The embOBJ object library

This document describes the embOBJ object library for use on the ARM and DSPIC 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 TBD 1](#_Toc377370774)

[2 Some useful rubbish 2](#_Toc377370775)

[2.1 The object EOtheBOARDtransceiver 3](#_Toc377370776)

[2.1.1 The object EOnvSet 4](#_Toc377370777)

TBD

This document describes tbd.

Some useful rubbish

The ROPframe exchange between the host (PC104) and the board (EMS) is done using two specialised objects, the EOhostTransceiver and the EOtheBOARDtransceiver, which reside respectively on the PC104 and the EMS. The board object is a singleton as there must be only one 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 and 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 a board, the EOhostTransceiver queries its EOnvSet about presence of that variable on the remote board, and then the EOtransceiver builds the ROP and puts 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 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 an ask<> ROP and sends it to the board. The board processes the ROP and forms 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.

There is also the case in which the value of the PID is logically associated to the board but it is effectively owned by another board in a CAN subnet. In such a case, the variable with value is said to be proxied and the actions called by the ROPs must be propagated to the owner of the variable.

The set<> received by a board shall behave in the same manner as the non-proxied case, but the user must use the callback to send the received value to the owner of the variable over the CAN bus.

The ask<> received by a board shall not generate an automatic ask<> but will instead execute a callback in which the user must send over the CAN bus proper messages to the owner of the variable to request the value back. Upon reception of the value, the user must manually feed the value to the transceiver which shall send the say<> back to the host.

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

**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

**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

**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.

The eOprot\_B01.c file

static const eOnvset\_EPcfg\_t s\_eoprot\_b01\_theEPcfgs[] =

{

{ // management

EO\_INIT(.endpoint) eoprot\_endpoint\_management,

EO\_INIT(.dummy) 0,

EO\_INIT(.epram\_sizeof) sizeof(eOprot\_b01\_management\_t),

EO\_INIT(.fptr\_ram\_initialise) eoprot\_fun\_INITIALISE\_mn,

EO\_INIT(.protif) (eOnvset\_protocol\_Interface\_t\*)&eoprot\_eonvset\_Interface

},

{ // motion-control

EO\_INIT(.endpoint) eoprot\_endpoint\_motioncontrol,

EO\_INIT(.dummy) 0,

EO\_INIT(.epram\_sizeof) sizeof(eOprot\_b01\_motioncontrol\_t),

EO\_INIT(.fptr\_ram\_initialise) eoprot\_fun\_INITIALISE\_mc,

EO\_INIT(.protif) (eOnvset\_protocol\_Interface\_t\*)&eoprot\_eonvset\_Interface },

{ // analog-sensors

EO\_INIT(.endpoint) eoprot\_endpoint\_analogsensors,

EO\_INIT(.dummy) 0,

EO\_INIT(.epram\_sizeof) sizeof(eOprot\_b01\_analogsensors\_t),

EO\_INIT(.fptr\_ram\_initialise) eoprot\_fun\_INITIALISE\_as,

EO\_INIT(.protif) (eOnvset\_protocol\_Interface\_t\*)&eoprot\_eonvset\_Interface

},

{ // hello-world

EO\_INIT(.endpoint) eoprot\_endpoint\_helloworld,

EO\_INIT(.dummy) 0,

EO\_INIT(.epram\_sizeof) sizeof(eOprot\_b01\_helloworld\_t),

EO\_INIT(.fptr\_ram\_initialise) eoprot\_fun\_INITIALISE\_hw,

EO\_INIT(.protif) (eOnvset\_protocol\_Interface\_t\*)&eoprot\_eonvset\_Interface

}

}; EO\_VERIFYsizeof(s\_eoprot\_b01\_theEPcfgs, sizeof(eOnvset\_EPcfg\_t)\*(eoprot\_b01\_endpoints\_numberof));

**Figure 50**: The new fake file eOprot\_B01.c file with hello world endpoint.

The eOprot\_B01.c file

const uint8\_t eoprot\_b01\_hw\_entities\_numberofeach[eohw\_entities\_numberof] =

{

eoprot\_b01\_hw\_entity00\_numberof,

eoprot\_b01\_hw\_entity01\_numberof

};

**Figure 51**: The new fake file eOprot\_B01.c file with hello world endpoint.

The eOprot\_B01.c file

extern eOresult\_t eoprot\_b01\_Initialise(eObool\_t islocal)

{

// must initialise the mc, the mn, the ...

eoprot\_config\_endpoint\_entities(eoprot\_b01\_boardnumber, eoprot\_endpoint\_management,

eoprot\_b01\_mn\_entities\_numberofeach);

eoprot\_config\_endpoint\_entities(eoprot\_b01\_boardnumber, eoprot\_endpoint\_motioncontrol,

eoprot\_b01\_mc\_entities\_numberofeach);

eoprot\_config\_endpoint\_entities(eoprot\_b01\_boardnumber, eoprot\_endpoint\_analogsensors,

eoprot\_b01\_as\_entities\_numberofeach);

eoprot\_config\_endpoint\_entities(eoprot\_b01\_boardnumber, eoprot\_endpoint\_helloworld,

eoprot\_b01\_hw\_entities\_numberofeach);

if(eobool\_true == islocal)

{

eoprot\_config\_board\_local(eoprot\_b01\_boardnumber);

}

return(eores\_OK);

}

**Figure 52**: The new fake file eOprot\_B01.c file with hello world endpoint.