

**Stäubli C programming interface  
for low level robot control  
s6.6.4**

## Table of content

<b>I.</b>	<b>Overview .....</b>	<b>3</b>
a.	<i>Programming interface</i>	3
b.	<i>Dynamic architecture</i>	3
<b>II.</b>	<b>Construction / destruction .....</b>	<b>4</b>
a.	<i>LLI_construct</i>	4
b.	<i>LLI_constructKinOnly</i>	4
c.	<i>LLI_destroy</i>	4
<b>III.</b>	<b>System programming interface .....</b>	<b>5</b>
a.	<i>LLI_state</i>	5
b.	<i>LLI_ioctl (configuration)</i>	5
c.	<i>Error management (LLI_plugEventFunc)</i>	6
d.	<i>IO functions</i>	6
e.	<i>Recorder functions</i>	6
<b>IV.</b>	<b>Control programming interface .....</b>	<b>7</b>
a.	<i>LLI_init</i>	7
b.	<i>LLI_enable</i>	7
c.	<i>LLI_disable</i>	7
d.	<i>LLI_get</i>	7
e.	<i>LLI_set</i>	7
<b>V.</b>	<b>Kinematics programming interface .....</b>	<b>9</b>
a.	<i>LLI_reverseKine</i>	9
b.	<i>LLI_forwardKine</i>	9

## I. Overview

### a. Programming interface

This document describes the C programming interface for low level robot control; the library header is fully defined with the file `lli_lowlevelinterf.h`.

Each function uses a `LLI_RobotId` parameter (created with `LLI_construct`) and return a `LLI_Status` to report if the function was completed correctly.

The programming interface is described in this document in four sections:

- Construction / destruction (library management)
- System programming interface (system parameters and state)
- Control programming interface (robot control functions)
- Kinematics programming interface (kinematics functions)

### b. Dynamic architecture

The use of the LLI library for robot control requires:

- an interrupt function, called by a hardware-generated clock, and synchronized with the drive control (see `LLI_construct`)
- a synchronous task, synchronized with the interrupt function, to control the robot (see `LLI_get` and `LLI_set`). The synchronous task must have the priority given by the `LLI_ioctl` command (`LLI_GET_SYNCHRO_TASK_PRIORITY`).
- some asynchronous tasks, to initialize and configure the system. The asynchronous task must have a lower priority than the synchronous task (a higher value for the VxWorks priority parameter).

This specification details which functions must be used in a synchronous context or asynchronous context, and which instruction can be used in both contexts.

---

## II. Construction / destruction

Each constructor can be used to specify the sys, usr and log directories to be used by LLI. If NULL, the corresponding partitions of the flash disk are used. It can be used to boot on a USB key (see readme - Developments).

Construction errors are stored in a file (see Error management (LLI\_pluginEventFunc)).

---

### a. LLI\_construct

This constructor gives access to control and kinematics programming interface (refer to Control programming interface and Kinematics programming interface).

LLI\_construct creates and configures the (static) software modules of the library, but performs no dynamic initialization that must be done later with LLI\_init.

The first parameter defines a function to be called in the context of the Sercos (CS8) or Starc (CS8C) system interrupt. It should not be called from anywhere else:

- it is called at the end of the interrupt, after commands are sent to the drives.
- the code for this function must be very short and valid in the context of the interrupt (for example, no wait on semaphores, etc...).

Typically, this function will just release a binary semaphore on which is suspended the synchronous task (see test program). When the synchronous task takes this semaphore, it must call the LLI\_get, then the LLI\_set functions.

---

### b. LLI\_constructKinOnly

This constructor gives access to the kinematics programming interface only (refer to Kinematics programming interface).

---

### c. 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.

### III. System programming interface

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

#### a. LLI\_state

LLI\_state returns 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

LLI\_state can be called in both synchronous and asynchronous contexts.

#### b. LLI\_ioctl (configuration)

The function LLI\_ioctl reads or modifies the system configuration; it can be called in both synchronous and asynchronous contexts, or only in asynchronous context, depending on the parameter:

Parameter	Set
LLI_GET_SOFTWARE_VERSION	Return the LLI library version (LLI_SoftwareVersion)
LLI_GET_ROBOT_TYPE	Return the robot type (LLI_robotType)
LLI_GET_JNT_NUMBER	Return the number of robot joints (unsigned int)
LLI_SET_USER_MARKS	(asynchronous context only) Set and store a new value of the user mark (double[])
LLI_GET_USER_MARKS	Get the value of the user mark position (double[]). In factory setting, this position corresponds to the physical mark on the joints
LLI_GET_JNT_RANGE	Get the arm software joints limits: maximum and minimum values (LLI_JointRange[]).
LLI_RECOVER_ABS_JNT_POS	(CS8 controller / asynchronous context only) Restore correct calibration parameters in the calibration board (DAPS). The robot must be near the position (double[]) passed to the function ( $\pm 1/2$ motor turn on each joint).
LLI_SET_ABS_JNT_POS	(asynchronous context only) Change the position (double[]) of each axis to match the position passed to the function. The new zero position of each axis is stored in the system configuration file arm.cfx.
LLI_GET_CYCLE_TIME	Get the time (double, in s) between two robot interrupts.
LLI_GET_SYNCHRO_TASK_PRIORITY	Get the VxWorks task priority (integer) to apply to the synchronous task.

Parameter	Set
LLI_SET_TORQUE_MODE	(asynchronous context only) Switch from the standard position control to the torque control mode.
LLI_SET_POS_VEL_MODE	(asynchronous context only) Switch from the torque control mode to the standard position control.
LLI_SET_TUNING	(asynchronous context only) Send a command to the drives to change arm tuning parameters. Arm power must be off.

### c. Error management (LLI\_plugEventFunc)

The controller outputs errors on its serial port COM1, (115200, 8 bits, no parity, 1 stop bit, VT100 emulation). Errors are also logged in file named error.log on the /log flash disk partition. Errors can be also redirected to the user using the function LLI\_plugEventFunc:

- This function plugs a user function that is called with each new system event.
- The parameter of the user function is pointer to the string that contains the event description

LLI\_plugEventFunc can be called in both synchronous and asynchronous contexts.

### d. IO functions

An IO is referenced by its name; LLI\_getDinId and LLI\_getDoutId convert this name into an id that is then used by LLI\_readDio or LLI\_writeDout for a fast IO access.

- LLI\_getDinId returns an ID on a Digital Input, the entry parameter is the name of Din as defined in the robot controller (asynchronous context only)
- LLI\_getDoutId returns an ID on a Digital Output, the entry parameter is the name of Dout (asynchronous context only)
- LLI\_readDio returns the value of the IO (input or output): 0, 1 or undefined
- LLI\_writeDout sets the output to 1 or 0

LLI\_getDinId and LLI\_getDoutId must be called in asynchronous context only; LLI\_readDio and LLI\_writeDout can be called in both synchronous and asynchronous contexts.

### e. Recorder functions

The system recorder records traces of the system state (command, feedback, IOs, ...) in a synchronous task. The data to record, and some recording parameters, are specified in a system configuration file (by default /usr/recorder/record.cfx).

- LLI\_recInit() configures the system recorder with the specified configuration file
- LLI\_recStart() starts the record defined with LLI\_recInit
- LLI\_recStop() stops the record started with LLI\_recStart
- LLI\_recStore() stores to the specified location the record completed with LLI\_recStop

LLI\_recInit and LLI\_recStore must be called in asynchronous context only; LLI\_recStart and LLI\_recStop can be called in both synchronous and asynchronous contexts.

---

## IV. Control programming interface

The following functions can be used after the constructor LLI\_construct has been called.

---

### a. LLI\_init

LLI\_init performs the dynamic initialization of the LLI library and completes the initialization started with LLI\_construct:

- hardware is initialized : Sercos (CS8) / Starc (CS8C) communication with drives
- robot is calibrated.

LLI\_init must be called in asynchronous context only.

---

### b. LLI\_enable

LLI\_enable sends a request to start the arm power enabling sequence; the enabling sequence is started with the next call of the LLI\_set function. An error is raised if the enabling sequence cannot be completed.

- The enabling sequence can be aborted at any time with the LLI\_disable function
- The enabling sequence can be tracked with the LLI\_state function

LLI\_enable can be called in both synchronous and asynchronous contexts.

---

### c. LLI\_disable

LLI\_disable sends a request to start the arm power disabling sequence; the disabling sequence is started with the next call of the LLI\_set function.

- The disabling sequence aborts the pending enabling sequence, if any, without error message.
- The disabling sequence can be tracked with the LLI\_state function.

LLI\_disable can be called in both synchronous and asynchronous contexts.

---

### d. LLI\_get

LLI\_get updates the system state by reading feedback from the drives. The refreshed state can then be read with the LLI\_state function.

LLI\_get must be called before LLI\_set, at each synchronous cycle.

---

### e. LLI\_set

LLI\_set updates the system state by sending a new command to the drives. An error is raised if invalid commands are detected (position, velocity, delta position or delta velocity out of range); arm power is then automatically disabled.

- The velocity command is supposed to be the instantaneous derivative of the position command. Position and velocity are micro-interpolated in the drives, to allow the servo loop to run at higher rates.

- Each drive contains a position, a velocity loop, and a current loop. External velocity and torque feedforwards may be added to improve the control, especially to compensate for the joint couplings.
- When torque mode is activated (see LLI\_ioctrl (configuration)), the position and velocity control loops are disabled. Then, the torque feedforward 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.

LLI\_set must be called after LLI\_get at each synchronous cycle.



---

## V. Kinematics programming interface

The following functions can be used after the constructor `LLI_construct` or `LLI_constructKinOnly` has been called.

---

### a. `LLI_reverseKine`

The `LLI_reverseKine` function computes a joint position corresponding to a specified Cartesian position and robot configuration. The Cartesian position is the position of the robot flange referenced to the base of the robot.

- The reference joint position and joint limits are used when several solutions can be found: the result is then the nearest joint position within the joint limits.
- The function returns a diagnostic when the computation has no solution.

`LLI_reverseKine` can be called in both synchronous and asynchronous contexts.

---

### b. `LLI_forwardKine`

The `LLI_forwardKine` function returns the Cartesian position of the robot flange, and the robot configuration corresponding to a specified joint position.

`LLI_forwardKine` can be called in both synchronous and asynchronous contexts.