Sintesi

ADVANCED IN MECHATRONICS

C4G Open Library

User's Guide

Version 1.2.2

Authors: Sabino Colonna, Giovanni Iacca, Giovanni Totaro

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

This user's guide refers to *C4G Open Library* v1.17.00, that works in conjunction with Comau C4G system software v3.20.305 with "C4G Open" option enabled.

C4G Open Library, distributed under the terms of GNU LGPL (GNU Lesser General Public License), is a C++ library whose aim is interfacing a GNU/Linux/RTAI/RTnet PC with "C4G Open" solution, in order to use all the operative modes provided by the Comau C4G controller. In this way you can control an industrial robot by an external device or supply the C4G with measures coming from external sensors.

The control of the robot is obtained by the development of PC-running applications that make use of the *C4G Open Library* APIs.

The hardware/software target of *C4G Open Library* is an x86 architecture with GNU/Linux 2.6.x/RTAI 3.5/RTnet 0.9.9 operating system, endowed with a Fast Ethernet RTnet-compliant network adapter.

In the following sections we present an overview of the C4G Open system (§ 1.1) and of the PC software components necessary to use the *C4G Open Library* (§ 1.2 and § 1.3).

1.1 C4G Open system overview

C4G Open is a software extension of the Comau C4G controller for industrial robots that allows a C4G to interact with an external device like a PC. As shown in Fig. 1.1, a C4G Open system is constituted by the following elements:

- the SMP+ (System Master Processor Plus), that contains C4G system software, PDL2 interpreter and trajectory generator;
- the DSA (Digital Servo Amplifier), that contains the control code for position, ve-

locity and current loops;

- a PC, that is the external device interacting with C4G;
- a Fast Ethernet switch.

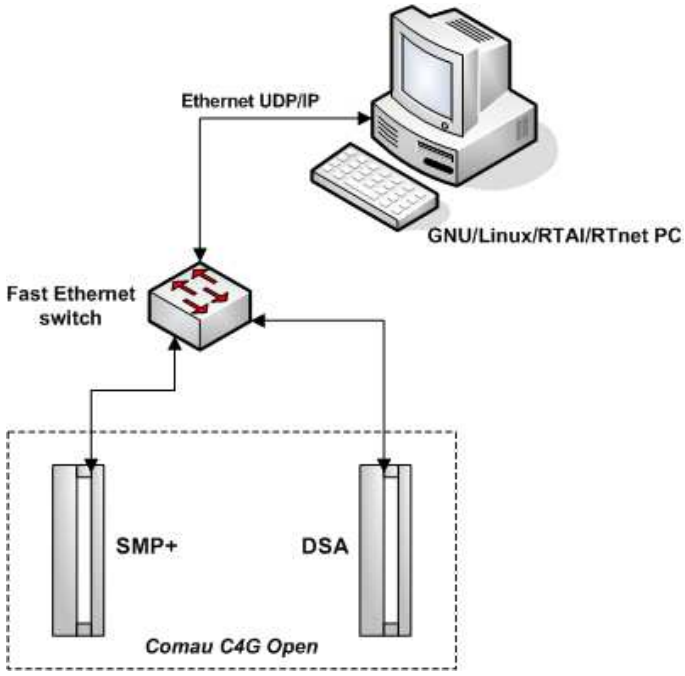

Figure 1.1: General scheme of C4G Open System

The communication between PC and C4G is based on UDP/IP protocol: PC acts as a server on 10.2.12.149:1000, while C4G acts as a client running on SMP+ at 10.2.12.150:1000. Moreover, the communication requires hard real-time capabilities, since PC must reply to C4G in the established cycle time (1 ms or 2 ms): this requirement is satisfied by means of RTnet protocol stack, described in § 1.3.

The application layer of PC-C4G communication is based upon the exchange of a packet whose structure is depicted in Fig. 1.2. Each packet has a three-fields header and as many nine-fields blocks as the number of open-enabled axes, each of which contains the informations corresponding to the used operative mode.

The operative modes actually supported by *C4G Open Library* and C4G Open extension of Comau C4G controller are the following ones:

- Mode 0, that is the normal behaviour of C4G with the additional requirement of a hard real-time communication with the PC;

Figure 1.2: Structure of a PC-C4G communication packet

- Mode 0', that is a debug mode aiming at verifying packets exchange between C4G and PC;
- Mode 1, that is the most complete open mode, since the PC must supply every open-enabled axis with target position, target velocity, feedforward velocity and feedforward current;
- Mode 2, that requires the PC to supply every open-enabled axis with feedforward velocity and feedforward current;
- Mode 4, that requires the PC to supply every open-enabled axis with absolute target position and velocity;
- Mode 5, that requires the PC to supply every open-enabled axis with relative target position and velocity;
- Mode 7, that requires the PC to supply every open-enabled axis with a delta target position;
- Mode 8, that requires the PC to supply every open-enabled axis with feedforward velocity;
- Mode 9, that requires the PC to supply every open-enabled axis with a delta current.

The following special modes are also provided:

- Mode 500, also called *Active freezing*, that allows the PC to perform non real-time operations and then to resume the normal communication with C4G;
- Mode 501, also called *Driving On*, that is active until the transient phase following the "Drive ON" procedure has finished;
- Mode 502, also called *Passive freezing*, that allows the C4G to perform non real-time operations, while the PC is waiting for the arrival of a packet;
- Mode 504, that allows to close the communication;
- Mode 505, that allows the PC to schedule a "Drive OFF" request;
- Mode 506, useful to restart the client process running on SMP+ without restarting the C4G;

- Mode 508, called *Following Error*, set when a *following error threshold* overcoming is detected.

For more informations about C4G Open features, including a description of each operative mode, please refer to [1].

You can find the detailed description of *C4G Open Library* APIs in Chapter 3 and the guidelines on how to use them for building your own C4G control applications in § 4.1.

1.2 RTAI overview

GNU/Linux is a standard time-sharing operating system that provides good average performance and highly sophisticated services, but it suffers from a lack of real-time support. To obtain a timing correctness behaviour, it is necessary to make some changes in the kernel sources, i.e. in the interrupt handling and scheduling policies.

RTAI (Real Time Application Interface) [2] is an open source project aiming at providing the standard GNU/Linux kernel with real-time capabilities. Strictly speaking, RTAI is not a real-time operating system, such as VXworks or QNX: it is based on the GNU/Linux kernel, providing the ability to make it fully preemptable.

RTAI offers the same services of the GNU/Linux kernel core, adding the features of an industrial real-time operating system. It consists basically of an interrupt dispatcher: RTAI mainly traps the peripherals interrupts and, if necessary, re-routes them to Linux. It is not an intrusive modification of the kernel; it uses the concept of HAL (Hardware Abstraction Layer) to get informations from Linux and to trap some fundamental functions. HAL provides few dependencies to GNU/Linux kernel, leading to a simple adaptation in the GNU/Linux kernel, an easy RTAI port from version to version of Linux and an easier use of other operating systems instead of RTAI.

The LXRT layer allows you to develop your own hard real-time applications in *user space*, so to make easier and faster the prototyping phase, while MUP and SMP schedulers provide support for multi-processor applications.

RTAI considers Linux as a background task running when no real-time activities occur.

1.3 RTnet overview

RTnet is an open source hard real-time network protocol stack for RTAI [3]. It makes use of standard Ethernet hardware and supports several popular NIC (Network Interface

Card) chipsets, including Gigabit Ethernet.

RTnet implements UDP/IP, ICMP and ARP in a deterministic way and provides POSIX-compliant APIs to be used both in kernel modules and in *user space* processes by means of RTAI LXRT layer (see § 1.2).

To avoid unpredictable collisions and congestions on Ethernet, an additional protocol layer called *RTmac* controls the media access. A dedicated Ethernet segment is required to guarantee bounded transmission delays, but RTnet also includes a mechanism to tunnel non real-time traffic like TCP/IP over RTmac, thus allowing a "single-cable" solution for connecting control systems.

The design of the RTnet stack is depicted in Fig. 1.3.

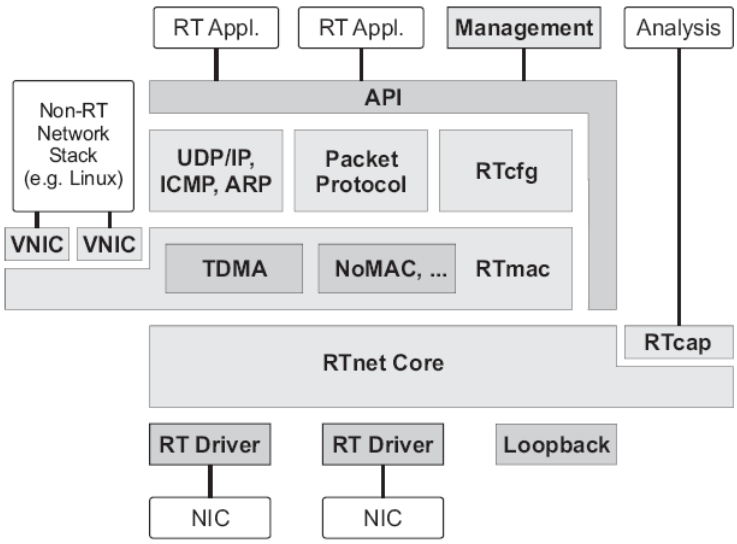

Figure 1.3: Overview of RTnet protocol stack

One of the most interesting features of RTnet software approach is its independence of specific hardware for supporting hard real-time communications, so that you can rely on standard network cards, like those you can use for office, provided that they are supported with an RTnet hard real-time driver.

For a complete list of all 10/100 Mbps RTnet-compliant chipsets, please refer to Appendix A.

Chapter 2

Preparing your system

In this section we provide the guidelines to make your system ready to use the *C4G Open Library*, including a list of all HW/SW requirements, and the procedures to install RTAI/RTnet and the library itself.

2.1 Hardware and Software requirements

Hardware requirements:

- PC with x86-compatible CPU at least of Pentium class;
- RTnet-compliant network card adapter (10/100 Mbps);
- three UTP cat.5 or superior network cables and a Fast Ethernet switch.

Software requirements:

- Linux vanilla kernel 2.6.x patched with RTAI 3.5/3.6;
- RTnet 0.9.9/0.9.10;
- GNU development tools for x86 applications.

Tested HW/SW:

- Intel Pentium 4, 2.8 GHz;
- D-Link System Inc DFE-500TX Fast Ethernet network adapter (DECchip 21140 chipset);

- Ubuntu 7.04 "The Feisty Fawn" CD x86 edition [4];
- Linux vanilla kernel 2.6.23.17 [5];
- RTAI 3.6.1 [2];
- RTnet 0.9.10 [3];
- GNU g++ 4.1.2 [6];
- GNU bash 3.2.13(1) [6].

2.2 Get hardware informations

One of the most important steps before using the *C4G Open Library* is checking configuration of the network card you have chosen for hard real-time communications between PC and C4G.

On Ubuntu distribution you can get informations about your NIC, including the model of its chipset and its IRQ line, by means of the following bash command:

```
lspci -v
```

Listing 2.1 shows the output of the command corresponding to the tested network card:

```
00:0b.0 Ethernet controller: Digital Equipment Corporation DECchip 21140 [FasterNet] (rev 22)
        Subsystem: D-Link System Inc DFE-500TX Fast Ethernet
        Flags: bus master, medium devsel, latency 32, IRQ 10
        I/O ports at c800 [size=128]
        Memory at e8143000 (32-bit, non-prefetchable) [size=128]
        [virtual] Expansion ROM at 30000000 [disabled] [size=256K]
```

Listing 2.1: Partial output of `lspci` command

Moreover, you can download the `hwinfo` utility from *Synaptic Package Manager* (typing `sudo synaptic`): it gives you detailed informations (including vendor and device IDs) about your hardware configuration and, in particular, about all installed PCI devices, by executing the following command:

```
hwinfo --pci
```

As you can see in listing 2.2 at line 14, the `hwinfo` utility gives you also the name of the standard Linux driver of a device (in this case, a network card).

```
19: PCI 0b.0: 0200 Ethernet controller
  [Created at pci.281]
  UDI: /org/freedesktop/Hal/devices/pci_1011_9
  Unique ID: JNkJ.AujCtKsDPG3
  SysFS ID: /devices/pci0000:00/0000:00:0b.0
  SysFS BusID: 0000:00:0b.0
  Hardware Class: network
  Model: "D-Link DFE-500TX Fast Ethernet"
  Vendor: pci 0x1011 "Digital Equipment Corporation"
  Device: pci 0x0009 "DECchip 21140 [FasterNet]"
  SubVendor: pci 0x1186 "D-Link System Inc"
  SubDevice: pci 0x1100 "DFE-500TX Fast Ethernet"
  Revision: 0x22
  Driver: "tulip"
  Driver Modules: "tulip"
  Device File: eth1
  I/O Ports: 0xc800-0xc87f (rw)
  Memory Range: 0xe8143000-0xe814307f (rw,non-prefetchable)
  Memory Range: 0x30000000-0x3003ffff (ro,prefetchable,disabled)
  IRQ: 10 (8 events)
  HW Address: 00:40:05:37:36:e4
  Module Alias: "pci:v00001011d00000009sv00001186sd00001100bc02sc00i00"
  Driver Info #0:
    Driver Status: tulip is active
    Driver Activation Cmd: "modprobe tulip"
  Config Status: cfg=new, avail=yes, need=no, active=unknown
```

Listing 2.2: Partial output of `hwinfo` command

Make sure that the IRQ line used by your NIC chipset is not shared by other devices, otherwise unpredictable behaviours could happen during hard real-time communications. Type `cat /proc/interrupts` to see whether the IRQ line is shared: if so, try to change your network card PCI slot, or disable unused devices, or remove kernel modules corresponding to IRQ-sharing devices.

```
  0:     291251   XT-PIC-XT   timer
  1:          2   XT-PIC-XT   i8042
  2:          0   XT-PIC-XT   cascade
  3:        119   XT-PIC-XT   SiS SI7012
  5:      16674   XT-PIC-XT   eth0
  6:          5   XT-PIC-XT   floppy
  7:          1   XT-PIC-XT   parport0
  8:          3   XT-PIC-XT   rtc
  9:          0   XT-PIC-XT   ehci_hcd:usb2
 10:          8   XT-PIC-XT   ohci_hcd:usb1, eth1
 11:      67755   XT-PIC-XT   ohci_hcd:usb3, radeon@pci:0000:01:00.0
 14:      27017   XT-PIC-XT   ide0
```

Listing 2.3: Output of `cat /proc/interrupts`

For example, if your network card is sharing the IRQ line 10 with devices using the `ohci_hcd` module (as in listing 2.3 at line 10), you have to type the following command as `root` or with `sudo`:

```
rmmod ohci_hcd
```

It could be a good solution using only PS/2 keyboard and mouse so to avoid USB devices and reduce the likelihood of IRQ collisions.

2.3 Installing GNU/Linux/RTAI/RTnet

The very first step consists in installing on a PC a GNU/Linux distribution of your choice (we suggest you Ubuntu 7.04), then compiling a Linux vanilla kernel 2.6.x patched with RTAI 3.5/3.6-x, to obtain a hard real-time capable host system [7].

Eventually, install RTnet 0.9.9/0.9.10 to provide your system with hard real-time communication capabilities [3].

If you prefer a semi automatic way to make your system ready, we suggest you to visit Orchestra Control Engine website [8], register and download the following package:

```
Orchestra_v1.0-2_RTOS.tar.bz2
```

and then follow the instructions described in [9].

Before proceeding in *C4G Open Library* installation and usage, please restart your system booting the just patched kernel (if you want the right kernel to boot automatically, modify the `/boot/grub/menu.lst` file).

2.4 Installing the C4G Open Library

Put *C4G Open Library* tarball in a directory for which you have write permissions (e.g. your home), then decompress it:

```
tar xjvf C4Gopen-v1.17.00.tar.bz2
```

Enter the just created `C4Gopen` directory and compile *C4G Open Library* by typing `make` in `C4GLibrary` directory. Eventually install it, by typing `make install` as `root` or with `sudo`.

Alternatively, double-left-click on `C4Gopen-v1.17.00.deb` file, left-click on "Install Package" and follow the installer instructions.

Remember that the default installation directory is `/usr/local/C4Gopen`.

2.5 Loading RTAI modules

The very first step to use the *C4G Open Library* consists in loading all the necessary RTAI modules, in order to provide your system with hard real-time capabilities. The only thing you have to do is launching the `loadRTAI` script (see listing 2.4) that you can find in `/usr/local/C4Gopen/bin` directory, by typing the following command as `root` or with `sudo`:

```
./loadRTAI
```

```
#!/bin/bash

RTAI_SRC_DIR=/usr/src/rtai
RTAI_MODULES_DIR=/usr/realtime/modules

sync
make -f ${RTAI_SRC_DIR}/GNUmakefile dev
insmod ${RTAI_MODULES_DIR}/rtai_hal.ko
insmod ${RTAI_MODULES_DIR}/rtai_lxrt.ko
insmod ${RTAI_MODULES_DIR}/rtai_sem.ko
insmod ${RTAI_MODULES_DIR}/rtai_mbx.ko
insmod ${RTAI_MODULES_DIR}/rtai_rtdm.ko
sync
```

Listing 2.4: The `loadRTAI` script

Notice that `rtai_rtdm` module (line 12) is requested to use RTnet (see § 2.6).

2.6 Loading RTnet modules

In order to allow your PC to satisfy hard real-time communication requirements, you have to load all the necessary RTnet kernel modules and properly configure your C4G Open subnet. Listing 2.5 shows the content of the `loadRTnet` script that you can find in `/usr/local/C4Gopen/bin` directory.

```
#!/bin/bash

# This script file :
# - removes Linux kernel module of the driver of your NIC;
# - loads RTnet modules necessary to make use of RTnet;
# - loads RTnet driver of your NIC;
# - configures the C4G Open subnet.

RTAI_MOD_DIR=/usr/realtime/modules
RTNET_MOD_DIR=/usr/local/rtnet/modules
RTNET_BIN_DIR=/usr/local/rtnet/sbin

ETH_DRV=linux_driver_name (e.g. 8139too)
RT_ETH_DRV=rtnet_driver_name (e.g. rt_8139too)

LOCAL_IP=10.2.12.149
TARGET_IP=10.2.12.150

TARGET_MAC_ADDR=smp+_mac_address (e.g. 00:04:50:48:36:88)

if [ -z "$ETH_DRV" ]; then
    echo
    echo -e "ETH_DRV not set in loadRTnet script"\
        "(please refer to \"C4G Open Library User's Guide\").\n"
    exit 1
fi

if [ -z "$RT_ETH_DRV" ]; then
    echo
    echo -e "RT_ETH_DRV not set in loadRTnet script"\
        "(please refer to \"C4G Open Library User's Guide\").\n"
    exit 1
fi

if [ -z "$TARGET_MAC_ADDR" ]; then
    echo
    echo -e "TARGET_MAC_ADDR not set in loadRTnet script"\
        "(please refer to \"C4G Open Library User's Guide\").\n"
    exit 1
fi

mknod /dev/rtnet c 10 240

rmmod ${ETH_DRV}

insmod ${RTNET_MOD_DIR}/rtnet.ko
insmod ${RTNET_MOD_DIR}/rtipv4.ko
insmod ${RTNET_MOD_DIR}/rtpacket.ko
insmod ${RTNET_MOD_DIR}/rt_loopback.ko
```

```
insmod ${RTNET_MOD_DIR}/${RT_ETH_DRV}.ko
51
${RTNET_BIN_DIR}/rtifconfig rtlo up 127.0.0.1
53 ${RTNET_BIN_DIR}/rtifconfig rteth0 up ${LOCAL_IP}
${RTNET_BIN_DIR}/rtroute add ${TARGET_IP} ${TARGET_MAC_ADDR} dev rteth0
```

Listing 2.5: The `loadRTnet` script

The bash variable `ETH_DRV` at line 13 refers to the name of the kernel module of the Linux driver for your network card, while the bash variable `RT_ETH_DRV` at line 14 refers to the name of the kernel module of RTnet hard real-time driver for your network card. See § 2.2 to learn how it is possible to get informations about a NIC installed on your PC and find the driver name of your network card.

To find the RTnet driver name suiting your network card, please refer to the list in Appendix A.

The bash variable `TARGET_MAC_ADDR` at line 19 is the MAC address of the network interface of SMP+ board that is involved in hard real-time communications with the PC. If you do not know how to get SMP+ MAC address, turn on the C4G controller and type the following command in `/usr/local/rtnet/sbin` as `root` or with `sudo`:

```
./rtroute solicit 10.2.12.150 dev rteth0
```

then type, in the same directory, `./rtroute`, and get the desired MAC address from the route table.

Notice that `LOCAL_IP` value is "10.2.12.149", while `TARGET_IP` value is "10.2.12.150", that is respectively the IP address of the PC and that of SMP+ inside the C4G Open subnet (see § 1.1).

Before launching the script, open it with a text editor as `root` or with `sudo`, and:

- insert the Linux driver name at line 13;
- insert the RTnet driver name at line 14;
- insert the SMP+ MAC address at line 19;

For example, if your NIC chipset is RTL8139, the standard Linux driver name is `8139too`, while the RTnet driver is `rt_8139too`, so the above lines will be:

```
ETH_DRV=8139too
RT_ETH_DRV=rt_8139too
```

After having modified and saved the script, you have to launch it, by simply executing the following command, as `root` or with `sudo`, in `/usr/local/C4Gopen/bin` directory:

```
./loadRTnet
```

Now your system is ready to communicate with C4G Open.

Chapter 3

C4G Open Library reference

In this section a detailed description of public methods of `C4gOpen` class is presented. These methods can be functionally grouped in:

- methods for initializing a `C4gOpen` instance, that is:
  a. opening and binding the RTnet socket;
  b. creating the RTAI hard real-time task;
  c. waiting for the arrival of the initialization packet from C4G.
- methods for sending/receiving a packet to/from C4G;
- *getter* methods for reading values in packets coming from C4G;
- *setter* methods for writing values in packets going to C4G;
- methods for verifying the correctness of operations.
- methods for finalizing a `C4gOpen` instance, that is:
  a. destroying RTAI hard real-time task;
  b. closing the RTnet socket.

For a complete reference of public/private members and methods of `C4gOpen` class, please enter the `/usr/local/C4Gopen/Documentation/html` directory and open the `index.html` file in a web browser.

3.1 C4gOpen Class Reference

Class containing members and methods to allow a PC to communicate with Comau C4Gopen.

```
#include <C4gOpen.hpp>
```

Public Member Functions

- C4gOpen ()
  Default constructor.
- C4gOpen (int portNumber)
  Overloaded constructor.
- ~C4gOpen ()
  Destructor.
- bool start (bool taskToInit=true)
  Initialize a C4gOpen instance.
- bool receive ()
  Receive a communication packet from C4G.
- bool send ()
  Send a communication packet to C4G.
- void stop (bool taskToClose=true)
  Finalize a C4gOpen instance.
- bool errorOccurred ()
  Check if an error occurred.
- ErrorCodes getLastError ()
  Get the value of the last occurred error.
- long getNumberOfOpenAxes ()
  Get the number of axes in open mode.

- long getSampleTime ()

  Get the communication cycle time in milliseconds.

- float getCalibrationConstant (int arm, int axis)

  Get the calibration constant of an axis in gear rotations.

- float getCurrentLimit (int arm, int axis)

  Get the current limit of an axis in ampere.

- float getTxRate (int arm, int axis)

  Get the transmission rate of an axis.

- float getKinInflCoeff54 (int arm)

  Get the kinematic influence coefficient of axis 4 on axis 5.

- float getKinInflCoeff64 (int arm)

  Get the kinematic influence coefficient of axis 4 on axis 6.

- float getKinInflCoeff65 (int arm)

  Get the kinematic influence coefficient of axis 6 on axis 5.

- float getFollowingErrorThreshold (int arm, int axis)

  Get the following error threshold of an axis in gear rotations.

- bool isInDriveOn (int arm)

  Check if an arm is in "Drive On".

- long getFunctionality (int arm)

  Get open mode functionality.

- long getMode (int arm)

  Get the open mode of an arm.

- float getTargetPosition (int arm, int axis)

  Get the target position of an axis in gear rotations.

- float getTargetVelocity (int arm, int axis)

  Get the target velocity of an axis.

- float getActualPosition (int arm, int axis)

Get the actual position of an axis in gear rotations.

- float getActualVelocity (int arm, int axis)
  Get the actual velocity of an axis.

- float getTargetCurrent (int arm, int axis)
  Get the target current of an axis in ampere.

- float getExtra1 (int arm, int axis)
  Get extra info 1 of an axis.

- float getExtra2 (int arm, int axis)
  Get extra info 2 of an axis.

- float getExtra3 (int arm, int axis)
  Get extra info 3 of an axis.

- bool exitFromOpen (int arm)
  Order an arm to exit from open mode.

- bool setMode (int arm, long mode)
  Set the open mode of an arm.

- bool setTargetPosition (int arm, int axis, float targetPosition)
  Set the target position of an axis in gear rotations.

- bool setTargetVelocity (int arm, int axis, float targetVelocity)
  Set the target velocity of an axis.

- bool setMeasure (int arm, int axis, float measure)
  Set the measure given by an external sensor.

- bool setFeedForwardVelocity (int arm, int axis, float ffwVelocity)
  Set the feedforward velocity of an axis.

- bool setFeedForwardCurrent (int arm, int axis, float ffwCurrent)
  Set the feedforward current of an axis.

- bool setExtra1 (int arm, int axis, float extra1)
  Set extra info 1 of an axis.

- bool setExtra2 (int arm, int axis, float extra2)

  Set extra info 2 of an axis.

- bool setExtra3 (int arm, int axis, float extra3)

  Set extra info 3 of an axis.

3.1.1 Detailed Description

Class containing members and methods to allow a PC to communicate with Comau C4Gopen.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 C4gOpen::C4gOpen ()

Default constructor.

This method constructs an instance of the C4gOpen class, initializing some of its members.

3.1.2.2 C4gOpen::C4gOpen (int *portNumber*)

Overloaded constructor.

This method constructs an instance of the C4gOpen class, calling the default constructor and initializing the C4G-PC socket port number to the number passed as argument.

Parameters:

portNumber Port number of C4G-PC UDP socket.

3.1.3 Member Function Documentation

3.1.3.1 bool C4gOpen::start (bool *taskToInit* = `true`)

Initialize a C4gOpen instance.

This method is a public interface to startWithoutRtTask() and startWithRtTask(), depending on if you have just created an RTAI task before using a C4gOpen object or not.

Parameters:

taskToInit Flag indicating if a brand new RTAI task has to be created (default is true).

Returns:

True if all the initialization procedure goes well, false otherwise.

3.1.3.2 bool C4gOpen::receive ()

Receive a communication packet from C4G.

This method receives a communication packet coming from C4G after having set fields of the incoming communication packet to their default value by calling resetCommPacket-Rx().

When a timeout set to 1 s expires, this method returns false.

Returns:

True if the number of received bytes equals the size of an Rx communication packet (with numberOfOpenAxes fields), false otherwise and lastError is set to RT_-SOCKET_RECEIVE_ERROR.

3.1.3.3 bool C4gOpen::send ()

Send a communication packet to C4G.

This method sends a communication packet to C4G, then set fields of the outgoing communication packet to their default value by calling resetCommPacketTx(). If any error has occurred, this method schedules a DRIVE OFF request, by setting the mode of the outgoing packet to C4G_OPEN_DRIVE_OFF. If a following error threshold has been overcome (see checkFollowingError()), the mode of the outgoing packet is set to C4G_OPEN_FOLLOWING_ERROR.

Returns:

True if the packet has been successfully sent, that is number of sent bytes equals the size of a Tx communication packet (with numberOfOpenAxes fields), false otherwise and lastError is set to RT_SOCKET_SEND_ERROR.

3.1.3.4 void C4gOpen::stop (bool *taskToClose* = `true`)

Finalize a C4gOpen instance.

This method calls closeRealTimeTask() if a brand new RTAI task has been previously created, and closeRealTimeSocket() to close a previously opened RTnet socket.

Parameters:

taskToClose Flag indicating if a brand new created RTAI task has to be deleted (default is true).

3.1.3.5 long long C4gOpen::getCommunicationTimeout ()

Get the communication timeout.

Returns:

The socket timeout during normal communication between C4G and PC in nanoseconds.

3.1.3.6 void C4gOpen::setCommunicationTimeout (long long *timeout*)

Set the communication timeout.

Parameters:

timeout The socket timeout during normal communication between C4G and PC in nanoseconds.

3.1.3.7 bool C4gOpen::errorOccurred ()

Check if an error occurred.

Returns:

True if an error occurred, false otherwise.

3.1.3.8 ErrorCodes C4gOpen::getLastError ()

Get the value of the last occurred error.

Returns:

The value of the last occurred error.

3.1.3.9 void C4gOpen::resetError ()

Reset the last occurred error.

This method clears the last occurred error, allowing to keep the communication going on when, e.g., "Drive Off" command is sent to C4G or a "following error overcome" is detected by the PC.

3.1.3.10 long C4gOpen::getNumberOfOpenAxes ()

Get the number of axes in open mode.

Returns:

Number of axes in open mode.

3.1.3.11 long C4gOpen::getSampleTime ()

Get the communication cycle time in milliseconds.

Returns:

Communication cycle time in milliseconds.

3.1.3.12 float C4gOpen::getCalibrationConstant (int *arm*, int *axis*)

Get the calibration constant of an axis in gear rotations.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Calibration constant of the specified axis in gear rotations.

3.1.3.13 float C4gOpen::getCurrentLimit (int *arm*, int *axis*)

Get the current limit of an axis in ampere.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Current limit of the specified axis in ampere.

3.1.3.14 float C4gOpen::getTxRate (int *arm*, int *axis*)

Get the transmission rate of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Transmission rate of the specified axis.

3.1.3.15 float C4gOpen::getKinInflCoeff54 (int *arm*)

Get the kinematic influence coefficient of axis 4 on axis 5.

Parameters:

arm Number of the arm.

Returns:

Kinematic influence coefficient of axis 4 on axis 5.

3.1.3.16 float C4gOpen::getKinInflCoeff64 (int *arm*)

Get the kinematic influence coefficient of axis 4 on axis 6.

Parameters:

arm Number of the arm.

Returns:

Kinematic influence coefficient of axis 4 on axis 6.

3.1.3.17 float C4gOpen::getKinInflCoeff65 (int *arm*)

Get the kinematic influence coefficient of axis 6 on axis 5.

Parameters:

arm Number of the arm.

Returns:

Kinematic influence coefficient of axis 6 on axis 5.

3.1.3.18 float C4gOpen::getFollowingErrorThreshold (int *arm*, int *axis*)

Get the following error threshold of an axis in gear rotations.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Following error threshold of the specified axis in gear rotations.

3.1.3.19 bool C4gOpen::isInDriveOn (int *arm*)

Check if an arm is in "Drive On".

Parameters:

arm Number of the arm.

Returns:

True if the specified arm is in "Drive On", false otherwise.

3.1.3.20 long C4gOpen::getFunctionality (int *arm*)

Get open mode functionality.

This method gets the functionality provided for a certain open mode on the specified axis.

Parameters:

arm Number of the arm.

Returns:

The open mode functionality, -1 if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.21 long C4gOpen::getMode (int *arm*)

Get the open mode of an arm.

Parameters:

arm Number of the arm.

Returns:

The open mode of the specified arm.

3.1.3.22 float C4gOpen::getTargetPosition (int *arm*, int *axis*)

Get the target position of an axis in gear rotations.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Target position of the specified axis in gear rotations, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.23 float C4gOpen::getTargetVelocity (int *arm*, int *axis*)

Get the target velocity of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Target velocity of the specified axis, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.24 float C4gOpen::getActualPosition (int *arm*, int *axis*)

Get the actual position of an axis in gear rotations.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Actual position of the specified axis in gear rotations, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.25 float C4gOpen::getActualVelocity (int *arm*, int *axis*)

Get the actual velocity of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Actual velocity of the specified axis, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.26 float C4gOpen::getTargetCurrent (int *arm*, int *axis*)

Get the target current of an axis in ampere.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Target current of the specified axis in ampere, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.27 float C4gOpen::getDynamicModel (int *arm*, int *axis*)

Get the dynamic model of an axis in ampere.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Dynamic model of the specified axis in ampere, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.28 float C4gOpen::getDiagonalInertia (int *arm*, int *axis*)

Get the diagonal inertia of an axis in $kg * m^2$.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Diagonal inertia of the specified axis in $kg * m^2$, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.29 float C4gOpen::getExtra1 (int *arm*, int *axis*)

Get extra info 1 of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Extra info 1 of the specified axis, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.30 float C4gOpen::getExtra2 (int *arm*, int *axis*)

Get extra info 2 of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Extra info 2 of the specified axis, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.31 float C4gOpen::getExtra3 (int *arm*, int *axis*)

Get extra info 3 of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

Returns:

Extra info 3 of the specified axis, NAN if an error has occurred or the specified arm and/or axis are not valid.

3.1.3.32 bool C4gOpen::exitFromOpen (int *arm*)

Order an arm to exit from open mode.

Parameters:

arm Number of the arm.

Returns:

True if the specified arm is valid, false otherwise.

3.1.3.33 bool C4gOpen::setMode (int *arm*, long *mode*)

Set the open mode of an arm.

This method sets the open mode of an arm, if and only if:

- an exitFromOpen() has called in the previous step;
- or the actual mode is 0;
- or the specified mode is equal to the actual one.

Parameters:

arm Number of the arm.

mode Open mode to set.

Returns:

True if the mode is successfully set, false otherwise and lastError is appropriately set:

- INVALID_OPEN_MODE if the specified mode is not a valid open mode (i.e. it is not included in validOpenModes);
- SET_MODE_NOT_ALLOWED if the operation is not permitted (i.e. the above conditions are not satisfied).

3.1.3.34 bool C4gOpen::setTargetPosition (int *arm*, int *axis*, float *targetPosition*)

Set the target position of an axis in gear rotations.

This method sets the target position of the specified axis, checking if the following error threshold has been overcome.

Parameters:

arm Number of the arm.

axis Number of the axis.

targetPosition Target position of the specified axis in gear rotations.

Returns:

True if the target position has been successfully set, false if:

- the specified arm and/or axis are not valid;
- the following error threshold is overcome.

3.1.3.35 bool C4gOpen::setTargetVelocity (int *arm*, int *axis*, float *targetVelocity*)

Set the target velocity of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

targetVelocity Target velocity of the specified axis.

Returns:

True if the target velocity has been successfully set, false if the specified arm and/or axis are not valid.

3.1.3.36 bool C4gOpen::setMeasure (int *arm*, int *axis*, float *measure*)

Set the measure given by an external sensor.

Parameters:

arm Number of the arm.

axis Number of the axis.

measure Measure given by the external sensor.

Returns:

True if the measure has been successfully set, false if the specified arm and/or axis are not valid.

3.1.3.37 bool C4gOpen::setFeedForwardVelocity (int *arm*, int *axis*, float *ffwVelocity*)

Set the feedforward velocity of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

ffwVelocity Feedforward velocity of the specified axis.

Returns:

True if the feedforward velocity has been successfully set, false if the specified arm and/or axis are not valid.

3.1.3.38 bool C4gOpen::setFeedForwardCurrent (int *arm*, int *axis*, float *ffwCurrent*)

Set the feedforward current of an axis.

This method sets the feedforward current of the specified axis, checking if the given value overcomes current limit of the specified axis: if so this method sets the feedforward current to the current limit of the specified axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

ffwCurrent Feedforward current of the specified axis.

Returns:

True if the feedforward current has been successfully set, false if the specified arm and/or axis are not valid.

3.1.3.39 bool C4gOpen::setDeltaCurrent (int *arm*, int *axis*, float *deltaCurrent*)

Set delta current of an axis.

This method sets delta current of the specified axis, checking if the given value added to the actual current value overcomes current limit of the specified axis: if so this method sets the feedforward current to the current limit of the specified axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

deltaCurrent Delta current of the specified axis.

Returns:

True if delta current has been successfully set, false if the specified arm and/or axis are not valid.

3.1.3.40 bool C4gOpen::setExtra1 (int *arm*, int *axis*, float *extra1*)

Set extra info 1 of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

extra1 Extra info 1 of the specified axis.

Returns:

True if extra info 1 has been successfully set, false if the specified arm and/or axis are not valid.

3.1.3.41 bool C4gOpen::setExtra2 (int *arm*, int *axis*, float *extra2*)

Set extra info 2 of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

extra2 Extra info 2 of the specified axis.

Returns:

True if extra info 2 has been successfully set, false if the specified arm and/or axis are not valid.

3.1.3.42 bool C4gOpen::setExtra3 (int *arm*, int *axis*, float *extra3*)

Set extra info 3 of an axis.

Parameters:

arm Number of the arm.

axis Number of the axis.

extra3 Extra info 3 of the specified axis.

Returns:

True if extra info 3 has been successfully set, false if the specified arm and/or axis are not valid.

3.2 C4gOpenConstants Reference

Constants and enums used by C4gOpen class.

Defines

- #define DEFAULT_PORT_NUMBER 1000

  Default port number of PC-C4G socket.

- #define MAX_NUM_ARMS 4

  Maximum number of arms.

- #define MAX_NUM_AXES_PER_ARM 10

  Maximum number of axes per arm.

- #define MAX_NUM_OPEN_AXES 20

  Maximum number of axes in 'open mode'.

- #define EXIT_FROM_OPEN 1

  Arm status used in the status field of an outgoing communication packet.

- #define DRIVE_ON 5

  Arm status used in the status field of an incoming communication packet.

Enumerations

- enum ErrorCodes

  Error codes corresponding to errors occurring in the use of C4gOpen class methods.

Variables

- const long validOpenModes [ ]
  Vector of the valid open modes that the user can set.

3.2.1 Enumeration Type Documentation

3.2.1.1 enum ErrorCodes

Error codes corresponding to errors occurring in the use of C4gOpen class methods.

Enumerator:

NO_ERROR

RT_TASK_INIT_ERROR Error during the initialization of RTAI task.

RT_SOCKET_INIT_ERROR Error during the initialization of RTnet socket.

RT_SOCKET_OPEN_ERROR Error while opening RTnet socket.

RT_SOCKET_BINDING_ERROR Error while binding RTnet socket.

RT_SOCKET_SEND_ERROR Error while sending a packet to C4G.

RT_SOCKET_RECEIVE_ERROR Error while receiving a packet from C4G.

INIT_PACKET_UNEXPECTED_SEQ_NUM Unexpected sequence number of the initialization packet.

INIT_PACKET_RECEIVE_ERROR Error while receiving the initialization packet from C4G.

INVALID_ARM_NUMBER Access to an invalid arm number.

INVALID_AXIS_NUMBER Access to an invalid axis number.

ARM_NOT_IN_OPEN_MODE Access to a not 'open mode' arm.

AXIS_NOT_IN_OPEN_MODE Access to a not 'open mode' axis.

INVALID_OPEN_MODE Setting of an invalid open mode.

SET_MODE_NOT_ALLOWED Call to C4gOpen::setMode when C4gOpen::canChangeMode is false.

OPERATION_NOT_ALLOWED Call to an operation not allowed by the actual open mode.

SET_TARGET_POSITION_MISSING Target position not set even if requested by the actual open mode.

SET_TARGET_VELOCITY_MISSING Target velocity not set even if requested by the actual open mode.

SET_MEASURE_MISSING Measure from external sensor not set even if requested by the actual open mode.

SET_FFW_VELOCITY_MISSING Feedforward velocity not set even if requested by the actual open mode.

SET_FFW_CURRENT_MISSING Feedforward current not set even if requested by the actual open mode.

SET_DELTA_CURRENT_MISSING Delta current not set even if requested by the actual open mode.

FOLLOWING_ERROR_OVERCOME Following error threshold overcome.

Chapter 4

Using C4G Open Library

In this section we provide the guidelines to write your own C4G Open control application using the *C4G Open Library* APIs. Moreover, a brief description of the testsuite included in *C4G Open Library* package is given.

4.1 Building an application

As you can see in listing 4.1, building a C4G Open control application requires very few steps:

1. creating a `C4gOpen` class instance (line 7);
2. starting the `C4gOpen` instance, that is opening the RTnet socket, creating the RTAI task and waiting for the initialization packet (line 8);
3. putting all your elaborations inside a cycle in which:
   - receive a packet from C4G (line 51);
   - make computations according to the used open mode;
   - send a packet containing the just computed values to C4G (line 25).
4. stopping the `C4gOpen` instance, that is destroying the RTAI task and closing the RTnet socket.

```
#include <C4gOpen.hpp>
2
#define ARM 1
4
```

```
int main()
{
    C4gOpen c4gOpen;
    if (c4gOpen.start())
    {
        do
        {
            if (c4gOpen.receive())
            {
                // Put here your computations.
                c4gOpen.send();
            }
        }
        while (!c4gOpen.errorOccurred() &&
               c4gOpen.getMode(ARM) != C4G_OPEN_EXIT)
    }

    c4gOpen.stop();

    return 0;
}
```

Listing 4.1: Template of a C4G Open application

Table 4.1 lists all the C++ #define corresponding to open modes supported by the *C4G Open Library*.

Remember that you can check the correctness of each operation at every moment, by means of the `errorOccurred()` method and, if you want, get the corresponding error code by means of the `getLastError()` method, so that you can take the necessary countermeasures. Please, refer to § 3.2 to get a description of all possible errors.

4.1.1 Compiling and launching the application

After having coded your own C4G Open control application, you have to compile it by writing a `Makefile` like that of listing 4.2.

```
INCLUDES = -I/usr/src/linux/include -I/usr/local/C4Gopen/include \
           -I/usr/realtime/include -I/usr/local/rtnet/include
LIBS = -lm /usr/realtime/lib/liblxrt.a -lpthread -L/usr/local/C4Gopen/lib -lC4gOpen

default:
    g++ MyApplication.cpp $(INCLUDES) $(LIBS) -o MyApplication
```

Listing 4.2: Makefile of a C4G Open application

Please, notice that a command included in a Makefile rule (line 6) must be preceded by a "`tab`" space.

Now that your executable is ready, launch it as `root` or with `sudo`, then start the C4G controller, provided that you have properly set C4G system variables, that is `$ARM_DATA[i].C4GOPEN_JNT_MASK` and `$ARM_DATA[i].C4GOPEN_MODE[j]` (see [1]).

Listing 4.3 shows how to open-enable axes 1-6 of arm 1 via PDL2. Remember that, after having open-enabled the desired axes, you have to execute CSA (Configure Save All) to save the system configuration and CCRC (Configure Controller Restart Cold) to restart the system.

C++ #define	**Value**
C4G_OPEN_MODE_0	0
C4G_OPEN_MODE_0_DEBUG	10
C4G_OPEN_MODE_1	1
C4G_OPEN_MODE_1_STANDBY	11
C4G_OPEN_MODE_2	2
C4G_OPEN_MODE_2_STANDBY	12
C4G_OPEN_MODE_4	4
C4G_OPEN_MODE_4_STANDBY	14
C4G_OPEN_MODE_5	5
C4G_OPEN_MODE_5_STANDBY	15
C4G_OPEN_MODE_7	7
C4G_OPEN_MODE_7_STANDBY	17
C4G_OPEN_MODE_8	8
C4G_OPEN_MODE_9	9
C4G_OPEN_ACTIVE_FREEZING	500
C4G_OPEN_DRIVING_ON	501
C4G_OPEN_PASSIVE_FREEZING	502
C4G_OPEN_EXIT	504
C4G_OPEN_DRIVE_OFF	505
C4G_OPEN_CLIENT_RESTART	506
C4G_OPEN_FOLLOWING_ERROR	508

Table 4.1: C4G open modes supported by the library

```
PROGRAM GoOpen NOHOLD, &PA

VAR index : INTEGER
CONST
    arm = 1
    numAxes = 6

BEGIN
-- Set open axes mask (open-enable all the axes)
    $ARM_DATA[arm].C4GOPEN_JNT_MASK := 0x3F
```

```
-- Set mode 0
    FOR index := 1 TO numAxes DO
        $ARM_DATA[arm].C4GOPEN_MODE[index] := 0
    ENDFOR
END GoOpen
```

Listing 4.3: PDL2 program to open-enable all the axes of arm 1

Listing 4.4 shows how to set mode 'x' on the open-enabled axes of the arm 1 via PDL2.

```
PROGRAM SetModeX NOHOLD, &PA

VAR index : INTEGER
CONST
    arm = 1
    numAxes = 6

ROUTINE isOpen(axis : INTEGER) : BOOLEAN

BEGIN
    RETURN (BIT_TEST($ARM_DATA[arm].C4GOPEN_JNT_MASK, axis))
END isOpen

BEGIN
    FOR index := 1 TO numAxes DO
        IF isOpen(arm, index) THEN
            $ARM_DATA[arm].C4GOPEN_MODE[index] := modeX
        ENDIF
    ENDFOR
END SetModeX
```

Listing 4.4: PDL2 program to set mode 'x' for all open-enabled axes

Note: before launching your control applications, make sure to have executed the `loadRTAI` and `loadRTnet` scripts, as described in § 2.5 and § 2.6.

4.1.2 Notes on C4G Open Library

Here brief notes on the usage of *C4G Open Library* APIs are listed:

- the server process running on the PC (i.e. your control application) exits when a timeout equal to 5 minutes expires while it is waiting for the arrival of the initialization packet;

- the server process running on the PC (i.e. your control application) exits when a timeout equal to 1 s expires during the normal communication. If necessary, e.g. for *Mode 502* ("Passive freezing"), change the timeout value by means of `setCommunicationTimeout` method before entering your control cycle (i.e. after having started the `C4gOpen` instance);

- the server process on the PC (i.e. your control application) must be launched before starting the C4G;

- in order to restart the client process running on SMP+ without restarting the C4G by means of *Mode 506*, first start the server on the PC;

- whenever an error occurs, a packet with a "Drive OFF" request will be automatically sent to C4G, unless you have handled the error before calling the `send()` method. In this case C4G goes back in *Mode 0* and, by calling `resetError` method before receiving a new packet, the communication will go on in *Mode 0*;

- whenever a *following error threshold* overcoming is detected, a packet with *Mode 508* ("Following Error") will be automatically sent to C4G. In this case C4G goes back in *Mode 0* and, by calling `resetError` method before receiving a new packet, the communication will go on in *Mode 0*;

- whenever a *current limit* overcoming is detected, a packet filled with saturated current value will be automatically sent to C4G.

4.2 C4G Open Library testsuite

In `/usr/local/C4Gopen/testsuite` directory you can find some application examples showing the usage of *C4G Open Library* APIs with different open modes. All the examples have been tested on a Comau NS16-1.65 robot, with the PC-C4G communication cycle time set to 2 ms.

4.2.1 TestModeODebug

Description

Aim of this test is verifying the communication between PC and C4G, both in transmission and in reception, since *Mode 0'* requires PC to send back the received packets. The test will last until a *Mode 504* request has been scheduled on C4G via PDL2.

Launching the test

Enter the `/usr/local/C4Gopen/testsuite/TestMode0Debug` directory and type the following command as `root` or with `sudo`:

```
./TestMode0Debug
```

then enable the open mode for all the axes and start C4G in mode 0.

When the communication starts, "Drive ON" the robot and set *Mode 0'* on C4G via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server will close.

4.2.2 TestMode1

Description

Aim of this test is controlling ONLY the axis 6 of the robot in *Mode 1*, that is supplying axis 6 with target position, target velocity, feedforward velocity and feedforward current.

The test current value is equal to 5% of axis 6 current limit: a 1 Hz square wave between -testCurrent and +testCurrent is given, so that you will see axis 6 going backwards and forwards respect to the initial position. Target position is constantly set to the actual position, while target and feedforward velocity are constantly set to '0'.

Launching the test

Enter the `/usr/local/C4Gopen/testsuite/TestMode1` directory and type the following command as `root` or with `sudo`:

```
./TestMode1
```

then enable the open mode ONLY for axis 6 and start C4G in mode 0.

When the communication starts, "Drive ON" the robot and set *Mode 1* ONLY for axis 6 via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server will close.

4.2.3 TestMode2

Description

Aim of this test is controlling ONLY the axis 6 of the robot in *Mode 2*, that is supplying axis 6 with feedforward velocity and feedforward current.

The test current value is equal to 5% of axis 6 current limit: a 1 Hz square wave between -testCurrent and +testCurrent is given, so that you will see axis 6 going backwards and forwards respect to the initial position. Feedforward velocity is constantly set to '0'.

Launching the test

Enter the `/usr/local/C4Gopen/testsuite/TestMode2` directory and type the following command as `root` or with `sudo`:

```
./TestMode2
```

then enable the open mode ONLY for axis 6 and start C4G in mode 0.

When the communication starts, "Drive ON" the robot and set *Mode 2* ONLY for axis 6 via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server will close.

4.2.4 TestMode4

Description

Aim of this test is controlling the robot in *Mode 4*, that is supplying every open-enabled axis with absolute target position and target velocity according to a sinusoidal movement,

whose amplitude and frequency are chosen by the user, starting from the actual position of the robot (we suggest you to start from the calibration position). The test will last until a *Mode 504* request has been scheduled on C4G via PDL2.

C4G starts in *Mode 0* and the robot is driven off: PC begins to generate the trajectory when the "Drive ON" signal is given and C4G enters *Mode 4*. The test will last until a *Mode 504* request has been scheduled on C4G via PDL2.

Launching the test

Enter the `/usr/local/C4Gopen/testsuite/TestMode4` directory and type the following command as `root` or with `sudo`:

```
./TestMode4 amplitudeDeg freqHz enabledAxes
```

where

- `amplitudeDeg` is the sinusoid amplitude in degrees (for security reasons set a number below '10');
- `freqHz` is the sinusoid frequency in hertz;
- `enabledAxes` is a string containing the numbers of open-enabled axes (e.g. `146` if axes 1, 4, 6 have been open-enabled).

Enable the open mode for the axes corresponding to the `enabledAxes` argument and start C4G in mode 0.

When the communication starts, "Drive ON" the robot and set *Mode 4* on C4G via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server will close.

4.2.5 TestMode5

Description

Aim of the test is controlling the robot in *Mode 5*, that is supplying every open-enabled axis with relative target position and target velocity according to a sinusoidal movement,

whose amplitude and frequency are chosen by the user, starting from the actual position of the robot (we suggest you to start from the calibration position). The test will last until a *Mode 504* request has been scheduled on C4G via PDL2.

C4G starts in *Mode 0* and the robot is driven off: PC begins to generate the trajectory when the "Drive ON" signal is given and C4G enters *Mode 5*.

Launching the test

Enter the `/usr/local/C4Gopen/testsuite/TestMode5` directory and type the following command as `root` or with `sudo`:

```
./TestMode5 amplitudeDeg freqHz enabledAxes
```

where

- `amplitudeDeg` is the sinusoid amplitude in degrees (for security reasons set a number below '10');
- `freqHz` is the sinusoid frequency in hertz;
- `enabledAxes` is a string containing the numbers of open-enabled axes (e.g. `146` if axes 1, 4, 6 have been open-enabled).

Enable the open mode for the axes corresponding to the `enabledAxes` argument and start C4G in mode 0.

When the communication starts, "Drive ON" the robot and set *Mode 5* on C4G via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server will close.

4.2.6 TestMode8

Description

Aim of this test is controlling ONLY the axis 1 of the robot in *Mode 8*, that is supplying axis 1 with feedforward velocity.

Feedforward velocity is a 1 Hz sinusoidal signal, whose amplitude is equal to 0.0005 (delta motor turns).

Launching the test

Enter the `/usr/local/C4Gopen/testsuite/TestMode8` directory and type the following command as `root` or with `sudo`:

```
./TestMode8
```

then enable the open mode ONLY for axis 1 and start C4G in mode 0.

When the communication starts, "Drive ON" the robot and set *Mode 8* ONLY for axis 1 via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server will close.

4.2.7 TestMode9

Description

Aim of this test is controlling ONLY the axis 1 of the robot in *Mode 9*, that is supplying axis 1 with delta current.

Delta current is a 1 Hz sinusoidal signal, whose amplitude is equal to 25% of axis 1 current limit.

Launching the test

Enter the `/usr/local/C4Gopen/testsuite/TestMode9` directory and type the following command as `root` or with `sudo`:

```
./TestMode9
```

then enable the open mode ONLY for axis 1 and start C4G in mode 0.

When the communication starts, "Drive ON" the robot and set *Mode 9* ONLY for axis 1 via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server will close.

Chapter 5

OrchestraC4GLibrary

In this section we will introduce *OrchestraC4GLibrary*, i.e. a set of modules to be used in conjunction with OrchestraCore, one of the component of the Orchestra Control Engine suite [8], in order to develop a control application interfacing a Comau C4G Open system.

OrchestraC4GLibrary is distributed under the terms of OCL (Orchestra Community License) and its current version (1.17.00) works with *C4G Open Library* v1.17.00 and OrchestraCore v1.32.01 (included in Orchestra v1.5-2).

5.1 OrchestraCore overview

OrchestraCore is one of the component of the Orchestra Control Engine suite. It is a hard real-time framework based on x86 architecture and GNU/Linux 2.6.x/RTAI 3.6 operating system that allows the execution of control applications by handling synchronous/asynchronous and local/remote communications with other software entities, by supplying alarm/safety primitives and by providing the hard real-time periodic execution of discrete fixed-step algorithms obtained by the combination of dynamically loadable binary modules [10].

Each module has a certain number of input and/or output lines, can have internal states, both double-typed and user-typed, and can be characterized by some named parameters.

From an implementation point of view, each module is composed of a binary file (called DLC), an interface configuration XML file (called XMI), and an optional parameters configuration XML file (called XMP). At runtime you have also the possibility to change and save parameters by means of OrchestraHMI [11].

5.2 Installing OrchestraC4GLibrary

Before installing and using *OrchestraC4GLibrary*, you must install on your system both *C4G Open Library* v1.17.00 (see §2.4) and OrchestraCore v1.32.01 [10], then execute the following command in a Linux shell terminal:

```
bash install_OrchestraC4GLibrary-v1.17.00.run
```

After having read and accepted the Orchestra Community License, enter the just created `OrchestraC4GLibrary_v1.17.00` directory and decompress *OrchestraC4GLibrary* tarball:

```
tar xjvf OrchestraC4GLibrary-v1.17.00.tar.bz2
```

Enter the just created `OrchesraC4GLibrary` directory and compile *OrchesraC4GLibrary* by typing `make`. Then install it, by typing `make install` as `root` or `sudo make install` as normal user.

Alternatively, if you have installed *C4G Open Library* by means of .deb package (see §2.4), you can double-left-click on `OrchestraC4GLibrary-v1.17.00.deb` file, left-click on "Install Package" and follow the installer instructions.

Remember that the default installation directory is `/usr/local/C4Gopen/OrchestraC4GLibrary`.

5.3 OrchestraC4GLibrary overview

As you can see in Fig. 5.1, a typical control application running on OrchestraCore framework and interacting with a Comau C4G Open system is composed at least of two modules, i.e. `C4G Sensor` and `C4G Actuator`, that respectively receives and sends communication packets from/to C4G Open making use of *C4G Open Library* APIs (see Chapter 3).

Your own control algorithms are encapsulated within one or more modules linked with sensor and actuator so to form anyhow complex control loop topologies.

OrchestraC4GLibrary allows a control loop running on OrchestraCore framework to interact with a C4G Open system by providing the following items:

- source code and XMI file of a general purpose `C4G Sensor` module, i.e. working with any of the actually supported C4G open modes (see §4.1.1);

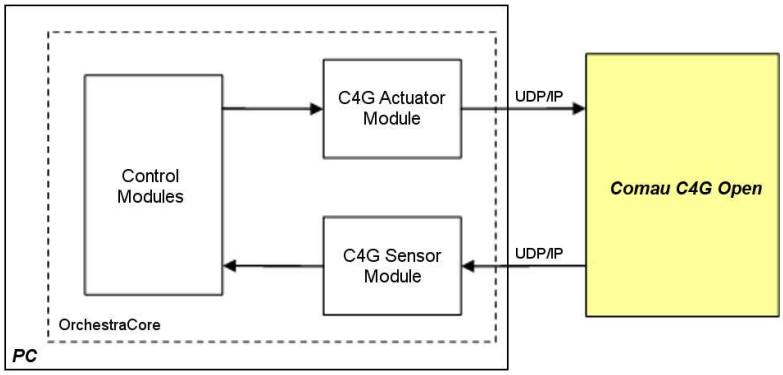

Figure 5.1: Scheme of a C4G control loop running with OrchestraCore

- source code and XMI file of a different `C4G Actuator` module for each actually supported C4G open mode.

In next sections a detailed description of `C4G Sensor` (see §5.4) and `C4G Actuator` (see §5.5) modules is given.

5.4 C4G Sensor

`C4G Sensor` is an OrchestraCore module that, by means of *C4G Open Library* APIs, receives communication packets from a C4G Open system and provides outputs corresponding to every field contained in received packets, apart from the currently used open mode: in other words, you can use `C4G Sensor` for any of the actually supported C4G open modes, linking all its outputs, or a subset of them, to the inputs of another module, depending on the control algorithms you are developing.

Keep in mind that the `C4G Sensor` is intended to be used with a Comau C4G Open system configured as follows:

- OPEN ARM: arm 1;
- OPEN AXES: axes 1 to 6 of arm 1.

You can, of course, modify the source code and the XMI file of `C4G Sensor` (or add an XMP file) if you need a different C4G Open configuration.

5.4.1 C4G Sensor DLC

Listing 5.1 contains the source code of `C4G Sensor` module.

```
#include <DLCInterface.hpp>
#include <C4gOpen.hpp>

#define ARM 1
#define NUM_OF_AXES 6

AUTHOR("Sintesi S.C.p.A.");
VERSION("1.17.00");

INITIALIZE_BEGIN
    ALLOCATE_EXTRA(0, 1, C4gOpen); // C4gOpen instance
INITIALIZE_END

STEP_BEGIN
    if (STEP_NUMBER == 1)
    {
        // Initialize C4gOpen instance without creating an RTAI task.
        if (!(EXTRA(0,0).start(false)))
            ALARM_WITH_MESSAGE(EXTRA(0,0).getLastError(),
                    "C4G Open: Failed to initialize communication.");

        do
        {
            EXTRA(0,0).receive();
            if (EXTRA(0,0).errorOccurred())
                ALARM_WITH_MESSAGE(EXTRA(0,0).getLastError(),
                        "C4G Open: Failed to initialize communication.");

            if (EXTRA(0,0).getMode(ARM) == C4G_OPEN_DRIVING_ON ||
                EXTRA(0,0).getMode(1) == C4G_OPEN_MODE_0)
            {
                EXTRA(0,0).send();
                if (EXTRA(0,0).errorOccurred())
                    ALARM_WITH_MESSAGE(EXTRA(0,0).getLastError(),
                            "C4G Open: Failed to initialize communication.");
            }
        }
        while (EXTRA(0,0).getMode(ARM) == C4G_OPEN_DRIVING_ON ||
               EXTRA(0,0).getMode(ARM) == C4G_OPEN_MODE_0);

        if (!EXTRA(0,0).errorOccurred()) EXTERNAL_SYNCHRO_DONE;
        else ALARM_WITH_MESSAGE(EXTRA(0,0).getLastError(),
                "C4G Open: Failed to initialize communication.");
```

```
44
        INFO_ONLY_MESSAGE("C4G Open: Communication successfully initialized.");
46  }

48  // Receive a packet from C4G.
    if (STEP_NUMBER > 1)
50  {
        if (EXTRA(0,0).receive()) EXTERNAL_SYNCHRO_DONE;
52      else ALARM_WITH_MESSAGE(EXTRA(0,0).getLastError(),
                "C4G Open: Failed to receive a packet.");
54  }
    if (EXTRA(0,0).getMode(ARM) == C4G_OPEN_EXIT) STOP_EXECUTION;
56
    // Write C4gOpen instance pointer to output #0.
58  OUTPUT(0,0) = (double)(unsigned long)&EXTRA(0,0);

60  for (unsigned int i = 0; i < NUM_OF_AXES; i++)
    {
62      // Write target position to output #1.
        OUTPUT(1,i) = EXTRA(0,0).getTargetPosition(ARM, i+1);
64      // Write target velocity to output #2.
        OUTPUT(2,i) = EXTRA(0,0).getTargetVelocity(ARM, i+1);
66      // Write actual position to output #3.
        OUTPUT(3,i) = EXTRA(0,0).getActualPosition(ARM, i+1);
68      // Write actual velocity to output #4.
        OUTPUT(4,i) = EXTRA(0,0).getActualVelocity(ARM, i+1);
70      // Write target current to output #5.
        OUTPUT(5,i) = EXTRA(0,0).getTargetCurrent(ARM, i+1);
72      // Write dynamic model to output #6.
        OUTPUT(6,i) = EXTRA(0,0).getDynamicModel(ARM, i+1);
74      // Write diagonal inertia to output #7.
        OUTPUT(7,i) = EXTRA(0,0).getDiagonalInertia(ARM, i+1);
76
    }
78 STEP_END

80 FINALIZE_BEGIN
    EXTRA(0,0).stop(false);
82  INFO_ONLY_MESSAGE("C4G Open: Communication successfully closed.");
    DEALLOCATE_EXTRA(0);
84 FINALIZE_END
```

Listing 5.1: `C4GSensor.cpp`

The include file `DLCInterface.hpp` (line 1) contains the definition of all the macros you can use to develop an OrchestraCore module. The include file `C4gOpen.hpp` (line 2) contains *C4G Open Library* API (see Chapter 3).

`C4G Sensor` module is functionally divided into:

- an initialization part (`INITIALIZE`), containing operations executed once and necessary to initialize `C4G Sensor`, i.e. allocate memory for an `EXTRA` element corresponding to the pointer to a `C4gOpen` class instance;

- a periodic execution part (`STEP`), in which code to be performed every control loop step is encapsulated. Notice that:

  - during the first step execution the `C4gOpen` instance is started (line 8) without creating a new RTAI task, since it is created and started by OrchestraCore framework itself;
  - at every step execution a communication packet is received from C4G (line 51);
  - at every step execution a reference to the `C4gOpen` instance is passed to `C4G Actuator` module on output line 0 (line 58) and the other values contained in the received communication packet are copied to output lines 1 to 7;
  - at every step execution a check whether *Mode 504* has been scheduled is performed, so that the control loop execution can be stopped (line 55);

- a finalization part (`FINALIZE`), containing operations executed once and necessary to finalize `C4G Sensor`, i.e. stop the `C4gOpen` instance and deallocate previously allocated memory for the `EXTRA` element.

After having written the source code of `C4G Sensor` module, the next step is its compilation by means of `makeDLC` utility: in this way the DLC file, dynamically loadable by OrchestraCore, will be created.

As you can see in listing 5.2 (a Makefile that call `makeDLC` with the right parameters), you need to specify the paths to `C4gOpen.hpp` header file and to RTnet include directory, the path to `C4G Open` library and, optionally, the destination path of your DLC.

```
C4GLIBRARY_INSTALL_DIR = /usr/local/C4Gopen
RTNET_INSTALL_DIR = /usr/local/rtnet

INCLUDES = -I$(C4GLIBRARY_INSTALL_DIR)/include \
           -I$(RTNET_INSTALL_DIR)/include
LIBS = -L$(C4GLIBRARY_INSTALL_DIR)/lib -lC4gOpen

default:
    makeDLC C4GSensor.cpp --includes="$(INCLUDES)" --libs="$(LIBS)"
```

Listing 5.2: Makefile of `C4GSensor.cpp`

5.4.2 C4G Sensor XMI

The XMI (XML Module Interface) file of a `C4G Sensor` module (see listing 5.3) contains informations about its outputs and extras.

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
   <!DOCTYPE Interface SYSTEM "xmi1.dtd">
3
   <Interface version="1.0">
5      <Outputs>
           <Out name="C4gOpen instance" noe="1" />
7          <Out name="Target Position" noe="6" unit="gear rotations" showable="true" />
           <Out name="Target Velocity" noe="6" unit="delta gear rotations" showable="true"/>
9          <Out name="Actual Position" noe="6" unit="gear rotations" showable="true" />
           <Out name="Actual Velocity" noe="6" unit="delta gear rotations" showable="true"/>
11         <Out name="Target Current" noe="6" unit="A" showable="true" />
           <Out name="Dynamic Model" noe="6" unit="A" showable="true" />
13         <Out name="Diagonal Inertia" noe="6" unit="kg*m^2" showable="true" />
       </Outputs>
15     <Extras noe="1" />
   </Interface>
```

Listing 5.3: XMI file of C4G Sensor

Notice that the header of the XMI file (line 2) contains a reference to `xmi1.dtd` for its validation during the parsing phase.

`C4G Sensor` module should not have any input line coming from other modules (because a sensor has only physical inputs coming directly from C4G Open system), but only one or more output lines, according to the topology of the control loop to which this module belongs. Notice that:

- the first output of `C4G Sensor` module is always the pointer to `C4gOpen` class instance;
- next seven outputs correspond to all values contained in a communication packet received from C4G; each of them is a 6 elements array since the `C4G Sensor` module is intended to be used with axes 1 to 6 of arm 1 open-enabled.

Every `C4G Sensor` module has at least one extra element (see the `Extras` section), which will be dinamically allocated during the runtime initialization of the module itself: it is the pointer to `C4gOpen` class instance that is sent to `C4G Actuator` module on output line '0' (see listing 5.1)

5.5 C4G Actuator

`C4G Actuator` is an OrchestraCore module that, by means of *C4G Open Library* APIs, sends communication packets to a C4G Open system, after having filled each field with the values coming from its input(s), according to the currently used open mode. For this reason there are as many `C4G Actuator` modules as the actually supported C4G open modes are, each of which has a different number of input lines linked to the outputs of other modules encapsulating your own control algorithms.

Keep in mind that `C4G Actuator` is intended to be used with a Comau C4G Open system configured as follows:

- OPEN ARM: arm 1;
- OPEN AXES: axes 1 to 6 of arm 1.

You can, of course, modify the source code and the XMI file of a `C4G Actuator` (or add an XMP file) if you need a different C4G Open configuration.

5.5.1 C4G Actuator DLC

Listing 5.4 contains the source code of the `C4G Actuator` module to be used in *Mode 1*.

```
#include <DLCInterface.hpp>
#include <C4gOpen.hpp>

#define ARM 1
#define NUM_OF_AXES 6

AUTHOR("Sintesi S.C.p.A.");
VERSION("1.17.00");

STEP_BEGIN
    // Get C4gOpen instance pointer from Sensor.
    C4gOpen *c4gOpen = (C4gOpen *)(unsigned long)INPUT(0,0);

    if (c4gOpen->getMode(ARM) == C4G_OPEN_MODE_1)
    {
        for (unsigned int i = 0; i < NUM_OF_AXES; i++)
        {
            c4gOpen->setTargetPosition(ARM, i+1, INPUT(1,i));
            c4gOpen->setTargetVelocity(ARM, i+1, INPUT(2,i));
```

```
20              c4gOpen->setFeedForwardVelocity(ARM, i+1, INPUT(3,i));
                c4gOpen->setFeedForwardCurrent(ARM, i+1, INPUT(4,i));
22          }
        }
24
        c4gOpen->send();
26
        if (c4gOpen->errorOccurred()) c4gOpen->resetError();
28  STEP_END
```

Listing 5.4: `C4GActuatorMode1.cpp`

The include file `DLCInterface.hpp` (line 1) contains the definition of all the macros you can use to develop an OrchestraCore module. The include file `C4gOpen.hpp` (line 2) contains *C4G Open Library* API (see Chapter 3).

`C4G Actuator` module has only a a periodic execution part (`STEP`), in which code to be performed every control loop step is encapsulated. Notice that:

- at every step execution the `C4gOpen` instance is got from `C4G Sensor` on input line 0 (line 12) and all the values needed for the currently used open mode (in this case *Mode 1*) are got from input lines 1 to 4 and used to fill the fields of an outgoing communication packet;
- at every step execution a communication packet is sent to C4G Open system (line 25);
- if an error occurred, it can be reset (line 27) or specifically handled, depending on what countermeasures you want to take.

After having written the source code of `C4G Actuator` module, the next step is its compilation by means of `makeDLC` utility: in this way the DLC file, dynamically loadable by OrchestraCore, will be created.

As you can see in listing 5.5 (a Makefile that call `makeDLC` with the right parameters), you need to specify the paths to `C4gOpen.hpp` header file and to RTnet include directory, the path to `C4G Open` library and, optionally, the destination path of your DLC.

```
C4GLIBRARY_INSTALL_DIR = /usr/local/C4Gopen
2 RTNET_INSTALL_DIR = /usr/local/rtnet

4 INCLUDES = -I$(C4GLIBRARY_INSTALL_DIR)/include \
                -I$(RTNET_INSTALL_DIR)/include
6 LIBS = -L$(C4GLIBRARY_INSTALL_DIR)/lib -lC4gOpen

8 default:
    makeDLC C4GActuator.cpp --includes="$(INCLUDES)" --libs="$(LIBS)"
```

Listing 5.5: Makefile of `C4GActuator.cpp`

5.5.2 C4G Actuator XMI

The XMI (XML Module Interface) file of a `C4G Actuator` module contains informations about its inputs.

Listing 5.6 represents the XMI file of `C4G Actuator` to be used in *Mode 1*.

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
   <!DOCTYPE Interface SYSTEM "xmi1.dtd">
3
   <Interface version="1.0">
5      <Inputs>
           <In name="C4gOpen instance" noe="1" />
7          <In name="Target Position" noe="6" />
           <In name="Target Velocity" noe="6" />
9          <In name="FFW Velocity" noe="6" />
           <In name="FFW Current" noe="6" />
11     </Inputs>
   </Interface>
```

Listing 5.6: XMI file of C4G Actuator for *Mode 1*

Notice that the header of the XMI file (line 2) contains a reference to `xmi1.dtd` for its validation during the parsing phase.

`C4G Actuator` module should not have any output line going to other modules (because an actuator has only physical outputs going directly to C4G Open system), but only one or more input lines, according to the topology of the control loop to which this module belongs and the currently used open mode. Notice that:

- the first input of `C4G Actuator` module is always the pointer to `C4gOpen` class instance, coming from output '0' of `C4G Sensor` module (see 5.3);
- the other inputs correspond to all values that will be used to fill the communication packet to be sent to C4G; each of them is a 6 elements array since the `C4G Actuator` module is intended to be used with axes 1 to 6 of arm 1 open-enabled.

5.6 C4G Control Loop

Every OrchestraCore control loop is characterized by an XCL (XML Control Loop) file, that is the XML representation of the overall structure of the control loop that will be executed by OrchestraCore: in fact it contains not only the reference to the modules

being part of the loop and to the corresponding DLC, XMI and XMP files, but also the description of the control loop topology.

Listing 5.7 is a template XCL file of a control loop interfacing a C4G Open system; it contains only the reference to `C4G Sensor` and `C4G Actuator` modules, but it should be completed with elements corresponding to other modules that you make use of in your topology.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE ControlLoop SYSTEM "xcl1.dtd">

<ControlLoop name="C4Gopen Control Loop" period="0.002"
        timing-source="External" timing-module="C4Gopen Sensor" lead-perc="50">
    <Module
        name="C4Gopen Sensor"
        dlc="C4GSensor.dlc"
        xmi="C4GSensor.xmi" >
    </Module>
        ...
        ...
    <Module
        name="C4Gopen Actuator"
        dlc="C4GActuator.dlc"
        xmi="C4GActuator.xmi" >
        <Input name="C4Gopen Sensor" out-line="0" />
                ...
                ...
    </Module>
</ControlLoop>
```

Listing 5.7: XCL file of a C4G Control Loop

Notice that the header of the XCL file (line 2) contains a reference to `xcl1.dtd` for its validation during the parsing phase and that `ControlLoop` element has the following attributes:

- `name`, that is the name of the control loop;
- `period`, that is the PC-C4G communication cycle time, expressed in seconds (0.002 or 0.001);
- `timing-source`, whose value is "External" since the control loop step timing is asynchronously given by an external event, i.e. the arrival of a communication packet from C4G;
- `lead-perc`, that is the amount, in percentage, of the control loop sample time by which the wait of the next external synchronization signal will be anticipated respect to the nominal sample time.

5.7 OrchestraC4GLibrary testsuite

In `/usr/local/C4Gopen/OrchestraC4GLibrary/examples` directory you can find some ready-to-use control loops showing the usage of *OrchestraC4GLibrary*.

In addition, you can also monitor what is happening during the execution of a control loop by means of OrchestraHMI [11]. In this case, you need to properly configure an Orchestra Control Network and launch an OrchestraNameServer: for more details, please refer to [10].

All the examples have been tested on a Comau NS-16 robot, with the PC-C4G communication cycle time set to 2 ms.

5.7.1 Mode1

Description

Aim of this test is controlling ONLY the axis 6 of the robot in *Mode 1*, that is supplying axis 6 with target position, target velocity, feedforward velocity and feedforward current.

The test current value is equal to 5% of axis 6 current limit: a 1 Hz square wave between -testCurrent and +testCurrent is given, so that you will see axis 6 going backwards and forwards respect to the initial position. Target position is constantly set to the actual position, while target and feedforward velocity are constantly set to '0'.

Notice that `Generator` module, whose aim is computing all values requested by *Mode 1* and sending them to `Actuator`, is characterized by a parameter named `currentCoeff`, defined in XMP file as follows:

- initial value: 0.05, i.e. 5% of axis 6 current limit;
- lower limit: 0.01, i.e. 1% of axis 6 current limit;
- upper limit: 0.1, i.e. 10% of axis 6 current limit.

Launching the test

Enter the `/usr/local/C4Gopen/OrchestraC4GLibrary/examples/Mode1` directory and type the following command as `root` or with `sudo`:

```
OrchestraCore Loop.xcl
```

where `Loop.xcl` is the XML configuration file describing the topology of the overall control loop.

Enable the open mode ONLY for axis 6 and start C4G in mode 0; when the communication starts, "Drive ON" the robot and set *Mode 1* ONLY for axis 6 via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server, i.e. Orchestra control loop, will close.

5.7.2 Mode5

Description

Aim of the test is controlling the robot in *Mode 5*, that is supplying every open-enabled axis with relative target position and target velocity according to a sinusoidal movement, whose amplitude and frequency are chosen by the user, starting from the actual position of the robot (we suggest you to start from the calibration position). The test will last until a *Mode 504* request has been scheduled on C4G via PDL2.

C4G starts in *Mode 0* and the robot is driven off: PC begins to generate the trajectory when the "Drive ON" signal is given and C4G enters *Mode 5*.

Notice that `Generator` module, whose aim is computing all values requested by *Mode 5* and sending them to `Actuator`, is characterized by the following parameters:

- `frequency`, i.e. sinusoidal movement frequency in hertz:
  - initial value: 0.25;
  - lower limit: 0.1;
  - upper limit: 1.0;
  - tunable: `yes`, i.e. you can modify the value at runtime by means of OrchestraHMI;
- `amplitude`, i.e. sinusoidal movement amplitude in degrees:
  - initial value: 5;
  - lower limit: 1;
  - upper limit: 10;
  - tunable: `yes`, i.e. you can modify the value at runtime by means of OrchestraHMI;

- `enabledAxes`, i.e. array of open-enabled axes, whose initial value is [1 1 1 1 1 1] (all axes are open: 1 means 'open', 0 'close').

Launching the test

Enter the `/usr/local/C4Gopen/OrchestraC4GLibrary/examples/Mode5` directory and type the following command as `root` or with `sudo`:

```
OrchestraCore Loop.xcl
```

where `Loop.xcl` is the XML configuration file describing the topology of the overall control loop.

Enable the open mode for the axes corresponding to the `enabledAxes` parameter and start C4G in mode 0; when the communication starts, "Drive ON" the robot and set *Mode 5* on C4G via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server, i.e. Orchestra control loop, will close.

5.7.3 Mode8

Description

Aim of this test is controlling ONLY the axis 1 of the robot in *Mode 8*, that is supplying axis 1 with feedforward velocity.

Feedforward velocity is a 1 Hz sinusoidal signal, whose amplitude is equal to 0.0005 (delta motor turns).

Notice that `Generator` module, whose aim is computing all values requested by *Mode 8* and sending them to `Actuator`, is characterized by the following parameters:

- `speedCoeff`, i.e. sinusoidal signal amplitude in delta rpm:
  - initial value: 0.0005;
  - lower limit: 0.0, i.e. no signal;
  - upper limit: 0.002;
  - tunable: `yes`, i.e. you can modify the value at runtime by means of OrchestraHMI;

- `frequency`, i.e. sinusoidal signal frequency in hertz:
  - initial value: 1.0;
  - lower limit: 0.0;
  - upper limit: 30.0;
  - tunable: `yes`, i.e. you can modify the value at runtime by means of OrchestraHMI.

Launching the test

Enter the `/usr/local/C4Gopen/OrchestraC4GLibrary/examples/Mode8` directory and type the following command as `root` or with `sudo`:

```
OrchestraCore Loop.xcl
```

where `Loop.xcl` is the XML configuration file describing the topology of the overall control loop.

Enable the open mode ONLY for axis 1 and start C4G in mode 0; when the communication starts, "Drive ON" the robot and set *Mode 8* ONLY for axis 6 via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server, i.e. Orchestra control loop, will close.

5.7.4 Mode9

Description

Aim of this test is controlling ONLY the axis 1 of the robot in *Mode 9*, that is supplying axis 1 with delta current.

Delta current is a 1 Hz sinusoidal signal, whose amplitude is equal to 25% of axis 1 current limit.

Notice that `Generator` module, whose aim is computing all values requested by *Mode 9* and sending them to `Actuator`, is characterized by the following parameters:

- `currentCoeff`, i.e. sinusoidal signal amplitude in percentage of axis 1 current limit:
  - initial value: 0.25, i.e. 25% of axis 1 current limit;

  - lower limit: 0.0, i.e. no signal;
  - upper limit: 1.0, i.e. current limit itself;
  - tunable: `yes`, i.e. you can modify the value at runtime by means of OrchestraHMI;

- `frequency`, i.e. sinusoidal signal frequency in hertz:
  - initial value: 1.0;
  - lower limit: 0.0;
  - upper limit: 30.0;
  - tunable: `yes`, i.e. you can modify the value at runtime by means of OrchestraHMI.

Launching the test

Enter the `/usr/local/C4Gopen/OrchestraC4GLibrary/examples/Mode9` directory and type the following command as `root` or with `sudo`:

```
OrchestraCore Loop.xcl
```

where `Loop.xcl` is the XML configuration file describing the topology of the overall control loop.

Enable the open mode ONLY for axis 1 and start C4G in mode 0; when the communication starts, "Drive ON" the robot and set *Mode 9* ONLY for axis 6 via PDL2.

Stopping the test

"Drive OFF" the robot and schedule a *Mode 504* request on C4G via PDL2, so that the server, i.e. Orchestra control loop, will close.

Appendix A

RTnet-compliant chipsets

Table A.1: RTnet-compliant network card chipsets (10/100 Mbps)

RTnet driver name	Vendor ID	Device ID	Manufacturer	Chip number
rt_8139too	0x10ec	0x8139	RealTek	RTL8139/8139C/8139C+
	0x10ec	0x8138	RealTek	RTL8139 (B/C)
	0x4033	0x1360	Delta Networks	RTL8139
	0x1186	0x1300	D-Link	RTL8139
	0x1186	0x1340	D-Link	DFE-690TXD
	0x13d1	0xab06	AboCom	RTL8139
	0x1259	0xa117	Allied Tsyn	RTL81xx
rt_eepro100	0x8086	0x1229	Intel	82557/8/9/0/1
	0x8086	0x1209	Intel	8255xER/IT
	0x8086	0x1029	Intel	82559
	0x8086	0x1030	Intel	825593
	0x8086	0x1031	Intel	82801CAM
	0x8086	0x1032	Intel	82801CAM
	0x8086	0x1033	Intel	82801CAM
	0x8086	0x1034	Intel	82801CAM
	0x8086	0x1035	Intel	82562EH
	0x8086	0x1036	Intel	82562EH
	0x8086	0x1037	Intel	82801CAM
	0x8086	0x1038	Intel	82801CAM
	0x8086	0x1039	Intel	82801DB
	0x8086	0x103A	Intel	82801DB
	0x8086	0x103B	Intel	82801DB
	0x8086	0x103C	Intel	82801DB
	0x8086	0x103D	Intel	82801DB
	0x8086	0x103E	Intel	82801DB
	0x8086	0x1227	Intel	82865g
	0x8086	0x1228	Intel	82556
	0x8086	0x2449	Intel	82559ER
	0x8086	0x2459	Intel	82801E
	0x8086	0x245D	Intel	82801E
	0x8086	0x5200	Intel	EE PRO/100
	0x8086	0x5201	Intel	EE PRO/100
rt_natsemi	0x100b	0x0001	National Semiconductors	DP83810
	0x100b	0x0020	National Semiconductors	DP83815/16

RTnet driver name	Vendor ID	Device ID	Manufacturer	Chip number
rt_tulip	0x1011	0x0002	DEC	DC21040
	0x1011	0x0014	DEC	DC21041
	0x1011	0x0009	DEC	DC21140
	0x1011	0x0019	DEC	DC21142/3
	0x11ad	0x0002	Lite-On Communications	NGMC169B
	0x11ad	0xc115	Lite-On Communications	LC82C115
	0x10d9	0x0512	Macronix	MX98713
	0x10d9	0x0531	Macronix	MX98715/25
	0x1317	0x0981	ADMtek	AN981
	0x1317	0x0985	ADMtek	AN983B
	0x1317	0x1985	ADMtek	AN985
	0x1317	0x9511	ADMtek	ADM9511
	0x104a	0x0981	STMicroelectronics	21x4x
	0x104a	0x2774	STMicroelectronics	21x4x
	0x11f6	0x9881	Powermatic	TXA9881
	0x1282	0x9102	Davicom	DM9102/A/AF
	0x1113	0x1216	Accton	EN5251BE
	0x1113	0x1217	Accton	EN2242
	0x1113	0x9511	Accton	SMC EN5251BE
rt_3c59x (experimental)	0x10b7	0x5900	3Com	3C590
	0x10b7	0x5920	3Com	3C592
	0x10b7	0x5970	3Com	3C597
	0x10b7	0x5950	3Com	3C595
	0x10b7	0x5951	3Com	3C595
	0x10b7	0x5952	3Com	3C595
	0x10b7	0x9000	3Com	3C900
	0x10b7	0x9001	3Com	3C900
	0x10b7	0x9004	3Com	3C900B-TPO
	0x10b7	0x9005	3Com	3C900B-Combo
	0x10b7	0x9006	3Com	3C900B-TPC
	0x10b7	0x900a	3Com	3C900B-FL
	0x10b7	0x9050	3Com	3C905
	0x10b7	0x9051	3Com	3C905
	0x10b7	0x9055	3Com	3C905B
	0x10b7	0x9058	3Com	3C905B
	0x10b7	0x905a	3Com	3C905B-FX
	0x10b7	0x9200	3Com	3C905C-TX/TX-M
	0x10b7	0x9800	3Com	3C980-TX
	0x10b7	0x9805	3Com	3C980-C
	0x10b7	0x7646	3Com	3CSOHO100-TX
	0x10b7	0x5055	3Com	3C555
	0x10b7	0x6055	3Com	3C556
	0x10b7	0x6056	3Com	3C556B
	0x10b7	0x5b57	3Com	3C595
	0x10b7	0x5057	3Com	3C575
	0x10b7	0x5157	3Com	3CCFE575BT
	0x10b7	0x5257	3Com	3CCFE575CT
	0x10b7	0x6560	3Com	3CCFE656
	0x10b7	0x6562	3Com	3CCFEM656B
	0x10b7	0x6564	3Com	3CXFEM656C
	0x10b7	0x4500	3Com	3C450

RTnet driver name	Vendor ID	Device ID	Manufacturer	Chip number
rt_via-rhine	0x1106	0x3043	VIA	VT86C100A
	0x1106	0x3065	VIA	VT6102
	0x1106	0x3106	VIA	VT6105
	0x1106	0x3053	VIA	VT6105M

Bibliography

[1] *C4G Open, System Software Rel. 3.1x Instruction Handbook*, Comau S.p.A. Robotics & Final Assembly, 2008.

[2] RTAI Website - http://www.rtai.org

[3] RTnet Website - http://www.rtnet.org

[4] Ubuntu Website - http://www.ubuntu.com

[5] The Linux Kernel Archives - http://www.kernel.org

[6] The GNU Project - http://www.gnu.org

[7] Bucher, Mannori, Netter, *RTAI-Lab tutorial: Scilab, Comedi, and real-time control*, May 2007.

[8] Orchestra Control Engine - http://www.orchestracontrol.com

[9] *Orchestra_ v1.5-1 Installation Procedure v1.0.0*, Sintesi S.C.p.A., 2008

[10] *OrchestraCore User's Guide v1.4.0*, Sintesi S.C.p.A., 2008

[11] *OrchestraHMI User's Guide v1.4.0*, Sintesi S.C.p.A., 2008