc362965424)

[3.2 Operations on the board 19](#_Toc362965425)

[3.2.1 Build a ROP say<> 19](#_Toc362965426)

[3.2.2 Manage the reception of a ROP set<> 19](#_Toc362965427)

[3.2.3 Have access of a given variable from inside the callback function 19](#_Toc362965428)

[3.2.4 Have access of a given variable from its ID 19](#_Toc362965429)

[3.2.5 Configure the regular transmission of a ROP 19](#_Toc362965430)

[4 Configuration of a board 20](#_Toc362965431)

[4.1 The eOprot\_Bxx files 20](#_Toc362965432)

[5 Change of an endpoint 21](#_Toc362965433)

[5.1 Adding a tag on an existing entity 21](#_Toc362965434)

[5.2 Adding an entire entity 21](#_Toc362965435)

[5.3 Adding a complete endpoint 21](#_Toc362965436)

The implementation of emProtocol

This document describes the implementation of emProtocol under embOBJ environment for use both on the EMS boards and PC104 host.

In short, the implemented mechanism allows exchanging ROPframes containing ROPs related to a pre-loaded set of network variables. The ROPs are automatically managed by the above mechanism which calls user-defined functions when the RAM of a given network variable is changed.

At first it is described which are the objects which are used to exchange ROPframes and how the ROPframes are decoded and actions related to the contained ROPs are executed. Also, it is shown how the objects can be configured to “know about” a given set of network variables.

It is also shown which methods are available to the user to perform simple operations such as: build a ROP, send / receive a ROPframe, personalise actions on reception of ROPs, have access to the RAM of a given network variable.

Later it is described how it is possible to operate changes in the set of network variables, so that a user can add or remove network variables.

The communication between the host and the board

The ROPframe exchange between the host (PC104) and the board (EMS) is done using two specialised objects, the EOhostTransceiver and the EOtheBOARDtransceiver, which resides respectively on the PC104 and the EMS. The board object is a singleton as there must be only instance of it, whereas the host object can be instantiated many times.

On the PC104 there are many instances of EOhostTransceiver, each of them paired with the EOtheBOARDtransceiver of a given EMS.

ROPFRAME EXCHANGE

PC104

eOnvset\_DEVcfg\_t nvsB1

EOhostTransceiver

eOnvset\_DEVcfg\_t nvsB2

EOhostTransceiver

BOARD 1

eOnvset\_DEVcfg\_t nvsB1

EOtheBOARDtransceiver

BOARD 2

eOnvset\_DEVcfg\_t nvsB2

EOtheBOARDtransceiver

EOropframe

EOropframe

EOropframe

EOropframe

**Figure 1**: The EOropframe content is exchanged through the network between pairs of transceivers of kind EOhostTransceiver on the PC104 and EOtheBOARDtransceiver on the EMS board. Both transceivers are initialised with the same configuration of network variables.

Both objects contain one EOtransceiver and one EOnvSet. The EOtransceiver is responsible of managing ROPs and ROPframes but it uses the EOnvSet to perform operations on the network variables. The EOnvSet contains knowledge of all the network variables of a given board and also contains the RAM where the ROPs operate. The difference between the EOtheBOARDtransceiver and the EOhostTransceiver is that the former directly owns the variables inside the EOnvSet, whereas the latter uses the EOnvSet to form ROPs which sets values on the board and to cache inside its RAM the values that the board sends.

As an example, when the host wants to set the value of the PID of a joint on the associated board, the EOhostTransceiver queries its EOnvSet about presence of that variable on the remote board, then the EOtransceiver builds the ROP and fills it inside a ROPframe, so that the sending socket can retrieve it for transmission.

When the board receives the UDP packet it gives that to the EOtheBOARDtransceiver, where the ROPs are extracted by the internal EOtransceiver. The ROP is processed with the aid of the EOnvSet. Only if the ID inside the ROP is recognised, then the ROP is processed. In the case of a set<> the data field of the ROP is copied inside the RAM owned by the EOnvSet and a callback is executed.

On the other hand if the host wants to verify the values of the PID of a joint on the associated board, the EOhostTransceiver builds the ask<> ROP and sends it to the board. The board processes the ROP and form a reply with a say<> ROP which contains data copied from the RAM owned by its EOnvSet. Finally the EOhostTransceiver receives the say<> ROP and copies its data inside the RAM owned by the EOnvSet and executes a callback.

The above mechanism is executed as long as both EOhostTransceiver and the EOtheBOARDtransceiver have instances of EOnvSet configured with the same eOnvset\_DEVcfg\_t data.

## The object EOtheBOARDtransceiver

The object EOtheBOARDtransceiver is a singleton which manages ROP-based communication with a set of variables which are locally owned. As such, it manages incoming ROPs of type ask<> and set<> and sends out ROPs of type say<> and sig<>.

Whenever it receives a set<> ROP it: (a) copies the received data inside the variable identified by the ID, and (b) calls a user-defined update function which alerts the application about the change of value.

When it receives ask<> ROP it: (a) prepares a say<> ROP with the content of the queried variable.

Finally, it can send spontaneous sig<> ROPs to inform the host about the value of a variable.

An interesting feature of the EOtheBOARDtransceiver is that it can be configured (via a ROP obviously!) to emit sig<> ROPs on a regular basis.

STRUCTURE OF EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 2**: The EOtheBOARDtransceiver is a singleton on the EMS board. It contains one EOtransceiver and one EOnvSet object. The EOtransceiver is composed of one EOtransmitter and one EOreceiver which processes EOropframe objects. The EOnvSet contains knowledge of all the network variables of the specific board. It is also responsible of allocating the RAM for the variables.

### The object EOnvSet

The EOnvSet is responsible of loading the configuration of a given board which personalises the behaviour of EOtheBOARDtransceiver. The configuration is contained in a variable of type eOnvset\_DEVcfg\_t.

EOnvSet in EOtheBOARDtransceiver

BOARD 1

eOnvset\_DEVcfg\_t nvsB1

EOtheBOARDtransceiver

EOnvSet

HEAP

RAM of netvars

Contains information about every endpoint present on that board, on how many entities are contained (joints, motors, etc), about the size of RAM required, etc.

ROM

Endpoints, num of entities, user-def functions, etc.

Protocol library

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 3**: The EOnvSet object is initialised using a constant structure which contains information on the network variables of the board. For instance, it contains information on the managed endpoints and on how many entities are in any endpoint, so that the EOnvSet can allocate the RAM used for the network variables and initialises other data structures.

EOnvSet in EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

eOnvset\_DEVcfg\_t nvsB1

1. Loads every endpoint in EOnvSet with the number of entities specified for that board.
2. Allocates the RAM for the variables in the endpoints.
3. Initializes the protocol library passing to it the relevant RAM and the number of entities of each endpoint of the board.
4. Calls the user-defined init() function of every network variable

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 4**: In its initialisation, the EOnvSet performs a series of operations such as: load the endpoints specified in the eOnvset\_DEV\_cfg\_t board configuration, allocate the required RAM, initialise the protocol library by loading into it the number of entities and the RAM for each endpoint, calling the user-defined init() functions for every network variable.

Protocol library used by EOnvSet inside EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

HEAP

Protocol library

ROM

EP

* loads RAM,
* loads number of entities,
* already contains for each variable the data structure used by EOnv

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 5**: The protocol library is configured by the EOnvSet for each board and each endpoint the board supports. In particular, for each endpoint of a given board the protocol library is assigned the RAM containing all the variables and pointers to ROM data containing the number of entities. The protocol library already contains internal reference to all the data structure required for each variable. For instance if a board supports the motion-control endpoint with 4 joints, 4 motors and 1 controller, then the initialisation consists of passing a const array with [4, 4, 1], and a pointer to N bytes of RAM.

Main services of protocol library

BOARD 1

EOtheBOARDtransceiver

EOnvSet

eOprotID32\_t ID = eoprot\_ID\_get(EP, ENT, IND, TAG);

eObool\_t res = eoprot\_id\_isvalid(eoprot\_board\_localboard, ID);

void \*var = eoprot\_variable\_ramof\_get(eoprot\_board\_localboard, ID);

HEAP

RAM of netvars

Protocol library

ROM

Endpoints, num of entities, user-def functions, etc.

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 6**: After the protocol library is configured by the EOnvSet, it is able to offer services such as: verifying if an ID of a variable is supported by the board, retrieving the RAM and ROM information relative to an ID, convert an ID into (EP, ENT, IND, TAG) and the other way round.

EOnvSet and EOnv in EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

HEAP

RAM of netvars

EOnv

EOnv nv;

eo\_nvset\_NV\_Get(ID, &nv);

- pointer to RAM of variable

- size of variable

- mutex of the variable

- init() function

- update() function

Protocol library

ROM

Endpoints, num of entities, user-def functions, etc.

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 7**: Inside the EOtheBOARDtransceiver, the EOnvSet object is used to retrieve a given data structure associated to a network variable: the EOnv. The EOnv contains pointer to the RAM of the network variable but also its size and the mutex to be used when manipulating its data and some associate user-defined functions: init() and update(). The EOnv can be used to set / get values into / from the network variable. The EOtransceiver and associated objects use the EOnv to process the ROP.

EOnvSet in EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

void \*var = eo\_nvset\_RAMofVariable\_Get(ID); // RAM of PID of a given joint

void \*ent = eo\_nvset\_RAMofEntity\_Get(EP, entity, index); // RAM of a given joint

HEAP

RAM of netvars

Protocol library

ROM

Endpoints, num of entities, user-def functions, etc.

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 8**: The EOnvSet object also offers methods to retrieve the RAM and the size of a given network variable or of a given entity. These methods offer a more immediate access to data of variables than using a EOnv.

### The object EOtransceiver

The EOtransceiver is responsible of decoding and encoding of EOropframe objects. It uses services of other objects, such as the EOtheParser, the EOtheAgent, and the EOtheFormer.

EOtransceiver in EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

EOtheParser

EOtheFormer

EOtheAgent

EOropframe

EOropframe

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 9**: The EOtransceiver contains the EOreceiver, the EOtransmitter and a NULL handle to the EOconfirmationManager. It uses objects such as the EOtheParser and EOtheAgent to extract and process EOrop objects from the received EOropframe. It uses the EOtheFormer to prepare an EOrop to transmit.

Reception inside EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

EOrop

1. Find the EOnv associated to received ID,
2. If ROP is ask<>: read data from EOnv, prepare a reply EOrop, pass it to EOtransmitter
3. If ROP is set<>: write data inside EOnv, calls associated update() function. Prepare a confirmation if requested and passes it to EOtransmitter

For every EOrop inside the received EOropframe:

EOropframe

HEAP

RAM of netvars

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 10**: The EOreceiver decodes the input EOropframe and for each EOrop in its inside. It can be a set or ask. If it is a set it writes the data of EOrop in its RAM and calls a user-defined function update(). If it is a ROP of kind ask<>, it reads its RAM and prepares a say<> EOrop which it put inside the EOtransmitter.

Transmission inside EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

EOropframe

EOtransmitter

occasionals

regulars

replies

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 11**: The EOtransmitter contains three internal EOropframe: one with regular EOrop to be transmitted every time, one with the EOrop with replies from the EOreceiver, and one with the EOrop which the board wants to sends occasionally. The output EOropframe is the concatenation of the three.

## The object EOhostTransceiver

The object EOhostTransceiver differs from the EOtheBOARDtransceiver in a few aspects.

First, it is not a singleton and thus can be instantiated many times: one for each board. Second, it is configured to hold knowledge of network variables which are remotely owned, thus it transmits ROPs of type ask<> and set<> and manages incoming ROPs of type say<> and sig<>. Third, it can be configured to have a EOconfirmationManager which executes actions on out-coming ROPs with confirmation request flagged on and on reception of ACK or NAK for those ROPs.

### The object EOnvSet

The EOnvSet is initialised by loading a given eOnvset\_DEVcfg\_t data structure for each board. For any of its endpoints is responsible to allocate the RAM and to initialise the protocol library with this RAM and with the number of entities. As a result of initialisation of every EOhostTransceiver, the protocol library shall be initialised with data for every board.

Internals of EOhostTransceiver

BOARD 1

eOnvset\_DEVcfg\_t nvsB1

EOhostTransceiver

EOnvSet

HEAP

RAM of netvars

ROM

Endpoints, num of entities, user-def functions, etc.

Protocol library

BOARD 2

eOnvset\_DEVcfg\_t nvsB2

EOhostTransceiver

EOnvSet

EOtransceiver

EOtransmitter

EOreceiver

BOARD 2

BOARD 1

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

RAM and ROM for every endpoint of that board

**Figure 12**: The EOnvSet object of each instance of EOhostTransceiver is initialised with a proper eOnvset\_DEVcfg\_t data structure, use proper RAM and ROM to initialise the protocol library for its own board.

### The object EOtransceiver

The EOtransceiver contains three main objects: the EOtransmitter, the EOreceiver, and the EOconfirmationManager. The EOtransmitter typically sends ROPs of kind ask<> and set<>, possibly with confirmation request. The EOreceiver processes received ROPs of kind say<> or ask-nak<> if a sent ask<> was not recognised by the EMS, set-ack<> and set-nak<> in response to a sent set<>, and finally sig<> ROPs. The EOconfirmationManager is uses by the EOtransmitter to call a user-defined function when a ROP requests confirmation info and by the EOreceiver to call a user-defined function when the ACK/NAK is received.

Transmission inside EOtheBOARDtransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

EOropframe

EOtransmitter

occasionals

regulars

replies

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 13**: The EOtransmitter contains three internal EOropframe but typically uses only one of them: the one with the EOrop which the board wants to sends occasionally. The other two, one with regular EOrop and one with replies from the EOreceiver are typically empty. The output EOropframe is the concatenation of the three. If one of the occasional ROPs has a confirmation request flagged on, then the on-tx function of the EOconfirmationManager is called.

Reception inside EOhostTransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

EOrop

1. Find the EOnv associated to received ID,
2. If ROP is sig<> or say<>: write data inside EOnv and calls associated update() function.
3. If ROP is ack or nak of a set<> or ask<>: calls the proper function of the object EOconfManager.

For every EOrop inside the received EOropframe:

EOropframe

HEAP

RAM of netvars

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 14**: The EOreceiver decodes the input EOropframe and each EOrop in its inside. It can be a say<> or sig<> or even a ACK/NAK (set-ack<>, set-nak<>, ask-nak<>). If it is a say<> or sig<> it writes the data of EOrop in its RAM and calls a user-defined function update(). If it is a ROP with ACK/NAK information it calls the proper on-rx function of the EOconfirmationManager.

Use of EOconfirmationManager inside EOhostTransceiver

BOARD 1

EOtheBOARDtransceiver

EOnvSet

EOrop

1. Find the EOnv associated to received ID,
2. If ROP is sig<> or say<>: write data inside EOnv and calls associated update() function.
3. If ROP is ack or nak of a set<> or ask<>: calls the proper function of the object EOconfManager.

For every EOrop inside the received EOropframe:

EOropframe

HEAP

RAM of netvars

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 15**: The function eo\_confman\_Confirmation\_Requested() is called by the EOtransmitter when the user loads an occasional ROP which requires confirmation. The function eo\_confman\_Confirmation\_Received() is called when the EOreceiver processes an incoming ROP which is a ACK/NAK.

### Example of use of the EOconfirmationManager

The functions eo\_confman\_Confirmation\_Requested() and eo\_confman\_Confirmation\_Received() are called by the by the EOtransmitter when the user loads an occasional ROP which requires confirmation and when the EOreceiver processes an incoming ROP which is a ACK/NAK.

The default implementation of the EOconfirmationManager contains empty functions which can be specialised at creation of the object by passing user-defined callback functions. If the user writes them in a proper way he/she can effectively manage the verification of delivery of a ROP. An example is the following.

The eo\_confman\_Confirmation\_Requested() calls a user-defined function which puts the transmitted ROP inside a list and starts a countdown timer. The countdown timer is used to do an action if after a timeout no ACK/NAK is received. The function eo\_confman\_Confirmation\_Received() is called when the EOreceiver processes the ACK/NAK of a previously transmitted ROP. This method calls call a user-defined function which searches for the sent ROP inside the list and stops the countdown timer. If it the timer expires before having received a ACK/NAK, then the user may issue a warning of ROP not delivered or organise a new transmission.

Most common operations

Her are described the most common operations on the PC104 and on the EMS.

On the PC104 they are: to build a ROP to be sent, to have access to the values of the network variables being received from the EMS, to override the update callback functions so that they can be used to trigger actions on reception of ROPs.

On the EMS they are: to build a ROP to be sent, to configure the transmission of a regular ROP, to have access to the values of the network variables belonging to the EMS, to override the update callback functions so that they can be used to trigger actions on reception of ROPs.

It will be shown an example for the case of motion-control.

## Operations on the host

### Initialisation

For each remote board it is created an object EOhostTransceiver with a given configuration. There are methods to retrieve the IP address of the remote board, or its number, the EOtransceiver or the EOnvSet inside the EOhostTransceiver.

Amongst configuration modes, there is the possibility to provide protection vs concurrent use of the methods of the object EOtransceiver and of EOnvSet.

Initialisation

Thread INIT

// we must create a host transceiver for a given board

// and having access to its internal transceiver so that use can sue its methods

#define BRD1 0

const eOhosttransceiver\_cfg\_t\* brd1cfg = getit(BRD1);

EOhostTransceiver\* hosttrans1 = eo\_hosttransceiver\_New(brd1cfg);

EOtransceiver\* txrx1 = eo\_hosttransceiver\_GetTransceiver(hosttrans1);

EOnvSet\* nvset1 eo\_hosttransceiver\_GetNVset(hosttrans1);

eOnvBRD\_t brd1 = eo\_hosttransceiver\_GetBoardNumber(hosttrans1);

eOipv4addr\_t ip1 = eo\_hosttransceiver\_GetRemoteIP (hosttrans1);

EOhostTransceiver

EOnvSet

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

**Figure 15**: Initialisation.

### Regular use

There are three main activities: (a) processing a received EOropframe, (b) adding occasional ROPs for transmission and (b) retrieving an object EOropframe for its transmission.

TX, RX, ROP send

Thread TX

EOpacket\* pkt;

uint16\_t numofrops2tx;

eo\_transceiver\_Transmit(txrx1, &pkt, &numofrops2tx);

< use pkt to send the UDP packet to the socket >

EOhostTransceiver

EOnvSet

EOtransceiver

EOtransmitter

EOreceiver

EOconfManager

Thread RX

EOpacket rxpkt;

< read from the socket and fill rxpkt>

uint16\_t numofrxrops;

eo\_transceiver\_Receive(txrx1, &pkt, &numofrxrops, NULL);

// the above function also calls the update() functions

Thread ROP send

Thread ROP send

eOropdescriptor\_t ropdes;

< fill the ropdes >

eo\_transceiver\_rop\_occasional\_Load (txrx1, &ropdes);

**Figure 15**: Use of EOtransceiver by multiple threads.

These three activities can be executed by at least three different threads (one for transmission, one for reception, and many for occasional insertion of ROPs), thus some sort of protection vs. concurrent use of shared resources is required.

The protection can be configured to be externally provided or to be internal. In the first case, every method of EOtransceiver must be mutually exclusive and protected by a mutex or a semaphore. In the second case, the methods of EOtransceiver protect only those sections which manipulate shared data.

### Add an occasional ROP set<> to the transceiver

It is possible to add occasional ROPs to the transceiver in the following way. The example shows how to set the PID of position for the joint 1 (the second one). The used method requires a description of the ROP which mimics all the possibilities of the protocol, but in here we simply send a set<> without time or signature and without confirmation request. The identification of the variable comes through its ID which can be obtained from its endpoint, entity, index and tag.

How to add a set<> ROP to EOhostTransceiver

// here we have the variable that we want to set

eOmc\_joint\_config\_t jointcfg = { }; // fill with desired values

void \*var\_data = &jointcfg.pidposition;

uint16\_t var\_size = sizeof(jointcfg.pidposition);

eOnvID32\_t var\_id = eoprot\_ID\_get( eoprot\_endpoint\_management, // the endpoint: mc

eoprot\_entity\_mc\_joint, // the entity: joint

1, // the index: the second one

eoprot\_tag\_mc\_joint\_config\_pidposition // the tag: the pid position

);

// then we need to build a rop descriptor

eOropdescriptor\_t ropdescr =

{

.control =

{

.confinfo = eo\_ropconf\_none,

.plustime = 0,

.plussign = 0,

.rqsttime = 0,

.rqstconf = 0,

.version = EOK\_ROP\_VERSION\_0

},

.ropcode = eo\_ropcode\_set,

.size = var\_size, // of the target variable

.id32 = var\_id, // of the target variable

.data = var\_data, // of the target variable

.signature = 0,

.time = 0

};

// now we can add the rop as an occasional transmission

eOresult\_t res = eo\_transceiver\_rop\_regular\_Load(txrx1, &ropdescr);

**Figure 16**: The occasional ROP set<> is added using a method of EOtransceiver.

### Send a ROPframe

Once all the ROPs to transmit are inserted inside the transceiver, and typically at a regular pace, it is possible to retrieve the packet to be transmitted. The transceiver internally forms a EOropframe and fills it inside a EOpacket (a data structure which contains a payload, its size, the destination IP address and port).

The EOpacket must be given to the socket for transmission. On the PC104 it is a ACE socket.

How to retrieve a EOpacket from EOhostTransceiver

// here is a handle to a packet

EOpacket\* packet = NULL;

uint16\_t numberofrops = 0;

// then we retrieve the pointer of the fully formed packet internal to the transceiver

// numberofrops contains the number of ROPs inside the EOropframe contained inside the payload

eOresult\_t res = eo\_transceiver\_Transmit(txrx1, &packet, &numberofrops);

uint8\_t \*payload\_data;

uint16\_t payload\_size;

eOipv4addr\_t ipaddr;

eOipv4port ipport;

res = eo\_packet\_Payload\_Get(packet, &payload\_data, &payload\_size);

res = eo\_packet\_Addressing\_Get(packet, &ipaddr, &ipport);

// now we can send payload\_size bytes contained inside payload\_data to a given ip address and port.

// by the way … ip and port returned by eo\_transceiver\_Transmit() are those specified in the brd1cfg

// variable used to create the transceiver.

**Figure 16**: The occasional ROP set<> is added using a method of EOtransceiver.

### Add an occasional ROP ask<> to the transceiver

It is possible to ask for a given value by forming a ask<> ROP. In the following we show the request of the entire configuration of joint 1 of board 1.

How to add a ask<> ROP to EOhostTransceiver

// then we define the variable that we want to ask for

eOmc\_joint\_config\_t jointcfg = { }; // fill with desired values

void \*var\_data = NULL; // we don’t set thus the ROP does not have any payload

uint16\_t var\_size = sizeof(eOmc\_joint\_config\_t);

eOnvID32\_t var\_id = eoprot\_ID\_get( eoprot\_endpoint\_management, // the endpoint: mc

eoprot\_entity\_mc\_joint, // the entity: joint

1, // the index: the second one

eoprot\_tag\_mc\_joint\_config // the tag: the config

);

// then we need to build a rop descriptor

eOropdescriptor\_t ropdescr =

{

.control =

{

.confinfo = eo\_ropconf\_none,

.plustime = 0,

.plussign = 0,

.rqsttime = 0,

.rqstconf = 0,

.version = EOK\_ROP\_VERSION\_0

},

.ropcode = eo\_ropcode\_ask,

.size = var\_size, // of the target variable

.id32 = var\_id, // of the target variable

.data = var\_data,

.signature = 0,

.time = 0

};

// now we can add the rop as an occasional transmission

eOresult\_t res = eo\_transceiver\_rop\_regular\_Load(txrx1, &ropdescr);

**Figure 16**: The occasional ROP ask<> is added using a method of EOtransceiver.

### Manage the reception of a ROP say<>

After having sent a ROP ask<ID>, the remote board replies with a ROP say<ID, value>. The EOhostTransceiver decodes the ROP, copies its value inside its RAM and calls an update function, which can be overridden to alert higher layers of the reception.

update() callback at reception of a say<> inside EOhostTransceiver

// the function is empty by default and for motion control is weakly defined inside EoProtocolMC\_fun.c

\_\_weak extern void eoprot\_fun\_UPDT\_mc\_joint\_config(const EOnv\* nv, const eOropdescriptor\_t\* rd) {}

// the function can be redefined as in the following

extern void eoprot\_fun\_UPDT\_mc\_joint\_config(const EOnv\* nv, const eOropdescriptor\_t\* rd)

{

// which joint?

eOprotIndex\_t index = eoprot\_ID2index(rd->id);

// which is the value ? and its size?

eOmc\_joint\_config\_t config; eOmc\_joint\_config\_t\* cfg; uint16\_t size;

// mode 1: through a method of the EOnv. IT IS THE SAFEST (it uses the mutex to make a safe copy)

eo\_nv\_Get(nv, eo\_nv\_strg\_volatile, &config, &size); // RAM is copied into config

// mode 2: through direct access to protected fields of the EOnv.

cfg = (eOmc\_joint\_config\_t\*) nv->ram; // pointer to RAM

size = nv->rom.capacity;

// mode 3: through the rop descriptor. IT IS THE QUICKEST and CLEANEST

cfg = (eOmc\_joint\_config\_t\*) rd->data;

size = rd->size;

eOropcode\_t ropcode = rd->ropcode;

// what if I want a consecutive id? I must know the board number.

// I have the IP and I need a function to retrieve it from it

eOprotBRD\_t brd = fromIP2brd(rd->ip);

eOprotProgNumber\_t prog = eoprot\_id2prognum(brd, rd->id);

// now I alert higher levels

alert\_motioncontrol\_received\_reply(brd, rd->id, cfg, size, prog, ropcode);

}

**Figure 16**: The ROP say<> is received and the relevant update() function is called.

### Managing a ask<> and say<>

When the host wants to know the remote values of a variable it sends a ROP ask<> and waits for a ROP say<>. The host may want to implement a blocking mechanism which stops the thread asking the value until the task managing ROP decoding alerts it about the reception. It can be useful using the signature feature, a 32-bit number that is specified in the ask<> and which must be contained in the say<>.

See example below.

update() callback used to unblock a semaphore

// this function asks the config of a motor and waits until a reply is reached. The BOARD is BRD1

extern void get\_remote\_value\_by\_rop(eOnvID32\_t id, void\* value, uint16\_t\* size)

{

// if not passed externally ...

id = eoprot\_ID\_get( eoprot\_endpoint\_management, eoprot\_entity\_mc\_motor,

0, eoprot\_tag\_mc\_motor\_config);

Semaphore\* sem = new Semaphore(0); // counter is already zero.

eOropdescriptor\_t ropdescr = {0};

memcpy(&ropdescr.control, & eok\_ropctrl\_basic, sizeof(eOropctrl\_t));

ropdescr.plussign = 1; // PLUS SIGNATURE !!!!!!!!!

ropdescr.ropcode = eo\_ropcode\_ask;

ropdescr.signature = (uint32\_t) &sem, // valid ONLY IF pointers are 32 bits

// now we add the rop as an occasional transmission

eOresult\_t res = eo\_transceiver\_rop\_regular\_Load(txrx1, &ropdescr);

// and we wait until some other thread increment the semaphore.

sem->decrement();

// in here if we have received the say<> with our unique signature.

// then get the value and the size. We can do it in many ways. See below

get\_remote\_value\_RAM\_mapped(id, value, size);

delete sem; // remember to delete it ...

}

extern void eoprot\_fun\_UPDT\_mc\_motor\_config(const EOnv\* nv, const eOropdescriptor\_t\* rd)

{

if((1 == rd->control.plussign) && (eo\_ropcode\_say == rd->ropcode))

{

Semaphore\* sem = (Semaphore\*) rd->signature;

sem->increment();

}

// yes: that’s all.

}

**Figure 16**: A task sends a ask<ID> with a given signature and waits until some other task which receives the relevant say<ID, val> increments a semaphore inside the update function.

### Have access of a given variable from its ID

How to access a variable from its ID

// these functions retrieve a value of avariable. The BOARD is BRD1

get\_remote\_value\_RAM\_mapped0(eOnvID32\_t id, void\* value, uint16\_t\* size)

{

// mode 0: by EOnv. Access to variables is protected vs concurrency by the mutex of EOnvSet.

EOnv nv;

eo\_nvset\_NV\_Get(nvset1, ip1, id, &nv);

eo\_nv\_Get(&nv, eo\_nv\_strg\_volatile, value, size);

}

extern void get\_remote\_value\_RAM\_mapped1(eOnvID32\_t id, void\* value, uint16\_t\* size)

{

// mode 1: by library. Access to variables is not protected vs concurrency.

\*size = eoprot\_variable\_sizeof\_get(brd1, id);

void\* temp = eoprot\_variable\_ramof\_get(brd1, id);

memcpy(value, temp, \*size);

}

**Figure 16**: There are two ways to retrieve the value and size of a variable from its ID: using the EOnv method or using directly the protocol library.

### Build IDs

The IDs are built using the method eoprot\_ID\_get() of the protocol library, where its arguments are taken in the following way. The endpoint and entity are taken from file EoProtocol.h, the index is just a number, and the tags are taken from file EoProtocolXX.h, where XX is AS for analog sensors, MC for motion control, MN for management, SK for skin etc.

How to build IDs

#include “EoProtocol.h”

#include “EoProtocolMC.h”

#include “EoMotionControl.h”

eOprotID32\_t id;

uint8\_t MaxIndex;

// joint …

MaxIndex = eoprot\_entity\_numberof\_get( brd1, // from 0 to eoprot\_boards\_maxnumberof-1

eoprot\_endpoint\_motioncontrol, // see enum eOprot\_endpoint\_t

eoprot\_entity\_mc\_joint // see enum eOprot\_entity\_t

);

id = eoprot\_ID\_get( eoprot\_endpoint\_motioncontrol, // see enum eOprot\_endpoint\_t

eoprot\_entity\_mc\_joint, // see enum eOprot\_entity\_t

3, // from 0 to MaxIndex-1

eoprot\_tag\_mc\_joint\_status // see enum eOprot\_tag\_mc\_joint\_t

);

eOmc\_joint\_status\_t joint\_status;

**Figure 16**: An ID in motion control.

How to build IDs

#include “EoProtocol.h”

#include “EoProtocolAS.h”

#include “EoAnalogSensors.h”

eOprotID32\_t id;

uint8\_t MaxIndex;

// strain …

MaxIndex = eoprot\_entity\_numberof\_get( brd1, // from 0 to eoprot\_boards\_maxnumberof-1

eoprot\_endpoint\_analogsensors, // see enum eOprot\_endpoint\_t

eoprot\_entity\_as\_strain // see enum eOprot\_entity\_t

);

// the strain is present only if MaxIndex is > 0.

id = eoprot\_ID\_get( eoprot\_endpoint\_analogsensors, // see enum eOprot\_endpoint\_t

eoprot\_entity\_as\_strain, // see enum eOprot\_entity\_t

0, // from 0 to MaxIndex-1

eoprot\_tag\_as\_strain\_config // see enum eOprot\_tag\_mc\_joint\_t

);

eOas\_strain\_config\_t strain\_cfg;

**Figure 16**: An ID in analog sensors.

## Operations on the board

### Build a ROP say<>

### Manage the reception of a ROP set<>

### Have access of a given variable from inside the callback function

### Have access of a given variable from its ID

### Configure the regular transmission of a ROP

Configuration of a board

Here is described how a board is configured to use a number of endpoints with a given amount of entities.

Tell:

1. The board must define some constants and data structures such as the ones in file brdxx.[c, h]
2. Refer to xxxx

## The eOprot\_Bxx files

Change of an endpoint

Here is described how to modify an existing endpoint by adding a tag to an existing entity or to add a new entity to an existing endpoint and even how to add a new endpoint.

## Adding a tag on an existing entity

To do:

1. Add the relevant field in the entity struct (file EoMotionControl.h)
2. Add the relevant tag (file EOprotocolMC.h)
3. Add the descriptor and put it inside the folded descriptor array
4. Add the callback functions

## Adding an entire entity

## Adding a complete endpoint