FAVERGES 14/06/2012

LOW LEVEL ROBOT CONTROL INTERFACE

LLI S7.4

STÄUBLI FAVERGES
Place Robert Stäubli - BP 70
74210 FAVERGES FRANCE
Tél 33 (0)4 50 65 60 60- Fax 33(0)4 50 65 60 70
www.staubli.com - ISO 9001

S.C.A au Capital 7 718 400 €
R.C.S Annecy B 325 720 720
N° Identification T.V.A FR 80 325 720 720
NAF 295 G

LOW LEVEL ROBOT CONTROL 7.4

TABLE OF CONTENTS

LOW LEVEL ROBOT CONTROL 7.4

1. PURPOSE OF THE DOCUMENT

The documents describes how to install, configure and create robot control application with Stäubli Low Level Robot Control.

In this document, LLI stands for Low Level Robot Control Interface.

2. LLI RELEASE NOTES

LLI s7.4 is based on a standard SRC s7.4 system.

Please refer to the SRC s7.4 release notes.

3. LLI INSTALLATION AND CONFIGURATION

3.1. Prerequisites

To install LLI on a controller, you will need:

- A computer running Microsoft Windows XP SP3 or later.
- A Staübli VAL3 / LLI DVD
- A CS8/CS8C controller

Before installing LLI on your controller, we advise you to connect a terminal to the serial port of the controller. This is not required to perform a successful installation, however this will help you in case of problems.

You can either use your favorite V100-emulation software or install *TeraTerm* (on the DVD).

Connect the J203 serial port of the controller to one of the serial ports of your computer, and configure your terminal software to:

- 115 200 bauds,
- 8 bits data,
- no parity,
- 1 stop bit,
- no flow control.

3.2. Installing LLI on the controller internal memory

The Stäubli LLI installation DVD is a typical VAL3 DVD with additional LLI contents (SDK, sample program and documentation).

The installation of LLI on a CS8 / CS8C controller requires the following steps:

1. Install VAL3 system by using a script that will send software files to the controller.
    - Open a command-line window (cmd.exe)
    - Navigate to the *System* directory on the DVD.
    - Call the *update.bat* script with the following parameters:


```
update.bat IP_ADRESS   PROFILE   PASSWORD
```

Where:

- *IP_ADRESS* is the ip address of the controller.
- *PROFILE* is one of the user profiles defined on the controller (use *maintenance* by default).
- *PASSWORD* is the password of the profile (*spec_cal* is the default *maintenance* password).

- The *update.bat* script is a very basic FTP script that does not check for update errors. Make sure that the update is OK before going to the next step.

2. Enable the LLI license on the controller. You can either use the SRS option manager or add the following line in file /usr/config/controller.cfx:

```
<String name="lli" value="password" />
```

Where PASSWORD is the controller-specific 16 digits license key that Stäubli provided you.

3. Replace the standard VAL3 main software (/sys/cs8.out) by your own LLI-based software.
4. Reboot the CS8.

Remarks:

- The *update.bat* script is a very basic FTP script that does not check for update errors. Make sure that the update is OK before restarting the controller.

3.3. Configuring a CS8 back to VAL3

To replace the LLI software on a CS8 by the standard VAL3 environment, re-run the update.bat script from the DVD. Refer to §3.1, step 1.

Please note that there is no need to remove the LLI license line from /usr/config/controller.cfx.

3.4. Installation LLI on a USB Key

Warning: the CS8 USB driver has limitations and does not support newer USB keys. To make sure your USB key is supported, plug it into the robot controller running a VAL3 system and check that it appears in the user interface (open VAL3 application).

To make the USB key bootable by the robot controller:

- create a ‘boot’ directory at the root of the USB key: USB\boot
- copy the content of LLI/flash on CdRom into the boot directory of the USB key.
- Modify the USB\boot\usr\configs\network.cfx file of the USB key to specify the desired IP address for the controller.

Only the ‘User’ USB ports can be used. The ‘Sys’ USB ports are reserved for BIOS usage.

3.5. Configuration

3.5.1. Setting cycle time (CS8C only)

The CS8C cycle type is 4 ms by default.

This value can be modified to suit your application's needs. This value shall be a multiple of 0.2 ms. The cycle process execution length must be lower than the cycle time. Thus, be careful with CPU load when using low cycle time.

The cycle time can be modified by adding the following line in the /usr/config/controller.cfx file:

```
<Float name="lliCycleTime" value="0.001" />
```

Value is the cycle time expressed in seconds.

Please note that Stäubli can provide a 933Mhz CPU board on demand. This CPU board provides more processing power than the standard 400Mhz CPU board.

4. BUILDING YOUR APPLICATION WITH LLI

The LLI software consists of two parts:

- A set of robotic software that runs on the CS8 controller. This software is based on standard Stäubli VAL3 system software. This software interacts with the controller hardware and provides basic robotics functions.
- A software SDK that provides a programming interface to the robot functions. This interface is a C library.

The LLI SDK is located in the LLI directory of the DVD. The contents are:

Directory	Contents
LLI/inc	LLI SDK header file
LLI/bin-win32/debug	Windows 32 bits version of the LLI SDK (debug)
LLI/bin-win32/release	Windows 32 bits version of the LLI SDK (release)
LLI/bin-vxw5.5.1/debug	VxWorks 5.5.1 version of the LLI SDK (debug)
LLI/bin-vxw5.5.1/release	VxWorks 5.5.1 version of the LLI SDK (release)
LLI/sample/src	Source code of the sample application
LLI/sample/Visual Studio 2010	Visual Studio 2010 solution for the sample application.
LLI/sample/Tornado2.2	VxWorks 5.5.1 / Tornado 2.2.1 project for the sample application.

4.1. Windows

Building a LLI application for Microsoft Windows requires Visual Studio 2010.

The sample application is located in the **LLI\sample\Visual Studio 2010** directory on the DVD.

4.2. VxWorks 5.5.1

To develop your own C program using the LLI library, you need first to create a Tornado 2.2.1 project and configure it adequately.

Using the Tornado Wizard for project creation, you need to select “create downloadable application modules”, then “PENTIUMgnu” as toolchain.

We provide sample Tornado project and makefile file in the **LLI\sample\Tornado2.2** directory of the DVD.

4.3. Auto start

To have your C application automatically executed on the controller, you need to have as entry point the C function “void CS8_start(const char* x_sys, const char* x_usr, const char* x_log)”, and place your binary on the flash as /sys/cs8.out, or on a USB key as /boot/sys/cs8.out.

The x_sys, x_usr and x_log parameters of CS8_start() are set to NULL by default, or to the path on USB when a directory /boot/sys, /boot/usr or /boot/log has been found on the USB device.

4.4. Boot failure recovery

In case of failure at boot time when testing your VxWorks application, you may need to stop the boot sequence and cancel the execution of your application. This can be done with the VxWorks prompt at the beginning of the VxWorks System boot, by hitting the 's' key regularly (every 0.5s) as soon as the BIOS boot is finished. You can then edit the bootline.dat file (‘c’ command) and remove the startup binary or script.

5. LLI API REFERENCE

5.1. Construction / destruction

The LLI subsystem shall be initialized before use. The API provides two initialization functions: one initializes the complete subsystem, the other initializes only the kinematics functions.

5.1.1. LLI_construct

This constructor enables both the robot control API and the kinematics API.

This constructors requires a pointer to a callback function (CyclicFunction).

It also takes optional paths to the /sys, /usr and /log directories (can be NULL to use default values).

This function ensure that the robot is in a stable safe state after construction.

If an error occurs in the constructor, LLI_init cannot be performed properly, and only the following functions can be called without error: LLI_state.

Construction errors are memorized in a file (see Errors management).

5.1.2. LLI_constructKinOnly

This constructor gives access to kinematics programming interface only.

Construction errors are memorized in a file (see Errors management).

5.1.3. LLI_destroy

- All dynamic memory is freed.
- Robot is in a stable safe state after destruction.
- It is possible to reconstruct the robot interface after proper destruction.

5.2. Generic programming interface

The following functions can be used whatever the constructor previously called.

5.2.1. Error management (LLI_plugEventFunc)

- The controller outputs the errors on its serial port COM1, (115200, 8 bits, no parity, 1 stop bit, VT100 emulation).
- All the errors are also logged in file named error.log on the /log flash disk partition.

The errors can be also redirected to the user using the function **LLI_plugEventFunc**:

- This function plugs a user function to the event manager, at each event received the manager calls the user function.
- The parameter of the user function is pointer on the string which contains the event description
- It's possible to plug the user event function just before calling LLI_Init

5.2.2. Miscellaneous (LLI_ioctrl)

The function **LLI_ioctrl** sets or gets the following parameters:

Parameter	Set	Get
Software version	NA	Return the staubli library software version.
Robot type	NA	Return the current robot type.
Joint number	NA	Return the number of robot joints.
Robot user mark position	Set and store a new value of the user mark	Get the value of the user mark position. In factory setting, this position corresponds to the physical mark on the joints.
Robot joint range	NA	Get the arm joints limits: maximum and minimum values. Warning: the returned values are the cell limits.

5.3. Robot control programming interface

The following functions can be used when the constructor LLI_construct has been previously called.

5.3.1. General points

- The LLI interface functions should be accessed from 2 different contexts :
  - An asynchronous task with low priority.
  - A synchronous task, with higher priority than the asynchronous task.

The following table summarizes in which context the functions may be called.

Synchronous	Asynchronous
	LLI_construct LLI_destroy LLI_init
LLI_enable LLI_disable	LLI_enable LLI_disable
LLI_state LLI_ioctrl[1]	LLI_state LLI_home LLI_ioctrl LLI_getDinId LLI_getDoutId LLI_readDio LLI_writeDout
LLI_get LLI_set	

5.3.2. LLI_init

- The hardware is initialized : Sercos (CS8) / Starc (CS8C) communication with drives
- The robot is calibrated.
- Init cannot be called in synchronous context.
- Init must be called only one time after construction.
- The following functions return an error if the system was not initialized properly before they are called: LLI_enable, LLI_disable, LLI_home.

[1] For some robot parameters only.

5.3.3. LLI_enable

- *Enable* is a non-blocking function: the enable procedure is not handled in the context of the call to *Enable*, but in the *Set*/*Get* context. The enable procedure is not necessarily finished when the function returns.
- When calling *Enable*, an « enable request » is sent. If the system is disabled, and there is no reason to prevent enabling, the enabling procedure will be started at the next call of the Set function. Otherwise, the request is definitely cancelled, and an error is raised.
- The enabling procedure can be aborted at any moment, possibly without any error message (for example, by LLI_disable).
- The enabling procedure can be checked with the LLI_state function.
- In any case, the enabling procedure has a finite duration.

5.3.4. LLI_disable

- Disable is a non-blocking function: the disable procedure is not handled in the context of the call to Disable, but in the Set/Get context. The disable procedure is not necessarily finished when the function returns.
- When calling Disable, a « disable request » is sent. The disabling procedure will be started at the next call of the Set function.
- The disable procedure definitely aborts any enabling procedure that was going on, without any error message.
- The enabling procedure can be checked with the LLI_state function.
- If the disabling procedure lasts more than a fixed timeout, a timeout error is raised.

5.3.5. LLI_get

- *LLI_get* shall be called synchronously before LLI_set at each cycle, as soon as *CyclicFunction* is running.
- *LLI_get* updates the robot joint feedback, as well as all published statuses (see LLI_state).

5.3.6. LLI_set

- *LLI_set* defines the commands for the current cycle (position command, velocity command, torque feed forward). For more information, see also Control modes.
- Invalid commands (position, velocity, delta position or delta velocity out of range) are never sent to the drives.
- *LLI_set* must be called synchronously after LLI_get at each cycle, as soon as *CyclicFunction* is running.
- *LLI_set* updates all the published statuses.

5.3.7. LLI_state

- State gives the current state, which contains the following information :
    - initialized / not initialized / no license

  - enabled / enabling / disabling / disabled
  - calibrated / not calibrated / calibrating
  - settled / not settled (i.e. robot position stabilized)
  - emergency stop / no emergency stop
- *State* can be called either from synchronous or from asynchronous context.

5.3.8. LLI_home

- Calibrate the robot using:
  - calibration information stored on the flash disk
  - calibration information maintained by the calibration board
- After calling this function, commanded and measured positions are absolute.
- This function is automatically called at the end of LLI_init

5.3.9. LLI_ioctrl

- This function sets or gets the following parameters:

Parameter	Set	Get
(All parameters previously defines in "Generic programming interface" are available).		
Robot absolute position recovery	Restore correct calibration parameters in the calibration board according to the user mark position passed to the function. The robot must be on the user mark position (±½ motor turn on each joint). The calibration board parameters may be corrupted after calibration board was disconnected from motors or from its battery.	NA
Robot user mark position	Set and store a new value of the user mark	Get the value of the user mark position. In factory setting, this position corresponds to the physical mark on the joints.
TorqueMode	Switch from the standard position control to the torque control mode.	NA
PosVelMode	Switch from the torque control mode to the standard position control.	NA

- Ioctrl calibration functions cannot be called in synchronous context.

5.3.10. CyclicFunction

The cyclic function is a callback function provided at initialization, to the LLI_construct function.

This function is perdiodically called by the LLI subsystem, at each cycle, in the drive interrupt handler (Sercos ISR for CS8, Starc ISR for CS8C).

It is called at the end of the ISR, after commands were sent to the drives.

The execution time of this function shall be as short as possible and must be valid in the context of an interrupt handler (i.e. it shall not wait on semaphores).

When this function returns an error, the drives are disabled as soon as possible by the LLI subsystem.

Typically, this function will give a binary semaphore. The synchronous task will take this semaphore and call the LLI_get and LLI_set function. This task is a high priority task, priority of this task must be the one returned by the LLI_ioctrl command “cyclic task priority”.

5.3.11. IO functions

- **LLI_getDinId** returns an ID on a Digital Input, the entry parameter is the name of Din (see file /usr/cell.cf)
- **LLI_getDoutId** returns an ID on a Digital Output, the entry parameter is the name of Dout
- **LLI_readDio** returns the value of the IO (input or output): 0, 1 or undefined
- **LLI_writeDout** sets the output to 1 or 0

5.3.12. Control modes

The velocity command is supposed to be the instantaneous derivative of the position command. Position and velocity commands are checked before being sent to the drives at each cycle time (position range, velocity range,…). Then, they are micro-interpolated in the drives, which enable the servo loop to run at higher rates.

Each drive contains a position, a velocity loop, and a current loop. External torque feed-forward may be added to improve the control, especially to compensate for the joint couplings.

When torque mode is activated, the position and velocity control loops are disabled. Then, the torque feed-forward is used as the torque command. In this mode, the measured position and velocity should be fed back as commands. They are not used in the control loops, but they are still checked by the system, which provides a minimal safety level.

5.4. Kinematics programming interface

The following functions can be used when the constructor LLI_construct or LLI_constructKinOnly has been previously called.

5.4.1. LLI_reverseKine

This function returns the joint position corresponding to a Cartesian position, a robot configuration, starting joint position and joint limits.

The input Cartesian position is defines as the position of the flange of the robot referenced to the base of the robot.

The function returns the result of the computation.

5.4.2. LLI_forwardKine

This function returns the robot flange Cartesian position and the robot configuration corresponding to a joint position.

5.5. LLI Recording API

LLI provides an API to control the CS8 recorder.

The recorder can be used to record many internal variables.

5.5.1. Configuring the recorder

The **/usr/recorder/records.cfx** file configures the recorder:

- Path to file where the records will be stored
- Record max duration
- Record frequency, in percent of the cycle frequency: 100% means that a record is stored at every cycle, 50% means that a record is stored every other cycle, and so on.
- List of CS8 variables to be recorded.

5.5.2. LLI_recInit

Initializes a recorder.

Can only be started in an asynchronous task.

5.5.3. LLI_recStart

Starts the recording of data.

Can be either started in a synchronous or an asynchronous task.

5.5.4. LLI_recStop

Stops the recording of data.

Can be either started in a synchronous or an asynchronous task.

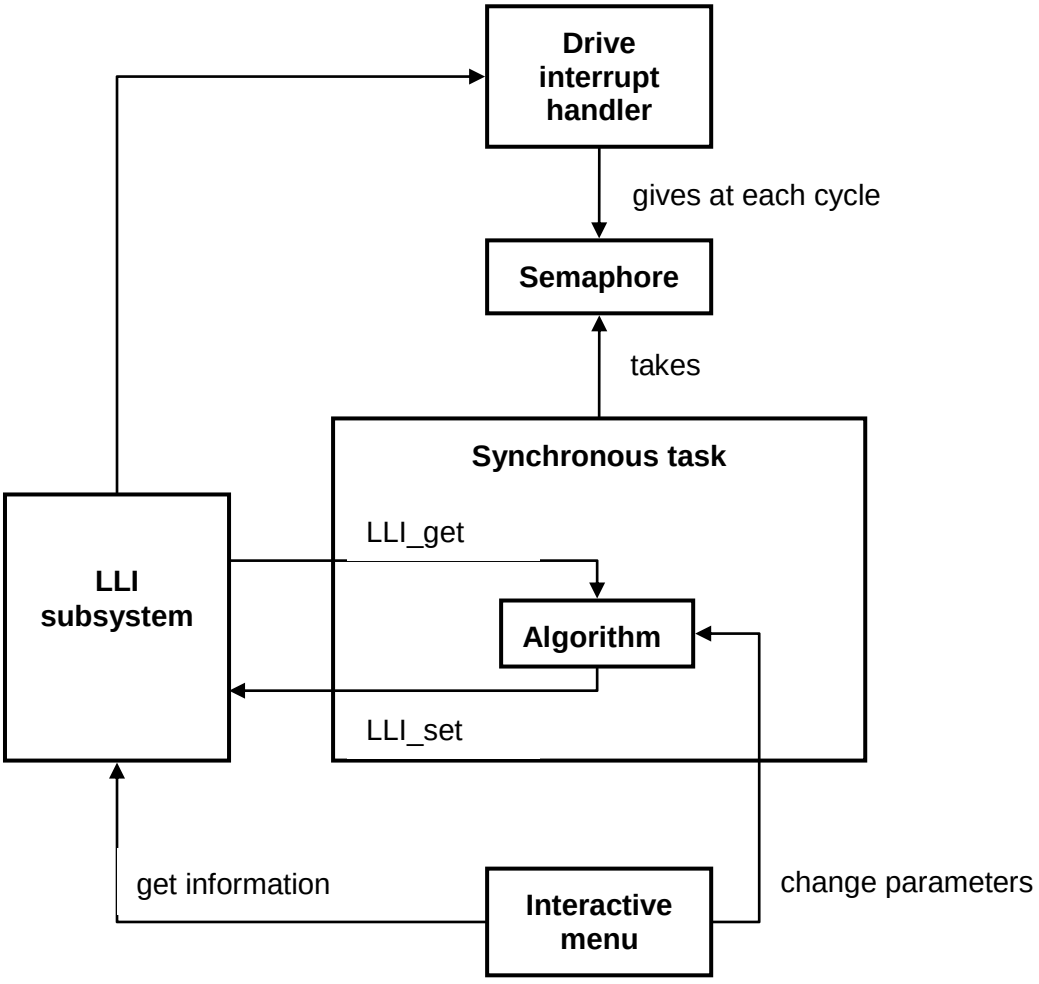
5.5.5. LLI_recStore

Stores the records on a file on the controller.

Can only be started in an asynchronous task.

6. SAMPLE PROGRAM

6.1. Sample program overview



6.2. Sample program detailed description

6.2.1. Drive interrupt handler

The LLI subsystem calls this handler within the context of the drive interrupt. The handler gives a semaphore at each interrupt. Two methods are provided:

- configuration : define which semaphore the IRS handler should give
- a function pointer (LLI_PlugIn) to be linked to LLI_Robot

6.2.2. Algorithm

The sample program provides two algorithms:

- Algo1: A very simple movement generator. Should be used when the robot is in standard position control mode. It provides the following functions:
  - initialize algorithm
  - compute the new setpoint at each cycle time
  - start a movement on a joint
- Algo2 : A very simple proportional position controller. Should be used when the robot is in torque control mode. It provides the following functions:
  - initialize algorithm
  - compute the new setpoint at each cycle time
  - set/get gains

Warning: in torque control mode, invalid gains may lead to robot instability, unsafe robot movements, damage to the robot, and injury to the user.

6.2.3. Synchronous task

This task is pending on the synchronization semaphore given by the drive ISR handler. At each cycle time, it reads the LLI feedbacks, and sets the new command. The command is compute by algo1 or algo2.

The following methods are provided

- start the synchronous task
- stop the synchronous task
- get current commands
- get current feedback
- set/get the current algorithm (1 or 2)

Warning: Please note that synchronous task stack size requirements changed between LLI s6 and LLI s7. Thus, when starting the synchronous task, make sure to specify a sufficiently large stack size (25KB for instance) to prevent stack overflow.

6.2.4. Interactive menu

It is a text user interface (menus) which enables to access:

- each asynchronous LLI function,
- synchronous function parameters (commands and feedbacks)
- parameters for algo1 (joint to be moved)
- parameters for algo2 (gains)

6.2.5. Test application

This is the main task of the test. It manages the following aspects:

- construct application, link objects together, and initialize the system.
- run user interface until exit
- manage application end, stop tasks, and destroy all objects

6.3. Sample program implementation

This sample program is implemented in C.

File	Description
Systemfunctions.h	Defines the interface for using « OS interface » and « Semaphore », independently of the OS (Windows or VxWorks)
WinFunctions.c	Implementation of the functions defined in Systemfunctions.h specific for Windows. This file is compiled only under Windows.
VxFunctions.c	Implementation of the functions defined in Systemfunctions.h specific for VxWorks. This file is compiled only under VxWorks.
Isr.h	Interface of Sercos ISR handler
Isr.c	Implementation of Sercos ISR handler
Synchro.h	Interface of the synchronous task
Synchro.c	Implementation of the synchronous task
Algo1.h	Interface of algo1
Algo1.c	Implementation of algo1
Algo2.h	Interface of algo2
Algo2.c	Implementation of algo2
_test.h, text.h	Header for test applications
Test.c	Test application and interactive menu for robot control
TestKin.c	Test application and interactive menu for robot kinematics

7. LIBRARY HEADER

```
// ----------------------------------------------------------------
// Types and structures
// ----------------------------------------------------------------
// LLI_RobotType type
#define     LLI_S_STR_ROBOT_TYPE    32
typedef char LLI_RobotType[LLI_S_STR_ROBOT_TYPE+1];
// LLI_SoftwareVersion type
#define     LLI_S_STR_SOFT_VERSION  32
typedef char LLI_SoftwareVersion[LLI_S_STR_SOFT_VERSION+1];

// Return code
typedef enum _LLI_Status
{
      LLI_OK,
      LLI_ERROR
} LLI_Status;

// Init state
typedef enum _LLI_InitState
{
      LLI_NO_LICENCE,
      LLI_NOT_INITIALIZED,
      LLI_INITIALIZED
} LLI_InitState;

// Enable state
typedef enum _LLI_EnableState
{
      LLI_DISABLED,
      LLI_ENABLING,
      LLI_ENABLED,
      LLI_DISABLING,
} LLI_EnableState;
```

```
// Calibration state
typedef enum _LLI_CalibState
{
      LLI_CALIBRATED,
      LLI_NOT_CALIBRATED,
} LLI_CalibState;

// Settle state
typedef enum _LLI_SettleState
{
      LLI_NOT_SETTLED,
      LLI_SETTLED
} LLI_SettleState;

// Estop State
typedef enum _LLI_EstopState
{
      LLI_ESTOP,
      LLI_NO_ESTOP,
} LLI_EstopState;

// LLI State (function LLI_state)
typedef struct _LLI_State
{
      LLI_InitState           m_initState;
      LLI_EnableState         m_enableState;
      LLI_CalibState          m_calibrateState;
      LLI_SettleState         m_settleState;
      LLI_EstopState          m_estopState;
} LLI_State;

// Ioctrl code
typedef enum _LLI_IoctrlCode
{
      LLI_GET_SOFTWARE_VERSION,
      LLI_GET_ROBOT_TYPE,
      LLI_GET_JNT_NUMBER,
```

```
	LLI_GET_CYCLE_TIME,
	LLI_GET_SYNCHRO_TASK_PRIORITY,
	LLI_SET_ABS_JNT_POS,
	LLI_RECOVER_ABS_JNT_POS,
	LLI_GET_USER_MARKS,
	LLI_SET_USER_MARKS,
	LLI_SET_TORQUE_MODE,
	LLI_SET_POS_VEL_MODE,
	LLI_GET_JNT_RANGE
} LLI_IoctrlCode;

// Io type
typedef enum _LLI_IoType
{
	IO_D_INPUT,
	IO_D_OUTPUT,
	IO_A_INPUT,
	IO_A_OUTPUT,
	IO_UNDEFINED
} LLI_IoType;

// Dio status (function LLI_readDio)
typedef enum _LLI_DioStatus
{
	DIO_ONE,
	DIO_ZERO,
	DIO_UNDEFINED
} LLI_DioStatus;

// Dout value (function LLI_writeDout)
typedef enum _LLI_DoutValue
{
	DOUT_ONE,
	DOUT_ZERO
} LLI_DoutValue;

// Io Id
```

```
typedef struct _LLI_IoId
{
      LLI_IoType        m_type;           // UNDEFINED if Io not found
      void*             m_pIo;            // NULL if Io not found
} LLI_IoId;

// LLI_PlugIn is a pointer on a function "void function(void)"
typedef void (*LLI_PlugIn)(void);

// LLI_EventFunc is a pointer on a function "void function(char *)"
typedef void (*LLI_EventFunc)(char *);

// Robot ID
struct _LLI_Robot;
typedef struct _LLI_Robot LLI_Robot;
typedef LLI_Robot* LLI_RobotId;

// LLI command (at the joint level, after motor reduction)
typedef struct _LLI_JointCmd
{
      double      m_pos;            /* rad or m */
      double      m_vel;            /* rad/s or m/s */
      double      m_torqueFfw;      /* N.m */
} LLI_JointCmd;

// LLI feedbacks (at the joint level, after motor reduction)
typedef struct _LLI_JointFbk
{
      double      m_pos;            /* rad or m */
      double      m_vel;            /* rad/s or m/s */
      double      m_posErr;         /* rad or m */
      double      m_torque;         /* N.m */
} LLI_JointFbk;

// Cartesian position
// nx ox    ax    px
// ny oy    ay    py
```

```
// nz oz    az    pz
//  0  0     0     1
typedef struct _LLI_Frame
{
      /*rot*/
      double m_nx;
      double m_ny;
      double m_nz;
      double m_ox;
      double m_oy;
      double m_oz;
      double m_ax;
      double m_ay;
      double m_az;
      /*pos*/
      double m_px;
      double m_py;
      double m_pz;
} LLI_Frame;

// Joint position
typedef double LLI_Joint;

// Joint range
typedef struct _LLI_JointRange
{
      LLI_Joint m_min;
      LLI_Joint m_max;
} LLI_JointRange;

// Robot configuration
typedef enum _LLI_ShoulderConfig
{
      LLI_SSAME=0,
      LLI_LEFTY=1,
      LLI_RIGHTY=2,
      LLI_SFREE=3
```

```
} LLI_ShoulderConfig;

typedef enum _LLI_PositiveNegativeConfig
{
      LLI_PNSAME=0,
      LLI_POSITIVE=1,
      LLI_NEGATIVE=2,
      LLI_PNFREE=3
} LLI_PositiveNegativeConfig;

typedef struct _LLI_RxConfig
{
      LLI_ShoulderConfig           m_shoulder;
      LLI_PositiveNegativeConfig   m_elbow;
      LLI_PositiveNegativeConfig   m_wrist;
} LLI_RxConfig;

typedef struct _LLI_ScaraConfig
{
      LLI_ShoulderConfig     m_shoulder;
} LLI_ScaraConfig;

typedef union _LLI_Config
{
      LLI_RxConfig      m_rxConfig;
      LLI_ScaraConfig   m_scaraConfig;
} LLI_Config;

// Inverse kinematics result
typedef enum _LLI_ReversingResult
{
      LLI_REVERSE_OK = 0,
      LLI_NO_CONVERGENCE,
      LLI_OUT_OF_JNT_RANGE,
      LLI_OUT_OF_WORKSPACE,
      LLI_INVALID_CONFIG,
      LLI_INVALID_ORIENTATION,
```

```
      LLI_UNSUPPORTED_KINEMATICS,
      LLI_UNCONSTRAINT_FRAME,
      LLI_INVALID_ERROR_CODE
} LLI_ReversingResult;
```

```
// ------------------------------------------------------------------
// Functions prototypes
// ------------------------------------------------------------------
#ifdef __cplusplus
extern "C" {
#endif // __cplusplus

LLI_RobotId LLI_construct
                  (
                        LLI_PlugIn x_pFunct,
                        const char* x_pathSys,
                        const char* x_pathUsr,
                        const char* x_pathLog
                  );

LLI_RobotId LLI_constructKinOnly
                  (
                        const char* x_pathSys,
                        const char* x_pathUsr,
                        const char* x_pathLog
                  );
```

LOW LEVEL ROBOT CONTROL 7.4

```
LLI_Status  LLI_destroy
                (LLI_RobotId* x_pRobotId);
LLI_Status  LLI_init
                (LLI_RobotId x_robotId);
LLI_Status  LLI_enable
                (LLI_RobotId x_robotId);
LLI_Status  LLI_disable
                (LLI_RobotId x_robotId);
LLI_Status  LLI_get
                (LLI_RobotId x_robotId, LLI_JointFbk x_jntFbk[]);
LLI_Status  LLI_set
                (LLI_RobotId x_robotId, LLI_JointCmd x_jntCmd[]);
LLI_Status  LLI_state
                (LLI_RobotId x_robotId, LLI_State* x_pState);
LLI_Status  LLI_home
                (LLI_RobotId x_robotId);
LLI_Status  LLI_plugEventFunc
                (LLI_RobotId x_robotId, LLI_EventFunc x_pFunc);
LLI_Status  LLI_ioctrl
                (LLI_RobotId x_robotId, int x_code, void* x_pIoctrlData);
LLI_Status  LLI_getDinId
                (LLI_RobotId x_robotId, char* x_pDioStr, LLI_IoId* x_pDinId);
LLI_Status  LLI_getDoutId
                (LLI_RobotId x_robotId, char* x_pDioStr, LLI_IoId* x_pDoutId);
LLI_Status  LLI_readDio
                (LLI_IoId* x_pDioId, LLI_DioStatus* x_pDioStatus);
LLI_Status  LLI_writeDout
                (LLI_IoId* x_pDoutId, LLI_DoutValue x_doutValue);
```

```
LLI_Status LLI_reverseKin
                  (
                        LLI_RobotId x_robotId,
                        const LLI_Joint x_jointIn[],
                        const LLI_Frame* x_cTarget,
                        const LLI_Config* x_config,
                        const LLI_JointRange x_jointRange[],
                        LLI_Joint x_jointOut[],
                        LLI_ReversingResult* x_reversingResult
                  );

LLI_Status LLI_forwardKin
                  (
                        LLI_RobotId x_robotId,
                        const LLI_Joint x_jointIn[],
                        LLI_Frame* x_position,
                        LLI_Config* x_config
                  );

// ------------------------------------------------------------------
// Recorder-related functions
// ------------------------------------------------------------------
LLI_Status LLI_recInit (LLI_RobotId x_robotId);
LLI_Status LLI_recStart(LLI_RobotId x_robotId);
LLI_Status LLI_recStop (LLI_RobotId x_robotId);
LLI_Status LLI_recStore(LLI_RobotId x_robotId);

#ifdef __cplusplus
} // extern "C"
#endif // __cplusplus
```