



-PINO

API Reference (sample) ver 1.10

API Reference

API REFERENCE	2
1 INTRODUCTION.....	2
2 API, GENERAL INFORMATION	3
2.1 NAMING CONVENTIONS	3
2.2 SEND-TYPE.....	3
2.3 QUERY-TYPE.....	4
2.4 SET-TYPE	4
2.5 GET-TYPE.....	4
3 API-FUNCTION GROUPS	4
3.1 SYSTEM GROUP, PINO_SYSTEM_.....	4
3.2 JOINT GROUP, PINO_JOINT_.....	5
3.2.1 <i>Joint Positions</i>	5
3.2.2 <i>Joint Motor and Power-Control</i>	7
3.3 SUBCPU GROUP, PINO_SUBCPU_.....	9
3.4 SENSOR GROUP, PINO_SENSOR_.....	9
3.4.1 <i>Reading Sensor Values</i>	9
3.4.2 <i>Gyroscope Sensors</i>	10
3.4.3 <i>Accelerometers</i>	11
3.4.4 <i>Foot force sensors</i>	13
3.5 CONFIG GROUP, PINO_CONFIG.....	14
3.6 MOTION GROUP, PINO_MOTION_.....	14
3.6.1 <i>Overview of Motion-group API-functions</i>	15
3.7 SENSORSERVO GROUP, PINO_SENSORSERVO.....	18
4 APPENDIX A, API PARAMETER TYPES.....	21
5 APPENDIX B, PINO JOINT NAMES.....	22

1 Introduction

Pino's PC-side software comes in form of a DLL (dynamic link library) file, named pino2core.dll and header-files written in C and C# language, named pino2core.h (C) and pino2core.cs (C#) respectively. The software has successfully been tested on Windows XP and Windows 2000 systems. There is also an additional file pino2error.h which provides names of the error codes.

Additionally a graphical user interface for Pino is provided including full source code written in C#. The intention of this GUI is for the user to quickly get started operating Pino and also serve as an example for how to use the DLL.

Some of the functionality of the Pino software requires rapid execution and quick response, things that can not be guaranteed on a non-realtime operating-system such as Microsoft Windows system. Therefore it is recommended that when using Pino, not to run other heavy applications on the computer.

2 API, general information

All API function-calls returns an error code of `PinoStatus`-type (unsigned long integer), which equals zero if the call is successful. If it is non-zero, something went wrong and you can get a string that describes the error by using the API call `PINO_QueryStatusMessage` and/or check the name of the error in `pino2error.h`.

Apart from a few API-functions the first parameter that most functions take is “`PinoSession`” (of type unsigned long integer) and this is the session-handler that is given after a successful initialization of Pino via the `PINO_Init` function.

2.1 Naming Conventions

All API function-names start with `PINO_`, then the name of the group of functions the function belongs to follows. There are seven groups of API functions: `System`, `Config`, `Joint`, `SubCPU`, `Sensor`, `Motion` and `SensorServo`. Thus all API functions starts with one of the following strings: `PINO_System_`, `PINO_Config_`, `PINO_Joint_`, `PINO_SubCPU_`, `PINO_Motion_`, `PINO_SensorServo_`.

Moreover all API function-names ends with an “action-name” that describes what kind of action the function performs. In between the group-name and the action-name there can optionally be a sub-group name. An example on this is the `PINO_Sensor_GyroZ_QueryRawData` function which has a sub-group name that is `GyroZ` to define that this function operates on the `GyroZ` in the `Sensor` group. The action-name in this example is then `QueryRawData`.

API-functions thus follow this template:

```
PINO_GroupName_(SubGroupName)_ActionName
```

Also the action-names have some naming conventions. There are four “standard types” of actions and a number of custom types. The standard-types are `Send`, `Query`, `Set` and `Get`. The action-name starts with the type if it belongs to one the standard types. The standard action-types will now be described.

2.2 Send-type

A successful call to a `Send`-type API-function means that one or more data-packets will be sent to Pino and on the CAN (controller area network) bus to one of the `SubCPU` boards. A `Send`-type API-function returns however immediately, before the packet actually has been dispatched from the PC. Thus, even if the call returns a success (zero) there is no guarantee that the call actually was successful and that the data reached the destination as intended. If there is a problem with the Pino-system, for example because of a disconnected CAN-cable to the destination subCPU or a broken sub-CPU board, these calls will still return success.

Normally the only way to test that a `Send`-call really was successful, is to use a corresponding `Query`-type API-function to check that the action of the `Send`-type function was correctly carried out. However as long as the Pino system is working correctly, a `Send`-call can be assumed to be successful when it returns a success (zero).

2.3 Query-type

Query-type API-functions will, just as the Send-type, send a data-packet to one of the SubCPU. However a Query-type function also waits for a response data-packet from the addressed subCPU, alternatively if no response is received in a specified time-period, the call will return the errorcode `PINO_E_TIMEOUT`. (The time-period can be set with the `PINO_Config_SetCanQueryTimeOut` API-function).

Thus a Query-type function will take some time before it returns. If it is successful the time is normally around 20-50mS, but can depend on the load of the CAN-bus and serial connection, if timed out the call will take the timeout-period, there the default is 500mS. If this can not be accepted, the call must be done in a separate thread in the user-software.

2.4 Set-type

A Set-type API-function will set a property or value in the PC-side software, no communication to Pino needs to take place. Thus these calls are very fast and a success return always means that the action also took place.

2.5 Get-type

A Get-type API-function is a way to get hold of properties and values that is located in the PC-side software. All Set-type functions have a corresponding Get-type function. Get-type functions on the other hand do not need to have a Set-type function, if the property is of such type that it can not be set directly or if it is a constant.

3 API-function groups

3.1 System group, `PINO_System_`

The system group consists of two functions used for setting up and closing a Pino Session, one function for getting the current version of the system and one function for converting errorcodes into messages.

```
PinoStatus PINO_System_Init (
    PinoCString    szPort,
    PinoPSession   phSession)
```

Description: This is the call that starts a Pino Session. Before calling this function to initiate the Pino session, Pino must be powered on and the serial connection must be assigned to the correct COM-port. `szPort` is a string with the com-port name (“com1”, “com2” etc). If the call is successful (ie `PinoStatus` equals zero) the `phSession` handle will be set and this handle needs to be used in basically all other API-calls.

```
PinoStatus PINO_System_Close (
    PinoSession    hSession)
```

Description: This ends the Pino Session and should be called when closing the PC-application.

```
PinoStatus PINO_System_QueryStatusMessage (
    PinoStatus     status,
    PinoString     szMessageBuffer,
```