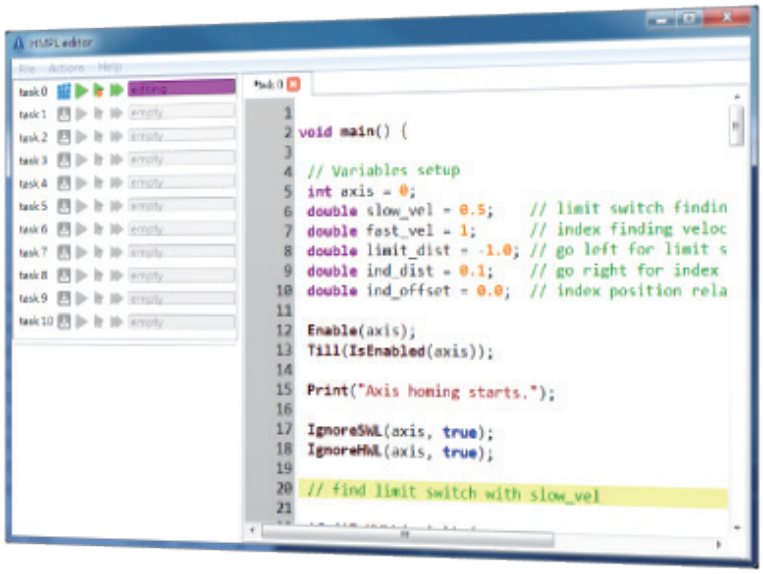
HIWIN®

Print("Hello World");



HIMC

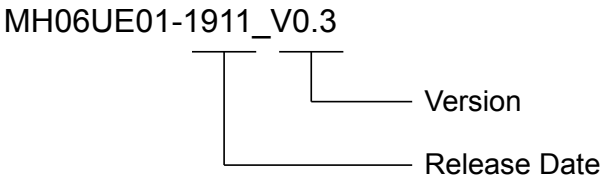
HMPL User Guide

www.hiwinmikro.tw

MH06UE01-1911_V0.3

Revision History

The version of the guide is also indicated on the bottom of the front cover.



Release Date	Version	Applicable Software Version	Revision Contents
Nov. 29th, 2019	0.3	iA Studio 1.2.4032.0	1. Section 2.10: Add remarks to function Till. 2. Chapter 11: Add flow of using PT function. 3. Chapter 18: Add homing procedure example for E1 series servo drive gantry mode.
Apr. 2nd, 2019	0.2	iA Studio 1.1.3772.0	1. Modify chapter's arrangement. 2. Add Chapter "Synchronized Motion Functions", "Filter Functions" and "Error Functions". 3. Rename function: ■ Chapter 2 IsOperMode → IsSystemOper IsPreOpMode → IsSystemPreOp ■ Chapter 9 SetGPO_OnOff → SetGPO SetGPO_Toggle → ToggleGPO 4. Add function: ■ Chapter 2 IsSystemError, GetSlaveNum RescanMoE, SetHMIScope ■ Section 5.3 GetSWRL, SetSWRL GetSWLL, SetSWLL ■ Section 8.3 GroupReset

			■ Chapter 9 SetSlvGPO, ToggleSlvGPO IsSlvGPI_On, IsSlvGPO_On ■ Chapter 10 SetTableValue, GetTableValue ■ Chapter 16 GetSlvSt, SetSlvSt GetConfigVar, SetConfigVar
Apr. 17th, 2018	0.1	iA Studio 1.0.2461.0	First edition.

(This page is intentionally left blank.)

Table of Contents

1. Introduction 1-1
1.1 How HMPL works 1-2
1.2 Legal disclaimer 1-2
1.3 Data types 1-3
1.4 Scope Rules 1-4

2. HIMC System Functions 2-1
2.1 IsSystemOper 2-2
2.2 IsSystemPreOp 2-3
2.3 IsSystemError 2-4
2.4 DisableAll 2-5
2.5 StopAll 2-6
2.6 EStop 2-7
2.7 GetSlaveNum 2-8
2.8 RescanMoE 2-9
2.9 SetHMIScope 2-10
2.10 Till 2-11
2.11 Sleep 2-12
2.12 SendEvent 2-13
2.13 RunScheduler 2-14
2.14 MutexLock 2-15
2.15 MutexUnlock 2-16
2.16 TON 2-17
2.17 TOF 2-19
2.18 _TASKID_ 2-21
2.19 _AUTORUN_ 2-22

3. String Functions 3-1
3.1 Overview 3-2
3.2 Print 3-3
3.3 StringPrint 3-5
3.4 StringLen 3-7
3.5 IsStringEqual 3-8
3.6 StrFindChar 3-9

Table of Contents

3.7 StrFindCharEx 3-10
3.8 StrFindStr 3-12
3.9 StringCopy 3-13
3.10 StringCopyEx 3-14
3.11 StringCat 3-16
3.12 StringCatEx 3-17
3.13 StringToDouble 3-19
3.14 MemoryCopy 3-20
3.15 MemorySet 3-22
3.16 IsMemoryEqual 3-23
3.17 START_ASCII_AGENT 3-24

4. Math Functions 4-1
4.1 sin 4-2
4.2 cos 4-3
4.3 tan 4-4
4.4 asin 4-5
4.5 acos 4-6
4.6 atan 4-7
4.7 atan2 4-8
4.8 abs 4-9
4.9 fabs 4-10
4.10 ceil 4-11
4.11 floor 4-12
4.12 Idexp 4-13
4.13 exp 4-14
4.14 pow 4-15
4.15 log 4-16
4.16 log10 4-17
4.17 sqrt 4-18
4.18 cbrt 4-19
4.19 hypot 4-20

Table of Contents

5. Axis Functions 5-1
5.1 Overview 5-3
5.1.1 Axis variables 5-3
5.1.1.1 Motion variables 5-3
5.1.1.2 Profile generator variables 5-5
5.1.2 Axis faults 5-6
5.1.3 Axis state diagram 5-6
5.2 Axis Motion Control 5-7
5.2.1 Enable 5-7
5.2.2 Disable 5-8
5.2.3 Reset 5-9
5.2.4 MoveAbs 5-10
5.2.5 MoveRel 5-11
5.2.6 MoveVel 5-12
5.2.7 Stop 5-13
5.3 Axis Setting 5-14
5.3.1 GetMaxVel 5-14
5.3.2 SetVel 5-15
5.3.3 GetMaxAcc 5-16
5.3.4 SetAcc 5-17
5.3.5 GetMaxDec 5-18
5.3.6 SetDec 5-19
5.3.7 GetSWRL 5-20
5.3.8 SetSWRL 5-21
5.3.9 GetSWLL 5-22
5.3.10 SetSWLL 5-23
5.3.11 GetSMTime 5-24
5.3.12 SetSMTime 5-25
5.3.13 GetMoveTime 5-26
5.3.14 GetSettlingTime 5-27
5.3.15 SetPos 5-28
5.3.16 GetPosFb 5-29
5.3.17 GetPosOffset 5-30
5.3.18 GetPosErr 5-31

Table of Contents

5.3.19 GetRefPos 5-32
5.3.20 GetRefVel 5-33
5.3.21 GetRefAcc....5-34
5.3.22 GetPosOut 5-35
5.3.23 GetVelOut 5-36
5.3.24 GetAccOut 5-37
5.3.25 IgnoreHWL....5-38
5.3.26 IgnoreSWL....5-39
5.3.27 GetAxisNum....5-40
5.4 Axis Status 5-41
5.4.1 IsEnabled 5-41
5.4.2 IsMoving....5-42
5.4.3 IsInPos 5-43
5.4.4 IsErrorStop....5-44
5.4.5 IsGantry 5-45
5.4.6 IsGrouped 5-46
5.4.7 IsSync 5-47
5.4.8 IsHWLL 5-48
5.4.9 IsHWRL....5-49
5.4.10 IsSWLL 5-50
5.4.11 IsSWRL....5-51
5.4.12 IsCompActive....5-52

6. Synchronized Motion Functions 6-1
6.1 Overview 6-2
6.1.1 Synchronized motion variables....6-3
6.1.2 Example 6-3
6.2 EnableGear.... 6-5
6.3 DisableGear 6-6
6.4 GearIn 6-7
6.5 GearOut 6-8

Table of Contents

7. Gantry Functions 7-1
7.1 Overview 7-2
7.1.1 Gantry pair setup example 7-3
7.2 EnableGantryPair 7-4
7.3 DisableGantryPair 7-5

8. Group Functions 8-1
8.1 Overview 8-2
8.1.1 Group motion variables 8-3
8.1.2 Group state diagram 8-4
8.1.3 Example 8-5
8.1.3.1 Basic group setup 8-5
8.1.3.2 Advanced group setup and velocity blending 8-6
8.2 Group Motion Control 8-8
8.2.1 EnableGroup 8-8
8.2.2 DisableGroup 8-9
8.2.3 Basic motion command 8-10
8.2.3.1 LineAbs2D 8-10
8.2.3.2 LineAbs3D 8-12
8.2.3.3 LineRel2D 8-13
8.2.3.4 LineRel3D 8-15
8.2.3.5 Arc2D 8-16
8.2.3.6 Circle2D 8-18
8.2.4 Advanced motion command 8-20
8.2.4.1 Bezier 8-20
8.2.4.2 LinAbs 8-23
8.2.4.3 LinRel 8-25
8.2.4.4 CircAbs 8-27
8.2.5 StopGroup 8-29
8.3 Group Setting 8-30
8.3.1 AddAxisToGrp 8-30
8.3.2 RemoveAxisFromGrp 8-31
8.3.3 SetupGroup 8-32
8.3.4 UngrpAllAxes 8-33

Table of Contents

8.3.5 GetGroupID 8-34
8.3.6 SetGrpMotionProfile 8-35
8.3.7 SetGrpKin 8-37
8.3.8 GroupReset 8-38
8.4 Group Status Functions 8-39
8.4.1 IsGrpEnabled 8-39
8.4.2 IsGrpMoving 8-40
8.4.3 IsGrpInPos 8-41

9. GPIO Functions 9-1
9.1 SetGPO 9-2
9.2 ToggleGPO 9-3
9.3 IsGPI_On 9-4
9.4 IsGPO_On 9-5
9.5 HIMC_GPI 9-6
9.6 HIMC_GPO 9-7
9.7 SetSlvGPO 9-8
9.8 ToggleSlvGPO 9-9
9.9 IsSlvGPI_On 9-10
9.10 IsSlvGPO_On 9-11
9.11 SLV_GPI 9-12
9.12 SLV_GPO 9-13

10. User Table Functions 10-1
10.1 SetUserTable 10-2
10.2 GetUserTable 10-4
10.3 SetTableValue 10-6
10.4 GetTableValue 10-7
10.5 SaveUserTable 10-8
10.6 LoadUserTable 10-10

Table of Contents

11. Position Trigger Functions 11-1
11.1 Overview 11-2
11.1.1 PT variables 11-2
11.1.2 Flow of using PT function 11-4
11.2 EnablePT 11-5
11.3 DisablePT 11-6
11.4 IsPTEnabled 11-7
11.5 SetPT_StartPos 11-8
11.6 SetPT_EndPos 11-9
11.7 SetPT_Interval 11-10
11.8 SetPT_PulseWidth 11-11
11.9 SetPT_Polarity 11-12

12. Touch Probe Functions 12-1
12.1 EnableTouchProbe 12-2
12.2 DisableTouchProbe 12-3
12.3 IsTouchProbeEnabled 12-4
12.4 IsTouchProbeTriggered 12-5
12.5 GetTouchProbePos 12-6

13. Dynamic Error Compensation Functions 13-1
13.1 EnableComp 13-2
13.2 DisableComp 13-3
13.3 SetupComp 13-4
13.4 SetupComp2D 13-7

14. Filter Functions 14-1
14.1 Overview 14-2
14.1.1 Example 14-2
14.2 EnableAxisVsf 14-3
14.3 DisableAxisVsf 14-4
14.4 SetAxisVsf 14-5
14.5 EnableAxisInShape 14-6
14.6 DisableAxisInShape 14-7

Table of Contents

14.7 SetAxisInShape 14-8
14.8 EnableGrpInShape 14-10
14.9 DisableGrpInShape 14-11
14.10 SetGrpInShape 14-12

15. HMPL Task Functions 15-1
15.1 StartTask 15-2
15.2 StartTaskFunc 15-3
15.3 StopTask 15-4
15.4 StopAllTask 15-5
15.5 IsTaskStop 15-6

16. Variables Operating Functions 16-1
16.1 GetSlvVar 16-2
16.2 GetSlvVarEx 16-3
16.3 SetSlvVar 16-4
16.4 GetSlvSt 16-5
16.5 SetSlvSt 16-6
16.6 GetConfigVar 16-7
16.7 SetConfigVar 16-8

17. Error Functions 17-1
17.1 GetSystemLastErr 17-2
17.2 GetAxisLastErr 17-3
17.3 ClearAxisLastErr 17-4
17.4 GetGrpLastErr 17-5
17.5 ClearGrpLastErr 17-6

18. Homing Procedure Examples 18-1
18.1 Basic: index signal only 18-2
18.2 Advanced: index signal & limit switch 18-4
18.3 For E1 series servo drive gantry mode 18-12

Table of Contents

19. Appendix 19-1
19.1 Buffer modes 19-2
19.2 Transition modes 19-3
19.3 Coordinate systems 19-4
19.4 Kinematics 19-5
19.5 Math constants 19-6
19.6 System variables 19-6
19.7 Bit manipulation 19-7

(This page is intentionally left blank.)

1. Introduction

1. Introduction 1-1
1.1 How HMPL works 1-2
1.2 Legal disclaimer 1-2
1.3 Data types 1-3
1.4 Scope Rules 1-4

1.1 How HMPL works

HIWIN Motion Programming Language (HMPL) constructs independent tasks with C-like syntax at users' disposal.

Note 1: All motion related variables in HMPL are in MKS unit system (meter-kilogram-second).

Note 2: The icon  indicates that the function can be used in iA Studio's Message Window or self-installed terminal.

1.2 Legal disclaimer

Users can adopt or modify any of the sample codes provided in this guide for specific uses. However, the correctness, effectiveness and safety cannot be guaranteed in different application scenarios. Users should take full responsibility for the safety and the effectiveness of the software implementations.

1.3 Data types

In HMPL, data types are used to declare variables or get the function's return value. The type of a variable determines the space size's occupation in storage and its valid value.

Table 1.3.1

Type	Description	Size (Byte)	Valid value
char int8_t	8-bit signed integer	1	-128 ~ 127
unsigned char uint8_t	8-bit unsigned integer	1	0 ~ 255
short int16_t	16-bit signed integer	2	-32768 ~ 32767
unsigned short uint16_t	16-bit unsigned integer	2	0 ~ 65535
int int32_t	32-bit signed integer	4	-2147483648 ~ 2147483647
unsigned int uint32_t	32-bit unsigned integer	4	0 ~ 4294967295
long long int64_t	64-bit signed integer	8	-9223372036854775808 ~ 9223372036854775807
unsigned long long uint64_t	64-bit unsigned integer	8	0 ~ 18446744073709551615
float	32-bit floating-point type (6 decimal digits precision)	4	1.17549e-38 ~ 3.40282e+38
double	64-bit floating-point type (15 decimal digits precision)	8	2.225074e-308 ~ 1.797693e+308
int* char* double* …	Pointer type, which contains the address of a storage location of a variable of a particular type.	8	N/A
void	A function with void return type returns no value.	N/A	N/A
void*	Generic pointer type, which contains the address of a storage location of a variable of any type.	8	N/A
Timer	A type to declare a timer object for function TON and TOF.	8	N/A

1.4 Scope Rules

The scope of a defined variable or function indicates its existence region in the HMPL task. Beyond the region, the variable and the function cannot be accessed.

Table 1.4.1

Type	Scope	Declaration placement	Description
global function	global scope	in **task 0**	can be used anywhere
task function	task scope	not in **task 0**	can only be used in that task
global variable	global scope	out of all functions but in **task 0**	can be used anywhere
task variable	task scope	out of all functions and **task 0**	can only be used in that task
local variable	block scope	in a block	can only be used in that block
	function scope	in a function	can only be used in that function

Note: Global variables and task variables will only be initialized at the compile time.

Example

1.

```
//  in task 0
//  Declare a global variable
int global_var = 100;

//  Declare a global function
void GlobalFunction1() {
    Print("%d", global_var);
}
```

2.

```
//  in task 1
//  Declare a task variable
int task_var = 0;

//  Declare a task function
void TaskFunction1() {
    //  Declare a local variable
    int local_var = task_var;
    for (int i = 0; i < local_var; ++i) {  //  block start
    global_var += i;  //  i is a local variable with block scope
  }  //  block end
  global_var += local_var;
}
void main() {
    task_var = 10;
    TaskFunction1();
    GlobalFunction1();  //  the output is 155
}
```

(This page is intentionally left blank.)

2. HIMC System Functions

2. HIMC System Functions 2-1
- 2.1 IsSystemOper 2-2
- 2.2 IsSystemPreOp 2-3
- 2.3 IsSystemError 2-4
- 2.4 DisableAll 2-5
- 2.5 StopAll 2-6
- 2.6 EStop 2-7
- 2.7 GetSlaveNum 2-8
- 2.8 RescanMoE 2-9
- 2.9 SetHMIScope 2-10
- 2.10 Till 2-11
- 2.11 Sleep 2-12
- 2.12 SendEvent 2-13
- 2.13 RunScheduler 2-14
- 2.14 MutexLock 2-15
- 2.15 MutexUnlock 2-16
- 2.16 TON 2-17
- 2.17 TOF 2-19
- 2.18 _TASKID_ 2-21
- 2.19 _AUTORUN_ 2-22

2.1 IsSystemOper

Purpose

To query whether HIMC system is at “operation” state. If it is, the motion related functions can be used.

Syntax

```
int IsSystemOper();
```

Parameter

N/A

Return value

It will return an **int** value **TRUE** (1) if HIMC system is at “operation” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.1.3761.0

2.2 IsSystemPreOp

Purpose

To query whether HIMC system is at “pre-operation” state. If it is, the communication between HIMC and the slaves is established, but the motion related functions cannot be used.

Syntax

```
int IsSystemPreOp();
```

Parameter

N/A

Return value

It will return an **int** value **TRUE** (1) if HIMC system is at “pre-operation” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.1.3761.0

2.3 IsSystemError

Purpose

To query whether HIMC system is at “error” state. If it is, an error occurs in the communication between HIMC and the slaves.

Syntax

```
int IsSystemError();
```

Parameter

N/A

Return value

It will return an **int** value **TRUE** (1) if HIMC system is at “error” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.1.3761.0

2.4 DisableAll

Purpose

To disable all axes and all axis groups.

Syntax

```
int DisableAll();
```

Parameter

N/A

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

The motion queues of the axes and the axis groups will be cleared.

Requirement

Minimum supported version	iA Studio 0.23.2087.0

2.5 StopAll

Purpose

To stop all axes and all axis groups with kill deceleration.

Syntax

```
int StopAll();
```

Parameter

N/A

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

The motion queues of the axes and the axis groups will be cleared.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

2.6 EStop

Purpose

The emergency-stop of the controller.

Syntax

```
int EStop();
```

Parameter

N/A

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.2156.0

2.7 GetSlaveNum

Purpose

To get the number of the slaves that are connected to the controller.

Syntax

```
int GetSlaveNum();
```

Parameter

N/A

Return value

The number of the slaves that are connected to the controller.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

2.8 RescanMoE

Purpose

To rescan MoE when the connection between HIMC and the slaves is broken.

Syntax

```
int RescanMoE();
```

Parameter

N/A

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

2.9 SetHMIScope

Purpose

To start or stop executing the scope.

Syntax

```
int SetHMIScope(
    int start
);
```

Parameter

start [in]	Set it as “1” to start recording data. Set it as “0” to stop recording data.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

2.10 Till

Purpose

Stop executing the HMPL task until the specific condition is met.

Syntax

```
Till(
    condition
);
```

Parameter

condition [in] Type: **int**

The result of the conditional evaluation → **true** (nonzero) or **false** (0)

Remarks

When calling this function, users should be responsible for HIMC's abnormal behavior led by the invalidity of HMPL application (when the specific condition cannot be met).

Example

```
int main() {
    Till(IsEnabled(0) && IsEnabled(1));

    //  Do something
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

2.11 Sleep

Purpose

Stop executing the HMPL task for a specific period.

Syntax

```
void Sleep(
    int ms
);
```

Parameter

ms [in] Time in milliseconds.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

2.12 SendEvent

Purpose

To send an event by ID to host PC.

Syntax

```
int SendEvent(
    unsigned short evt_id
);
```

Parameter

evt_id [in] User-defined event ID.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

(1) Host PC can set the callback function to capture the event ID with the function "HIMC_SetHmplEvtCallback" in HIMC API Reference Guide.

(2) This function cannot be called too often (typically 1KHz). If it is called too often, it will be blocked until the average calling frequency is lower than 1KHz.

Requirement

Minimum supported version	iA Studio 0.11.1555.0

2.13 RunScheduler

Purpose

Make the calling task release CPU resources for any other task that is ready to run.

Syntax

```
void RunScheduler();
```

Parameter

N/A

Requirement

Minimum supported version	iA Studio 0.23.1892.0

2.14 MutexLock

Purpose

To lock the mutex object by ID.

Syntax

```
void MutexLock(
    int mutex_id
);
```

Parameter

mutex_id [in] Mutex object ID.
There are 8 available mutex objects, so the ID can be 0~7.

Return value

N/A

Remark

(1) If the mutex object is not currently locked by any task, this function will lock the mutex object and return immediately. The mutex object is owned by the task which locks it, and it remains locked until the owner task calls MutexUnlock function or the owner task is stopped.
(2) If the mutex object is currently locked by another task, this function will be blocked until the mutex object is unlocked.
(3) If the mutex object is already locked by the same task calling this function, the task will be stopped and a run-time error message will be sent out.

Requirement

Minimum supported version	iA Studio 0.23.2033.0

2.15 MutexUnlock

Purpose

To unlock the mutex object by ID.

Syntax

```
void MutexUnLock(
    int mutex_id
);
```

Parameter

mutex_id [in] Mutex object ID.
There are 8 available mutex objects, so the ID can be 0~7.

Return value

N/A

Remark

If the mutex object is not currently locked by the calling task, nothing will happen.

Requirement

Minimum supported version	iA Studio 0.23.2033.0

2.16 TON

Purpose

Rising-edge timer function. When the IN condition is established (the parameter turns to 1), this function will return the return value after PT milliseconds.

Syntax

```
int TON(
    Timer *timer_p,
    int IN,
    int PT
);
```

Parameter

timer_p [in]	A pointer to the buffer to store the timer object.
IN [in]	Timer command.
PT [in]	Programmed time. Parameter unit: millisecond

Return value

It will return an **int** value **0** if the output signal is low, a **nonzero** value if the output signal is high.

Remark

(1) The timer starts on a rising pulse of IN input and stops as soon as the elapsed time is equal to the programmed time. A falling pulse of IN input resets the timer to 0. When the programmed time is elapsed, the output signal is set to 1. When the input command falls, it is reset to 0.

(2) To restart the timer, initialize the timer objects by `TimerInit` (timer initializer).

Example

```
int main() {

    Timer timer1 = TimerInit;
    Timer timer2 = TimerInit;
    int var1 = 0;
    int var2 = 0;
    int var3 = 0;
    int var4 = 0;
    for (;;) {

      var1 = ...;
      var2 = ...;
      var3 = TON(&timer1, var1 > var2, 100);

      if (TON(&timer2, var3, 500)) {
        var4 = var1 + var2 + var3;
      }
    }
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

2.17 TOF

Purpose

Falling-edge timer function. When the IN condition is established (the parameter turns to 1), this function will return the return value after PT milliseconds.

Syntax

```
int TOF(
    Timer *timer_p,
    int IN,
    int PT
);
```

Parameter

timer_p [in]	A pointer to the buffer to store the timer object.
IN [in]	Timer command.
PT [in]	Programmed time. Parameter unit: millisecond

Return value

It will return an **int** value **0** if the output signal is low, a **nonzero** value if the output signal is high.

Remark

(1) The timer starts on a falling pulse of IN input and stops as soon as the elapsed time is equal to the programmed time. A rising pulse of IN input resets the timer to 0. When the IN input rises to TRUE, the output signal is set to 1. When the programmed time is elapsed, it is reset to 0.

(2) To restart the timer, initialize the timer objects by `TimerInit` (timer initializer).

Example

```
int main() {

    Timer timer1 = TimerInit;
    Timer timer2 = TimerInit;
    int var1 = 0;
    int var2 = 0;
    int var3 = 0;
    int var4 = 0;
    for (;;) {

      var1 = ...;
      var2 = ...;
      var3 = TOF(&timer1, var1 > var2, 100);

      if (TOF(&timer2, var3, 500)) {
        var4 = var1 + var2 + var3;
      }
    }
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

2.18 _TASKID_

Purpose

To query current HMPL task ID.

Example

```
#if _TASKID_ == 0
int global_var = 0;  //  only be compiled if current task ID is 0
#endif

void test(){

    for (;;) {
      if (HIMC_GPI(0)) {
        StopTask( _TASKID_ );  //  Stop current task
    }
  }
}

void main() {
    test();
}
```

Requirement

Minimum supported version	iA Studio 0.23.2033.0

2.19 _AUTORUN_

Purpose

To automate a task at boot time.

Example

```
_AUTORUN_ void main() {
    Till(IsSystemOper());
    //  Do something
}
```

Requirement

Minimum supported version	iA Studio 0.22.1850.0

3. String Functions

3. String Functions 3-1
- 3.1 Overview 3-2
- 3.2 Print 3-3
- 3.3 StringPrint 3-5
- 3.4 StringLen 3-7
- 3.5 IsStringEqual 3-8
- 3.6 StrFindChar 3-9
- 3.7 StrFindCharEx 3-10
- 3.8 StrFindStr 3-12
- 3.9 StringCopy 3-13
- 3.10 StringCopyEx 3-14
- 3.11 StringCat 3-16
- 3.12 StringCatEx 3-17
- 3.13 StringToDouble 3-19
- 3.14 MemoryCopy 3-20
- 3.15 MemorySet 3-22
- 3.16 IsMemoryEqual 3-23
- 3.17 START_ASCII_AGENT 3-24

3.1 Overview

In HMPL, strings are one-dimensional arrays of characters, terminated by a null character '\0'. For example, the following declaration and initialization create a string "HIMC".

```
char str[5] = {'H', 'I', 'M', 'C', '\0'};
```

To hold the null character at the end of the array, the size of the character array containing the string should be the number of the characters in the text plus 1. Therefore, the size of the example is 5.

The above statement can also be written as follows. There is no need to place the null character at the end of a string. The HMPL compiler will automatically set the size of the string to 5 and place '\0' at the end of the string when it initializes the array.

```
char str[] = "HIMC";
```

The memory layout for the two strings above are the same, as Table 3.1.1 shows.

Table 3.1.1

str[0]	str[1]	str[2]	str[3]	str[4]
H	I	M	C	\0

Note: In HMPL, the maximum string length is 512. Users can get this value via "HMPL_STR_MAX_LEN".

3.2 Print

Purpose

To write a formatted string to the message window.

Syntax

```
void Print(
    char *format,
    …
);
```

Parameter

format [in] A pointer to the buffer which contains the text to be written to the message window.
It can optionally contain embedded format specifiers that follows the prototype "% specifier".
The specifier defines the type and the corresponding argument.

Specifier	Output	Example
d or i	Signed decimal integer	589
u	Unsigned decimal integer	589
x	Unsigned hexadecimal integer	24d
c	Character	M
s	String of characters	Hello world
f	Decimal floating point with six digits of precision	589.000000
e	Scientific notation with six digits of precision	5.890000e+02
g	The shortest representation of %e or %f	589
%	A % followed by another % presents a single %.	%

… [in] Additional parameters.
Each parameter contains a value to replace a format specifier in the format string. There should be at least as many as these parameters as the number of values specified in the format specifiers. Extra parameters will be ignored by the function.

Return value

The total number of the characters. If an error occurs, it will return -1.

Example

```
void main() {

    char str[] = "hello world";
    int var1 = 321;
    double var2 = 1428.57;

    Print("var1: %d, var2: %f, str: %s", var1, var2, str);
    //  var1: 321, var2: 1428.570000, str: hello world

    Print("var2: %e", var2);
    //  var2: 1.428570e+03

    Print("var2: %g", var2);
    //  var2: 1428.57
}
```

Requirement

Minimum supported version	iA Studio 0.23.1948.0

3.3 StringPrint

Purpose

To write a formatted string to the buffer.

Syntax

```
void StringPrint(
    char *destination,
    char *format,
    …
);
```

Parameter

destination [out]	A pointer to the buffer to store the result of the string.
format [in]	A pointer to the buffer which contains the text to be written to the message window. **Refer to Section 3.2 Print for details.**
… [in]	Additional parameters. Each parameter contains a value to replace a format specifier in the format string. There should be at least as many as these parameters as the number of values specified in the format specifiers. Extra parameters will be ignored by the function.

Return value

The total number of the characters. If an error occurs, it will return -1.

Remark

(1) This function is similar to Print. The difference is that the output string is written to the buffer instead of the message window.

(2) The source string's terminating null character will also be copied. To avoid overflow, the size of the array pointed by the destination string should be long enough to contain the source string (the terminating null character is included).

Example

```
void main() {

    char dest[80];
    char str[] = "hello world";
    int var1 = 321;
    double var2 = 1428.57;

    StringPrint(dest, "var1: %d, var2: %f, str: %s", var1, var2, str);
    Print("%s", dest);  //  var1: 321, var2: 1428.570000, str: hello world

    StringPrint(dest, "var2: %e", var2);
    Print("%s", dest);  //  var2: 1.428570e+03

    StringPrint(dest, "var2: %g", var2);
    Print("%s", dest);  //  var2: 1428.57
}
```

Requirement

Minimum supported version	iA Studio 0.23.1948.0

3.4 StringLen

Purpose

To get the length of a string.

Syntax

```
int StringLen(
    char *str
);
```

Parameter

str [in]

Return value

The length of a string (the terminating null character is not included).

Example

```
void main() {

    char str[] = "hello world";
    int len = StringLen(str);  //  len = 11
}
```

Requirement

Minimum supported version	iA Studio 0.23.2151.0

3.5 IsStringEqual

Purpose

To check whether the two strings are the same.

Syntax

```
int IsStringEqual(
    char *str1,
    char *str2
);
```

Parameter

str1 [in]

str2 [in]

Return value

It will return an **int** value **TRUE** (1) if the two strings are the same. Otherwise, it will return **FALSE** (0).

Example

```
void main() {

    char str1[] = "hello world";
    char str2[] = "hello world";
    char str3[] = "hello worldd";

    int is_equal = IsStringEqual(str1, str2);  //  is_equal = 1
    is_equal = IsStringEqual(str1, str3);  //  is_equal = 0
}
```

Requirement

Minimum supported version	iA Studio 0.23.2151.0

3.6 StrFindChar

Purpose

To locate the first occurrence of the character in a string.

Syntax

```
int StrFindChar(
    char *str,
    char character
);
```

Parameter

str [in]	The string.
character [in]	The character.

Return value

An offset value to the first occurrence of the character in the string.

If the character is not found, it will return -1.

Example

```
void main() {

    char str[] = "hello world";

    int offset = StrFindChar(str, 'h');  //  offset = 0
    offset = StrFindChar(str, 'l');  //  offset = 2
    offset = StrFindChar(str, 'z');  //  offset = -1
}
```

Requirement

Minimum supported version	iA Studio 0.23.2151.0

3.7 StrFindCharEx

Purpose

To locate the first occurrence of the character set or its complement in a string.

Syntax

```
int StrFindCharEx(
    char *str,
    char *char_set,
    int   complement_set
);
```

Parameter

str [in]	The string.
char_set [in]	The character set.
complement_set [in]	Locate option. False (**0**): Locate the character set True (**nonzero**): Locate the complement of the character set

Return value

An offset value to the first occurrence of the character set or its complement in a string.

If none of the characters is found, it will return -1.

Example

```
void main() {

    char str[] = "hello world";

    int offset = StrFindCharEx(str, "lo ", false);  //  offset = 2
    offset = StrFindCharEx(str, "lo ", true);  //  offset = 0
    offset = StrFindCharEx(str, "zx!c", false);  //  offset = -1
    offset = StrFindCharEx(str, "leh", true);  //  offset = 4
}
```

Requirement

Minimum supported version	iA Studio 0.25.2340.0

3.8 StrFindStr

Purpose

To locate the first occurrence of the substring in a string.

Syntax

```
int StrFindStr(
    char *str1,
    char *str2
);
```

Parameter

str1 [in]	The string.
str2 [in]	The substring.

Return value

An offset value to the first occurrence of the substring in the string.

If the substring is not found, it will return -1.

Example

```
void main() {

    char str[] = "hello world";

    int offset = StrFindStr(str, "hel");  //  offset = 0
    offset = StrFindStr(str, "wor");  //  offset = 6
    offset = StrFindStr(str, "wol");  //  offset = -1
}
```

Requirement

Minimum supported version	iA Studio 0.23.2151.0

3.9 StringCopy

Purpose

To copy a string.

Syntax

```
void StringCopy(
    char *destination,
    char *source
);
```

Parameter

destination [out]	A pointer to the buffer to store the result of the string.
source [in]	A string to be copied.

Return value

N/A

Remark

The source string's terminating null character will also be copied. To avoid overflow, the size of the array pointed by the destination string should be long enough to contain the source string (the terminating null character is included).

Example

```
void main() {

    char source[] = "hello world";
    char destination[80];

    StringCopy(destination, source);
    Print("%s", destination);  //  the output is hello world
}
```

Requirement

Minimum supported version	iA Studio 0.23.2151.0

3.10 StringCopyEx

Purpose

To copy a substring.

Syntax

```
void StringCopyEx(
    char *destination,
    char *source,
    int start_pos,
    int copy_len
);
```

Parameter

destination [out]	A pointer to the buffer to store the result of the string.
source [in]	A string to be copied.
start_pos [in]	The offset of the substring to be copied.
copy_len [in]	The length of the substring to be copied. If it is -1, all characters until the end of the string are copied.

Return value

N/A

Remark

The source string's terminating null character will also be copied. To avoid overflow, the size of the array pointed by the destination string should be long enough to contain the source string (the terminating null character is included).

Example

```
void main() {

    char source[] = "hello world";
    char destination[80];

    StringCopyEx(destination, source, 6, 3);
    Print("%s", destination);  //  the output is wor

    StringCopyEx(destination, source, 6, -1);
    Print("%s", destination);  //  the output is world

    StringCopyEx(destination, source, 0, -1);
    Print("%s", destination);  //  the output is hello world
}
```

Requirement

Minimum supported version	iA Studio 0.25.2340.0

3.11 StringCat

Purpose

To concatenate two strings.

Syntax

```
void StringCat(
    char *destination,
    char *source
);
```

Parameter

destination [out]	A pointer to the buffer to store the result of the string.
source [in]	A string to be appended.

Return value

N/A

Remark

Append a copy of the source string to the destination string. The terminating null character in the destination string is overwritten by the first character of the source string. To avoid overflow, the size of the array pointed by the destination string should be long enough to contain the source string (the terminating null character is included).

Example

```
void main() {

    char str[80] = "hello";
    StringCat(str, " world");
    Print("%s", str);  //  the output is hello world
}
```

Requirement

Minimum supported version	iA Studio 0.23.2151.0

3.12 StringCatEx

Purpose

To concatenate substrings.

Syntax

```
void StringCatEx(
    char *destination,
    char *source,
    int start_pos,
    int copy_len,
);
```

Parameter

destination [out]	A pointer to the buffer to store the result of the string.
source [in]	A string to be appended.
start_pos [in]	The offset of the substring to be copied.
copy_len [in]	The length of the substring to be copied. If it is -1, all characters until the end of the string are copied.

Return value

N/A

Remark

Append a copy of the source string to the destination string. The terminating null character in the destination string is overwritten by the first character of the source string. To avoid overflow, the size of the array pointed by the destination string should be long enough to contain the source string (the terminating null character is included).

Example

```
void main() {

    char source[] = "friendsmy ";
    char destination[80] = "hello ";

    StringCatEx(destination, source, 7, -1);
    Print("%s", destination);  //  the output is hello my

    //  now the destination is hello my
    StringCatEx(destination, source, 0, 7);
    Print("%s", destination);  //  the output is hello my friends
}
```

Requirement

Minimum supported version	iA Studio 0.25.2340.0

3.13 StringToDouble

Purpose

To convert a string to double type.

Syntax

```
double StringToDouble(
    char *str
);
```

Parameter

str [in]

Return value

The converted floating point value.

Example

```
void main() {
    double v = StringToDouble("1.234");  //  v = 1.234
}
```

Requirement

Minimum supported version	iA Studio 0.23.2164.0

3.14 MemoryCopy

Purpose

To copy bytes of data from the source memory to the destination memory.

Syntax

```
void MemoryCopy(
    void *destination,
    void *source,
    int byte_num
);
```

Parameter

destination [out]	A pointer to the buffer to store the result of the data.
source [in]	The data to be copied.
byte_num [in]	Number of bytes to be copied.

Return value

N/A

Example

```
void main() {

    int array1[5] = {1, 2, 3, 4, 5};
    int array2[5] = {11, 22, 33, 44, 55};
    int array3[5] = {345, 456, 567, 678, 789};

    MemoryCopy(array1, array2, sizeof(array2));
    //  now values in array1 are 11, 22, 33, 44, 55

    MemoryCopy(array1, array3, sizeof(int)*3);
    //  now values in array1 are 345, 456, 567, 44, 55

    MemoryCopy(&array1[3], &array3[3], sizeof(int)*2);
    //  now values in array1 are 345, 456, 567, 678, 789
}
```

Requirement

Minimum supported version	iA Studio 0.23.1948.0

3.15 MemorySet

Purpose

To set the first number of bytes of the destination memory to a specific value.

Syntax

```
void MemorySet(
    void *destination,
    int value,
    int byte_num
);
```

Parameter

destination [out]	A pointer to the buffer to store the result of the data.
value [in]	The value to be set. It is passed as an **int**, but the function fills the memory with the **char** conversion of this value.
byte_num [in]	Number of bytes to be set.

Return value

N/A

Example

```
void main() {

    int array1[5] = {1, 2, 3, 4, 5};

    MemorySet(array1, 0, sizeof(array1));
    //  now values in array1 are 0, 0, 0, 0, 0

    MemorySet(array1, 1, sizeof(int));
    //  now values in array1 are 16843009, 0, 0, 0, 0
    //  16843009 = 0x1111
}
```

Requirement

Minimum supported version	iA Studio 0.23.1948.0

3.16 IsMemoryEqual

Purpose

To check whether the two memory blocks are the same.

Syntax

```
void IsMemoryEqual(
    void *memory_ptr1,
    void *memory_ptr2,
    int byte_num
);
```

Parameter

memory_ptr1 [in]

memory_ptr2 [in]

byte_num [in] Number of bytes to be set.

Return value

It will return an **int** value **TRUE** (1) if the two memory blocks are the same.

Otherwise, it will return **FALSE** (0).

Example

```
void main() {

    int array1[5] = {1, 2, 3, 4, 5};
    int array2[5] = {1, 2, 3, 44, 55};
    int is_equal = false;
    is_equal = IsMemoryEqual(array1, array2, sizeof(array2));
    //  is_equal = false

    is_equal = IsMemoryEqual(array1, array2, sizeof(int)*3);
    //  is_equal = true
}
```

Requirement

Minimum supported version	iA Studio 0.23.1948.0

3.17 START_ASCII_AGENT

Purpose

To start a user-defined ascii command parser agent.

Syntax

```
START_ASCII_AGENT(
    parser_function
);
```

Parameter

parser_function [in] Name of parser function.
Parser function should be a binary function which allows ascii command to input and output a response.
In other words, its prototype is:
void (*ParserFunctionPrototype)(char *command, char *response)

Return value

N/A

Example

1.

```
void AsciiAgent(char *cmd, char *res) {

  for (int i = 0; ; ++i){

    if (cmd[i] != '\0 ') {
      res[i] = cmd[i] + 1;
    } else {
      res[i] = '\0 ';
      break;
    }
  }
}

void main() {
```

```
    START_ASCII_AGENT(AsciiAgent);
    //  Run it in any task and key some words in Message Window to get return ascii
    //  If the ascii command is “hello”, the response is “ifmmp”.
    //  If the ascii command is “asdf”, the response is “bteg”.
}
```

2.

```
void AsciiAgent(char *cmd, char *res) {

    char token_str[3][40];
    int token_start = 0;
    int token_num = 0;
    for (int i = 0; i < 3; ++i){
      int token_len = StrFindChar(&cmd[token_start], ' ');
      StringCopyEx(token_str[i], &cmd[token_start], 0, token_len);
      ++token_num;
      Print("%s", token_str[i]);

      if (token_len > 0) {
        int space_len = StrFindCharEx(&cmd[token_start + token_len], " ", true);
        token_start += token_len + space_len;
      } else {
        token_start = -1;
      }
      if (token_start < 0){
        break;
      }
    }
    Print("token number: %d", token_num);

    double token2_value = 0;
    double token3_value = 0;
    if (token_num >= 2){
      token2_value = StringToDouble(token_str[1]);
      if (token_num >= 3){
        token3_value = StringToDouble(token_str[2]);
      }
    }
```

```
    if (IsStringEqual(token_str[0], "ENABLE")){
      if (token_num == 2){
        Enable(token2_value);
      }
    }
    else if (IsStringEqual(token_str[0], "MOVEABS")){
      if (token_num == 3){
        MoveAbs(token2_value, token3_value);
      }
    } else if (IsStringEqual(token_str[0], "MOVEREL")){
      if (token_num == 3){
        MoveRel(token2_value, token3_value);
      }
    } else if (IsStringEqual(token_str[0], "STOP")){
      if (token_num == 2){
        Stop(token2_value);
      }
    }
}
void main() {
    Till(IsOperMode());
    START_ASCII_AGENT(AsciiAgent);
    //  the valid command is:
    //  ENABLE 0
    //  MOVEABS 0  0.05
    //  MOVEREL 0  0.01
    //  STOP 0
}
```

Requirement

Minimum supported version	iA Studio 0.23.1948.0

4. Math Functions

4. Math Functions 4-1
- 4.1 sin 4-2
- 4.2 cos 4-3
- 4.3 tan 4-4
- 4.4 asin 4-5
- 4.5 acos 4-6
- 4.6 atan 4-7
- 4.7 atan2 4-8
- 4.8 abs 4-9
- 4.9 fabs 4-10
- 4.10 ceil 4-11
- 4.11 floor 4-12
- 4.12 ldexp 4-13
- 4.13 exp 4-14
- 4.14 pow 4-15
- 4.15 log 4-16
- 4.16 log10 4-17
- 4.17 sqrt 4-18
- 4.18 cbrt 4-19
- 4.19 hypot 4-20

4.1 sin

Purpose

To get sine of x radians.

Syntax

```
double sin(
    double x
);
```

Parameter

x [in] A value expressing an angle in radians.
One radian is equivalent to 180/π degrees.

Return value

Sine of x radians.

Example

```
void main() {
    Print("sine of 30.0 degrees is %f.", sin(30.0 * PI / 180));
    //  Sine of 30.0 degrees is 0.5.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.2 cos

Purpose

To get cosine of x radians.

Syntax

```
double cos(
    double x
);
```

Parameter

x [in] A value expressing an angle in radians.
One radian is equivalent to 180/π degrees.

Return value

Cosine of x radians.

Example

```
void main() {
    Print("cosine of 60.0 degrees is %f.", cos(60.0 * PI / 180));
    //  Cosine of 60.0 degrees is 0.5.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.3 tan

Purpose

To get tangent of x radians.

Syntax

```
double tan(
    double x
);
```

Parameter

x [in] A value expressing an angle in radians.
One radian is equivalent to 180/π degrees.

Return value

Tangent of x radians.

Example

```
void main() {
    Print("tangent of 45.0 degrees is %f.", tan(45.0 * PI / 180));
    //  Tangent of 45.0 degrees is 1.0.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.4 asin

Purpose

To get arc sine of x. In trigonometric functions, arc sine is the inverse operation of sine.

Syntax

```
double asin(
    double x
);
```

Parameter

x [in] A value in the interval of [-1, +1].

Return value

Arc sine of x, in the interval of [-π/2, +π/2] radians.

One radian is equivalent to 180/π degrees.

Remark

If x is out of the interval, the return value cannot be defined.

Example

```
void main() {
    Print("arc sine of 0.5 is %f degrees", asin(0.5) * 180.0 / PI);
    //  Arc sine of 0.5 is 30.0 degrees.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.5 acos

Purpose

To get arc cosine of x. In trigonometric functions, arc cosine is the inverse operation of cosine.

Syntax

```
double acos(
    double x
);
```

Parameter

x [in] A value in the interval of [-1, +1].

Return value

Arc cosine of x, in the interval of [0, π] radians.

One radian is equivalent to 180/π degrees.

Remark

If x is out of the interval, the return value cannot be defined.

Example

```
void main() {
    Print("arc cosine of 0.5 is %f degrees", acos(0.5) * 180.0 / PI);
    //  Arc cosine of 0.5 is 60.0 degrees.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.6 atan

Purpose

To get arc tangent of x. In trigonometric functions, arc tangent is the inverse operation of tangent.

Syntax

```
double atan(
    double x
);
```

Parameter

x [in]

Return value

Arc tangent of x, in the interval of [-π/2, +π/2] radians.

One radian is equivalent to 180/π degrees.

Remark

Because of the sign ambiguity, the function cannot determine with certainty in which quadrant the angle falls only by its tangent value. Refer to Section 4.7 atan2 for an alternative that takes fractional parameters instead.

Example

```
void main() {
    Print("arc tangent of 1.0 is %f degrees", atan(1.0) * 180.0 / PI);
    //  Arc tangent of 1.0 is 45.0 degrees.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.7 atan2

Purpose

To get arc tangent of y/x.

Syntax

```
double atan2(
    double y,
    double x
);
```

Parameter

y [in]	A value which indicates the proportion of Y coordinate.
x [in]	A value which indicates the proportion of X coordinate.

Return value

Arc tangent of y/x, in the interval of [-π, +π] radians.

One radian is equivalent to 180/π degrees.

Example

```
void main() {
    Print("arc tangent for (x=-10, y=10) is %f degrees",
            atan2(10, -10) * 180.0 / PI);
    //  Arc tangent for (x=-10, y=10) is 135 degrees.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.8 abs

Purpose

To get the absolute value of an integer x: | x |.

Syntax

```
int abs(
    int x
);
```

Parameter

x [in] An integer.

Return value

The absolute value of an integer x.

Example

```
void main() {
    Print("absolute value of -3591 is %d.", abs(-3591));
    //  The absolute value of -3591 is 3591.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.9 fabs

Purpose

To get the absolute value of a double-precision floating point x: | x |.

Syntax

```
double fabs(
    double x
);
```

Parameter

x [in] A double-precision floating point.

Return value

The absolute value of a double-precision floating point x.

Example

```
void main() {
    Print("absolute value of -35.91 is %d.", fabs(-35.91));
    //  The absolute value of -35.91 is 35.91.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.10 ceil

Purpose

To round x up to an integer.

Syntax

```
double ceil(
    double x
);
```

Parameter

x [in]

Return value

The smallest integer that is not less than x.

Example

```
void main() {
    Print("ceil of 2.3 is %g", ceil(2.3));  //  Ceil of 2.3 is 3.0.
    Print("ceil of 3.8 is %g", ceil(3.8));  //  Ceil of 3.8 is 4.0.
    Print("ceil of -2.3 is %g", ceil(-2.3));  //  Ceil of -2.3 is -2.0.
    Print("ceil of -3.8 is %g", ceil(-3.8));  //  Ceil of -3.8 is -3.0.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.11 floor

Purpose

To round x down to an integer.

Syntax

```
double floor(
    double x
);
```

Parameter

x [in]

Return value

The largest integer that is not greater than x.

Example

```
void main() {
    Print("floor of 2.3 is %g", floor(2.3));  //  Floor of 2.3 is 2.0.
    Print("floor of 3.8 is %g", floor(3.8));  //  Floor of 3.8 is 3.0.
    Print("floor of -2.3 is %g", floor(-2.3));  //  Floor of -2.3 is -3.0.
    Print("floor of -3.8 is %g", floor(-3.8));  //  Floor of -3.8 is -4.0.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.12 ldexp

Purpose

To get the value of x multiplied by 2 raised to the power of y: x * 2^y.

Syntax

```
double ldexp(
    double x,
    int    y
);
```

Parameter

x [in]

y [in] An integer.

Return value

x * 2^y.

If the magnitude of the result is too large, it will return the largest representable double value.

Example

```
void main() {
    Print("0.95 * 2^4 = %f", ldexp(0.95, 4 ));  //  0.95 * 2^4 = 15.20
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.13 exp

Purpose

To get the value of e raised to the power of x: e^x.

e is the base of the natural logarithm. It is approximately equal to 2.71828.

Syntax

```
double exp(
    double x
);
```

Parameter

x [in]

Return value

e^x.

If the magnitude of the result is too large, it will return the largest representable double value.

Example

```
void main() {
    Print("The exponential value of 5.0 is %f.", exp(5.0));
    //  The exponential value of 5.0 is 148.413159.
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.14 pow

Purpose

To get the value of x raised to the power of y: x^y.

Syntax

```
double pow(
    double x,
    double y
);
```

Parameter

x [in]

y [in]

Return value

x^y.

If the magnitude of the result is too large, it will return the largest representable double value.

Remark

(1) If x is finite negative and y is finite but not an integer, the return value cannot be defined.

(2) If both x and y are zero, the return value cannot be defined.

(3) If x is zero and y is negative, the return value cannot be defined.

Example

```
void main() {
    Print("7.0 ^ 3.0 = %f", pow(7.0, 3.0));  //  7.0 ^ 3.0 = 343.0
    Print("4.73 ^ 12.0 = %f", pow(4.73, 12.0));  //  4.73 ^ 12.0 = 125410439.217423
    Print("32.01 ^ 1.54 = %f", pow(32.01, 1.54));  //  32.01 ^ 1.54 = 208.036691
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.15 log

Purpose

To get the natural logarithm (base-e) of x.

Syntax

```
double log(
    double x
);
```

Parameter

x [in]

Return value

The natural logarithm (base-e) of x.

Remark

(1) If x is negative or zero, the return value cannot be defined.

(2) Refer to Section 4.16 log10 for the common logarithm (base-10).

Example

```
void main() {
    Print("log(5.5) = %f", log(5.5));  //  log(5.5) = 1.704748
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.16 log10

Purpose

To get the logarithm (base-10) of x.

Syntax

```
double log10(
    double x
);
```

Parameter

x [in]

Return value

The logarithm (base-10) of x.

Remark

If x is negative or zero, the return value cannot be defined.

Example

```
void main() {
    Print("log10(1000.0) = %f", log10(1000.0));  //  log10(1000.0) = 3.0
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.17 sqrt

Purpose

To get square root of x.

Syntax

```
double sqrt(
    double x
);
```

Parameter

x [in]

Return value

Square root of x.

Remark

If x is negative, the return value cannot be defined.

Example

```
void main() {
    Print("sqrt(1024.0) = %f", sqrt(1024.0));  //  sqrt(1024.0) = 32.0
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.18 cbrt

Purpose

To get cubic root of x.

Syntax

```
double cbrt(
    double x
);
```

Parameter

x [in]

Return value

Cubic root of x.

Example

```
void main() {
    Print("cbrt (27.0) = %f", cbrt(27.0));  //  cbrt (27.0) = 3.0
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

4.19 hypot

Purpose

To get the hypotenuse of a right triangle whose legs are x and y.

Syntax

```
double hypot(
    double x,
    double y
);
```

Parameter

x [in]	One leg of a right triangle.
y [in]	The other leg of a right triangle.

Return value

The square root of $(x^2 + y^2)$.

If the magnitude of the result is too large, it will return the largest representable double value.

Example

```
void main() {
    Print("hypot(3.0, 4.0) = %f.", hypot(3.0, 4.0));
    //  hypot(3.0, 4.0) = 5
}
```

Requirement

Minimum supported version	iA Studio 0.23.2005.0

5. Axis Functions

5. Axis Functions 5-1
5.1 Overview 5-3
5.1.1 Axis variables 5-3
5.1.1.1 Motion variables 5-3
5.1.1.2 Profile generator variables 5-5
5.1.2 Axis faults 5-6
5.1.3 Axis state diagram 5-6
5.2 Axis Motion Control 5-7
5.2.1 Enable 5-7
5.2.2 Disable 5-8
5.2.3 Reset 5-9
5.2.4 MoveAbs 5-10
5.2.5 MoveRel 5-11
5.2.6 MoveVel 5-12
5.2.7 Stop 5-13
5.3 Axis Setting 5-14
5.3.1 GetMaxVel 5-14
5.3.2 SetVel 5-15
5.3.3 GetMaxAcc 5-16
5.3.4 SetAcc 5-17
5.3.5 GetMaxDec 5-18
5.3.6 SetDec 5-19
5.3.7 GetSWRL 5-20
5.3.8 SetSWRL 5-21
5.3.9 GetSWLL 5-22
5.3.10 SetSWLL 5-23
5.3.11 GetSMTime 5-24
5.3.12 SetSMTime 5-25
5.3.13 GetMoveTime 5-26
5.3.14 GetSettlingTime 5-27
5.3.15 SetPos 5-28
5.3.16 GetPosFb 5-29
5.3.17 GetPosOffset 5-30
5.3.18 GetPosErr 5-31
5.3.19 GetRefPos 5-32
5.3.20 GetRefVel 5-33

5.3.21 GetRefAcc....5-34
5.3.22 GetPosOut....5-35
5.3.23 GetVelOut....5-36
5.3.24 GetAccOut....5-37
5.3.25 IgnoreHWL....5-38
5.3.26 IgnoreSWL....5-39
5.3.27 GetAxisNum....5-40
5.4 Axis Status....5-41
5.4.1 IsEnabled....5-41
5.4.2 IsMoving....5-42
5.4.3 IsInPos....5-43
5.4.4 IsErrorStop....5-44
5.4.5 IsGantry....5-45
5.4.6 IsGrouped....5-46
5.4.7 IsSync....5-47
5.4.8 IsHWLL....5-48
5.4.9 IsHWRL....5-49
5.4.10 IsSWLL....5-50
5.4.11 IsSWRL....5-51
5.4.12 IsCompActive....5-52

5.1 Overview

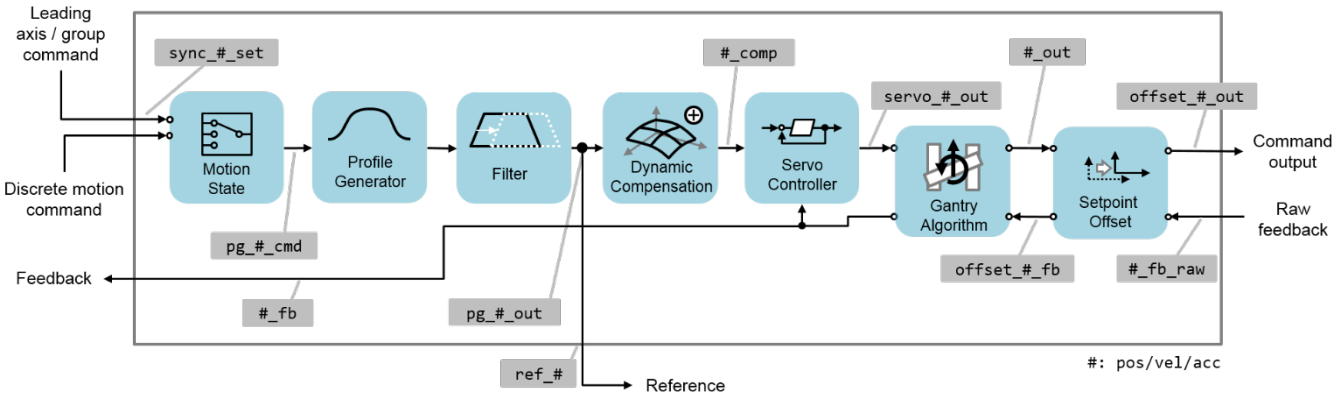


Figure 5.1.1

5.1.1 Axis variables

5.1.1.1 Motion variables

Common axis variables are given in Table 5.1.1.1.1. To access the variables in iA Studio, add prefix **"hcv.axis[&].(variable name)"** in the edit box. "&" stands for the axis index.

Table 5.1.1.1.1

Name	Type	Unit	Description
sync_pos_set	double	meter or radian	Synchronized position set-point. It is the target position to be followed when the axis is in synchronized motion (such as axis group, camming or gearing operations).
pg_pos_cmd	double	meter or radian	Profile generator position command. It is the target position to be followed when the axis is in discrete motion (point-to-point).
ref_pos	double	meter or radian	Reference position. It is the position set-point generated from the profile generator according to predefined motion profile.
ref_vel	double	m/s or rad/s	Reference velocity. It is the velocity set-point generated from the profile generator according to predefined motion profile.
ref_acc	double	m/s^2 or rad/s^2	Reference acceleration. It is the acceleration set-point generated from the profile generator according to predefined motion profile.
pos_comp_set	double	meter or radian	Position compensation set-point. It is the output of error map of dynamic error compensation function. If the function is disabled, it will be zero.

pos_comp	double	meter or radian	Compensated position. It is the position command corrected by dynamic error compensation.
pos_offset	double	meter or radian	Position offset. The default value is zero. If users set a new axis position without moving the motor, it will be nonzero.
pos_out	double	meter or radian	Position output. It is the axis position command without position offset.
offset_pos_out	double	meter or radian	Position output with offset. It is the final calculated axis position command with position offset. The value will be converted to the unit “count” and be transmitted to the corresponding slave.
pos_fb_raw	double	meter or radian	Raw position feedback. It is the feedback position read from the slave and converted from the encoder counts.
offset_pos_fb	double	meter or radian	Offsetted position feedback. It is the feedback position with position offset.
pos_fb	double	meter or radian	Position feedback. It is in an axis coordinate system.
pos_err	double	meter or radian	Following position error. It is the difference between position output and raw position feedback.

5.1.1.2 Profile generator variables

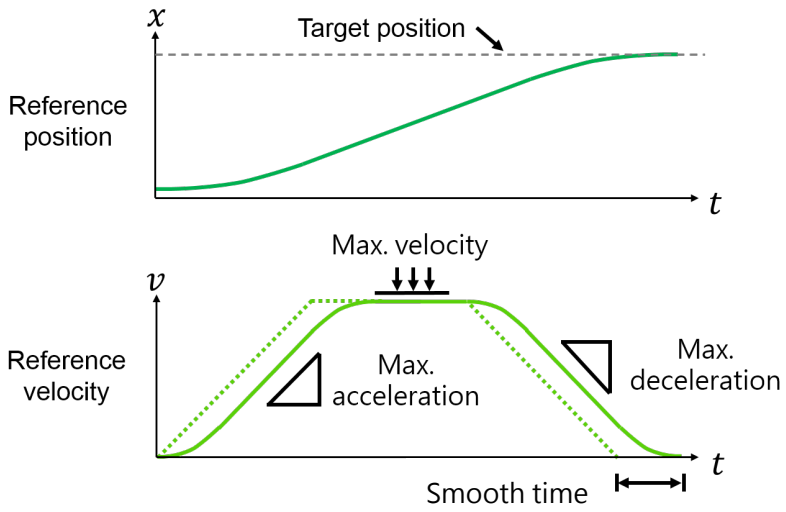


Figure 5.1.1.2.1

PG (profile generator) related variables are given in Table 5.1.1.2.1.

Table 5.1.1.2.1

Name	Type	Unit	Description
max_vel	double	m/s or rad/s	Maximum profile velocity. Not necessarily reached.
max_acc	double	m/s^2 or rad/s^2	Maximum profile acceleration. Not necessarily reached.
max_dec	double	m/s^2 or rad/s^2	Maximum profile deceleration. Not necessarily reached.
sm_time	double	second	Smooth time. Its input range is from 0.0 to 0.5. Increase the value to reduce mechanical vibration during motion. Note that the value would affect total motion time.

5.1.2 Axis faults

Variables: **fault_status**, **fault_resp**

Table 5.1.2.1

Bit	Name	Description	Default Response
0	Error Stop	Axis at “error stop” state	N/A
1	Drive fault	Slave drive fault	Controller disables the axis.
2	Position error	Position error exceeds protection limit	Controller disables the axis
3	Hardware right limit	Axis hardware right limit triggered	Controller stops the motion.
4	Hardware left limit	Axis hardware left limit triggered	Controller stops the motion.
5	Software right limit	Axis software right limit triggered	Controller stops the motion.
6	Software left limit	Axis software left limit triggered	Controller stops the motion.

5.1.3 Axis state diagram

The finite-state machine and the corresponding commands for a single axis are described in Figure 5.1.3.1.

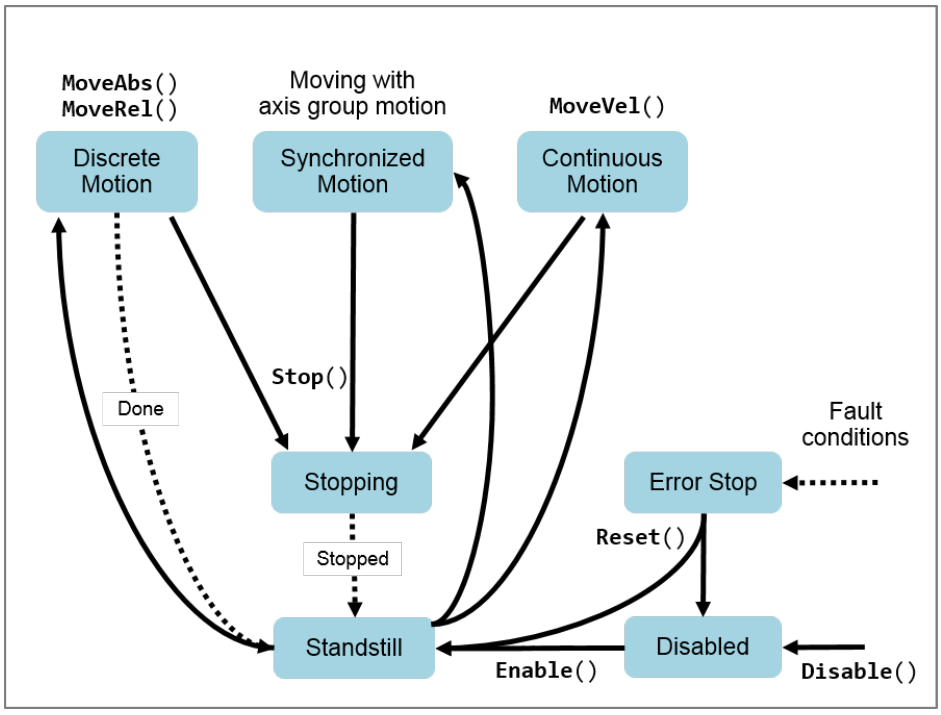


Figure 5.1.3.1

5.2 Axis Motion Control

5.2.1 Enable

Purpose

To enable an axis.

Syntax

```
int Enable(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.2.2 Disable

Purpose

To disable an axis.

Syntax

```
int Disable(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

The motion queue of the axis will be cleared.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.2.3 Reset

Purpose

To reset an axis when it is at the “error stop” state.

Syntax

```
int Reset(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.2.4 MoveAbs

Purpose

To move the axis to an absolute target position.

Syntax

```
int MoveAbs(
    int    axis_id,
    double pos
);
```

Parameter

axis_id [in]	Axis index.
pos [in]	The value of an absolute target position. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.2.5 MoveRel

Purpose

To move the axis by a relative distance.

Syntax

```
int MoveRel(
    int    axis_id,
    double rel_dist
);
```

Parameter

axis_id [in]	Axis index.
rel_dist [in]	The value to a relative distance. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.2.6 MoveVel

Purpose

To start a never-ending motion at a specific velocity.

Syntax

```
int MoveVel(
    int    axis_id,
    double vel
);
```

Parameter

axis_id [in]	Axis index.
vel [in]	The value of a specific velocity. It can be either positive or negative to indicate the direction of the motion. Parameter unit: m/s or rad/s

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.2.7 Stop

Purpose

To stop the motion of an axis.

Syntax

```
int Stop(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

The motion queue of the axis will be cleared.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3 Axis Setting

5.3.1 GetMaxVel

Purpose

To get the maximum profile velocity of an axis.

Syntax

```
double GetMaxVel(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The maximum profile velocity of the axis.

Unit: m/s or rad/s

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.2 SetVel

Purpose

To set the maximum profile velocity of an axis.

Syntax

```
int SetVel(
    int    axis_id,
    double vel
);
```

Parameter

axis_id [in]	Axis index.
vel [in]	The new maximum profile velocity of an axis. Parameter unit: m/s or rad/s Input range: nonzero positive value

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.3 GetMaxAcc

Purpose

To get the maximum profile acceleration of an axis.

Syntax

```
double GetMaxAcc(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The maximum profile acceleration of the axis.

Unit: m/s^2 or rad/s^2

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.4 SetAcc

Purpose

To set the maximum profile acceleration of an axis.

Syntax

```
int SetAcc(
    int    axis_id,
    double acc
);
```

Parameter

axis_id [in]	Axis index.
acc [in]	The new maximum profile acceleration of an axis. Parameter unit: m/s^2 or rad/s^2 Input range: nonzero positive value

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.5 GetMaxDec

Purpose

To get the maximum profile deceleration of an axis.

Syntax

```
double GetMaxDec(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The maximum profile deceleration of the axis.

Unit: m/s^2 or rad/s^2

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.6 SetDec

Purpose

To set the maximum profile deceleration of an axis.

Syntax

```
int SetDec(
    int    axis_id,
    double dec
);
```

Parameter

axis_id [in]	Axis index.
dec [in]	The new maximum profile deceleration of an axis. Parameter unit: m/s^2 or rad/s^2 Input range: nonzero positive value

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.7 GetSWRL

Purpose

To get the software right limit position of an axis.

Syntax

```
double GetSWRL(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The software right limit position of an axis.

Unit: meter or radian

Requirement

Minimum supported version	iA Studio 1.1.3761.0

5.3.8 SetSWRL

Purpose

To set the software right limit position of an axis.

Syntax

```
int SetSWRL(
    int    axis_id,
    double right_limit_pos
);
```

Parameter

axis_id [in]	Axis index.
right_limit_pos [in]	The new software right limit position of an axis. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

5.3.9 GetSWLL

Purpose

To get the software left limit position of an axis.

Syntax

```
double GetSWLL(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The software left limit position of an axis.

Unit: meter or radian

Requirement

Minimum supported version	iA Studio 1.1.3761.0

5.3.10 SetSWLL

Purpose

To set the software left limit position of an axis.

Syntax

```
int SetSWLL(
    int    axis_id,
    double left_limit_pos
);
```

Parameter

axis_id [in]	Axis index.
left_limit_pos [in]	The new software left limit position of an axis. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

5.3.11 GetSMTime

Purpose

To get the profile smooth time of an axis.

Syntax

```
double GetSMTime(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The profile smooth time of the axis.

Unit: second

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.12 SetSMTime

Purpose

To set the profile smooth time of an axis.

Syntax

```
int SetSMTime(
    int    axis_id,
    double smooth_time
);
```

Parameter

axis_id [in]	Axis index.
smooth_time [in]	The new profile smooth time of an axis. Parameter unit: second Input range: 0.0~0.5

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.13 GetMoveTime

Purpose

To get the move time of an axis.

Syntax

```
double GetMoveTime(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The move time of the axis.

Unit: second

Requirement

Minimum supported version	iA Studio 0.24.2310.0

5.3.14 GetSettlingTime

Purpose

To get the settling time of an axis.

Syntax

```
double GetSettlingTime(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The settling time of the axis.

Unit: second

Requirement

Minimum supported version	iA Studio 0.24.2310.0

5.3.15 SetPos

Purpose

To designate the value of the axis' current position.

Syntax

```
int SetPos(
    int    axis_id,
    double pos
);
```

Parameter

axis_id [in]	Axis index.
pos [in]	The value of the axis' current position. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the axis is in the sync mode, added to an axis group, or at the error state.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.16 GetPosFb

Purpose

To get the feedback position of an axis.

Syntax

```
double GetPosFb(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The feedback position of the axis.

Unit: meter or radian

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.17 GetPosOffset

Purpose

To get the position offset of an axis.

Syntax

```
double GetPosOffset(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The position offset of the axis.

Unit: meter or radian

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.18 GetPosErr

Purpose

To get the position error of an axis.

Syntax

```
double GetPosErr(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The position error of the axis.

Unit: meter or radian

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.19 GetRefPos

Purpose

To get the reference position of an axis.

Syntax

```
double GetRefPos(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The reference position of the axis.

Unit: meter or radian

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.20 GetRefVel

Purpose

To get the reference velocity of an axis.

Syntax

```
double GetRefVel(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The reference velocity of the axis.

Unit: m/s or rad/s

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.21 GetRefAcc

Purpose

To get the reference acceleration of an axis.

Syntax

```
double GetRefAcc(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The reference acceleration of the axis.

Unit: m/s^2 or rad/s^2

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.22 GetPosOut

Purpose

To get the position command output of an axis sent by the controller to the slave drive.

Syntax

```
double GetPosOut(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The position command output of the axis.

Unit: meter or radian

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.23 GetVelOut

Purpose

To get the velocity command output of an axis sent by the controller to the slave drive.

Syntax

```
double GetVelOut(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The velocity command output of the axis.

Unit: m/s or rad/s

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.24 GetAccOut

Purpose

To get the acceleration command output of an axis sent by the controller to the slave drive.

Syntax

```
double GetAccOut(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The acceleration command output of the axis.

Unit: m/s^2 or rad/s^2

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.25 IgnoreHWL

Purpose

Ignore the warning messages of hardware limit protection.

Syntax

```
int IgnoreHWL(
    int axis_id,
    int cmd
);
```

Parameter

axis_id [in]	Axis index.
cmd [in]	Set it as “1” to ignore the messages. Set it as “0” to restore the messages (default).

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.26 IgnoreSWL

Purpose

Ignore the warning messages of software limit protection.

Syntax

```
int IgnoreSWL(
    int axis_id,
    int cmd
);
```

Parameter

axis_id [in]	Axis index.
cmd [in]	Set it as “1” to ignore the messages. Set it as “0” to restore the messages (default).

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.3.27 GetAxisNum

Purpose

To get the number of the axes connected to the controller.

Syntax

```
int GetAxisNum(
    void
);
```

Return value

The number of the axes connected to the controller.

Requirement

Minimum supported version	iA Studio 0.23.2096.0

5.4 Axis Status

5.4.1 IsEnabled

Purpose

To query the “enable” status of an axis.

Syntax

```
int IsEnabled(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is enabled. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.2 IsMoving

Purpose

To query the “moving” status of an axis. If the axis is moving, PG (profile generator) continues outputting new positions.

Syntax

```
int IsMoving(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is moving. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.3 IsInPos

Purpose

To query the “in-position” status of an axis. If the axis is in-position, the position error is kept within an error window (target radius) for a specific duration (debounce time).

Syntax

```
int IsInPos(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is in-position. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.4 IsErrorStop

Purpose

To query whether the axis is at the “error stop” state.

Syntax

```
int IsErrorStop(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is at the “error stop” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1919.0

5.4.5 IsGantry

Purpose

To query whether the axis is at the “gantry” state.

Syntax

```
int IsGantry(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is at the “gantry” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.6 IsGrouped

Purpose

To query whether the axis is grouped into an axis group.

Syntax

```
int IsGrouped(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is at the “grouped” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.2096.0

5.4.7 IsSync

Purpose

To query whether the axis is at the “sync” state. If the axis is at the “sync” state, the axis follows the master’s command.

Syntax

```
int IsSync(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is at the “sync” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.8 IsHWLL

Purpose

To query whether the axis reaches the hardware left limit.

Syntax

```
int IsHWLL(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) If the axis is at the “HWLL” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.9 IsHWRL

Purpose

To query whether the axis reaches the hardware right limit.

Syntax

```
int IsHWRL(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) If the axis is at the “HWRL” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.10 IsSWLL

Purpose

To query whether the axis reaches the software left limit.

Syntax

```
int IsSWLL(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) If the axis is at the “SWLL” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.11 IsSWRL

Purpose

To query whether the axis reaches the software right limit.

Syntax

```
int IsSWRL(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) If the axis is at the “SWRL” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

5.4.12 IsCompActive

Purpose

To query whether the compensation function is active.

Syntax

```
int IsCompActive(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is at the “compensation active” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

6. Synchronized Motion Functions

6. Synchronized Motion Functions 6-1
6.1 Overview 6-2
6.1.1 Synchronized motion variables 6-3
6.1.2 Example 6-3
6.2 EnableGear 6-5
6.3 DisableGear 6-6
6.4 GearIn 6-7
6.5 GearOut 6-8

6.1 Overview

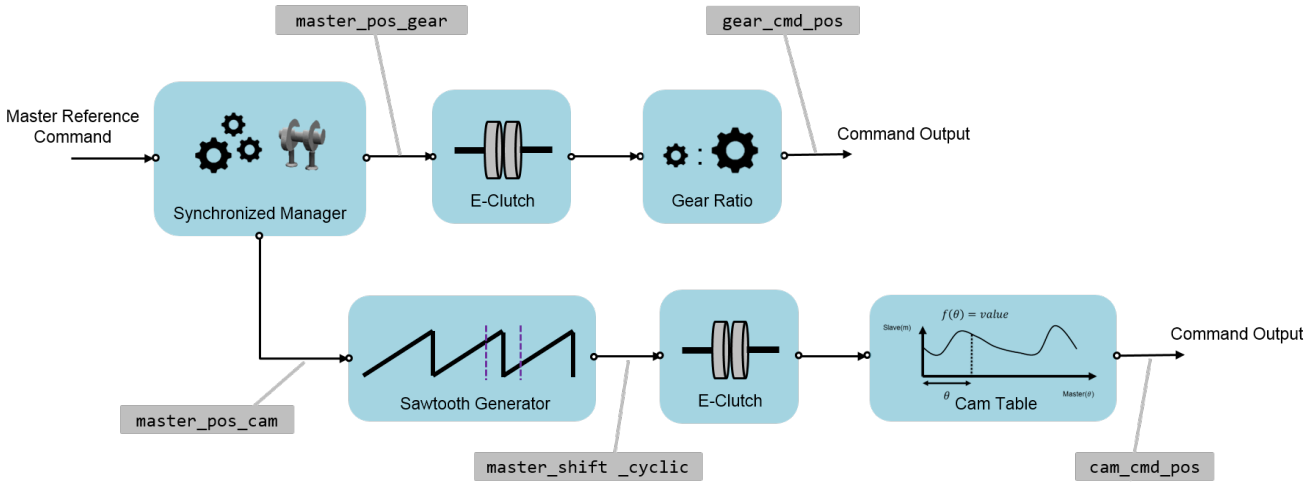


Figure 6.1.1

Users can define synchronized motion between one axis and another. Master axis, a leading axis, will generate position command first, and then slave axis will refer to master axis based on the motion configuration. If master-slave relationship is constant, the motion is electronic gearing. On the other hand, if slave axis needs to follow a pattern, the motion is electronic camming. In Figure 6.1.2, axis 0 acts as master axis, leading axis 1, 2, 3 and 4. Axis 1, 2 and 3 adopt electronic gearing, while axis 4 adopts electronic camming.

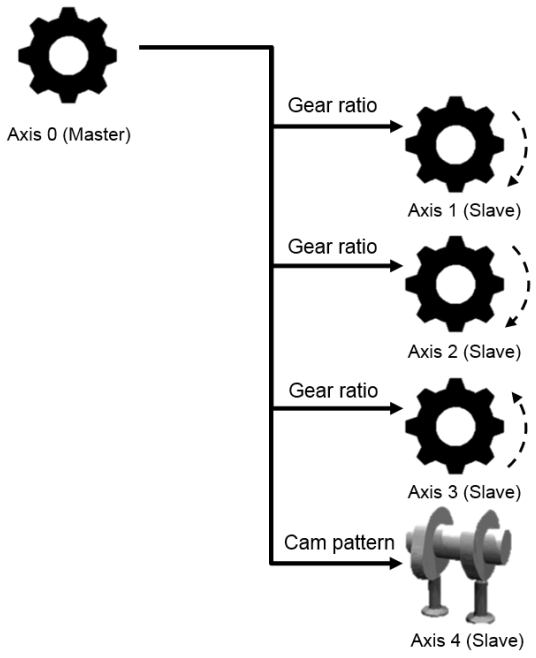


Figure 6.1.2

6.1.1 Synchronized motion variables

Common synchronized motion variables are given in Table 6.1.1.1. To access the variables in iA Studio, add prefix **"hcv.axis[&].(variable name)"** in the edit box. "&" stands for the axis index.

Table 6.1.1.1

Name	Type	Unit	Description
master_pos_gear	double	meter or radian	Position command from master axis.
gear_cmd_pos	double	meter or radian	Slave axis outputs position command.
gear_ratio	double	meter or radian	Gear ratio.

6.1.2 Example

```
void main()
{

    double target = 0.1;
    double Gear_ratio[4]={1.0, 2.0, 4.0, -1.0};
    int master = 0;
    int slave[4]={1, 2, 3, 4};

    Enable(master);
    Enable(slave[0]);
    Enable(slave[1]);
    Enable(slave[2]);
    Enable(slave[3]);
    Till(IsEnabled(slave[0])&&IsEnabled(slave[1])&&
    IsEnabled(slave[2])&&IsEnabled(slave[3])&&IsEnabled(master))

    //  Couple two axes in a master-slave relationship
    EnableGear(master, slave[0]);
    EnableGear(master, slave[1]);
    EnableGear(master, slave[2]);
    EnableGear(master, slave[3]);
```

```
    //  Change slave axis' state from disengaged to engaged
    GearIn(master, slave[0], Gear_ratio[0]);
    GearIn(master, slave[1], Gear_ratio[1]);
    GearIn(master, slave[2], Gear_ratio[2]);
    GearIn(master, slave[3], Gear_ratio[3]);

    MoveAbs(master, target);
    Till(IsInPos(master));

    //  Change slave axis' state from engaged to disengaged
    GearOut(slave[0]);
    GearOut(slave[1]);
    GearOut(slave[2]);
    GearOut(slave[3]);

}
```

6.2 EnableGear

Purpose

To couple two axes in a master-slave relationship.

Syntax

```
int EnableGear(
    int axis_master_id,
    int axis_slave_id
);
```

Parameter

axis_master_id [in] Master axis index.

axis_slave_id [in] Slave axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is applicable only when both axes are enabled.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

6.3 DisableGear

Purpose

To uncouple two axes from the master-slave relationship to two independent axes.

Syntax

```
int DisableGear(
    int axis_slave_id
);
```

Parameter

axis_slave_id [in] Slave axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

6.4 GearIn

Purpose

To change slave axis' state from disengaged to engaged.

Syntax

```
int GearIn(
    int axis_master_id,
    int axis_slave_id,
    double gear_ratio
);
```

Parameter

axis_master_id [in]	Master axis index.
axis_slave_id [in]	Slave axis index.
gear_ratio[in]	Value of gear ratio.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is applicable only when both axes are enabled.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

6.5 GearOut

Purpose

To change slave axis' state from engaged to disengaged.

Syntax

```
int GearOut(
    int axis_slave_id
);
```

Parameter

axis_slave_id [in] Slave axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

7. Gantry Functions

7. Gantry Functions 7-1
7.1 Overview 7-2
7.1.1 Gantry pair setup example 7-3
7.2 EnableGantryPair 7-4
7.3 DisableGantryPair 7-5

7.1 Overview

The gantry configuration transforms a pair of right-hand-side (RHS) axis and left-hand-side (LHS) axis into a pair of imaginary linear axis and yaw axis.

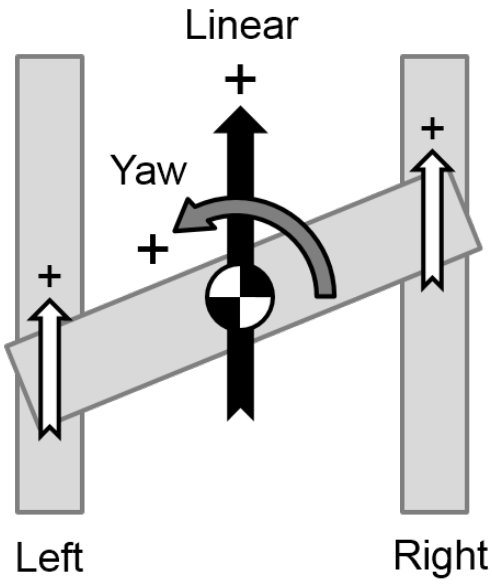


Figure 7.1.1

Linear axis drives both RHS and LHS axes in the same direction, while yaw axis generates rotary motion command.

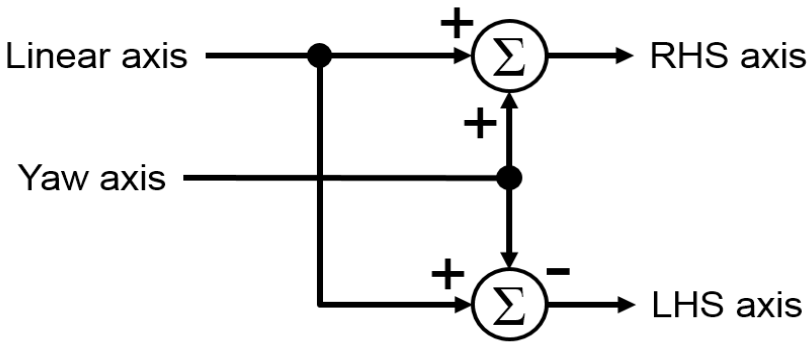


Figure 7.1.2

Feedback position of **Linear axis** = $\frac{(\mathrm{RHS}+\mathrm{LHS})}{2}$

Feedback position of **Yaw axis** = $\frac{(\mathrm{RHS}-\mathrm{LHS})}{2}$

7.1.1 Gantry pair setup example

The way to set up a gantry pair is shown in the following HMPL task.

```
void main() {

    int axis_0 = 0;  //  user variable definition
    int axis_1 = 1;

    DisableGantryPair(axis_0);  //  Disable the existing gantry settings
    Till(!IsGantry(axis_0) && !IsGantry(axis_1));

    Enable(axis_0);
    Till(IsEnabled(axis_0));
    Disable(axis_0);
    Till(!IsEnabled(axis_0));

    Enable(axis_1);
    Till(IsEnabled(axis_1));
    Disable(axis_1);
    Till(!IsEnabled(axis_1));

    EnableGantryPair(axis_0, axis_1);
    Enable(axis_0);

    Till(IsEnabled(axis_0) && IsEnabled(axis_1));
    Till(IsGantry(axis_0) && IsGantry(axis_1));
}
```

7.2 EnableGantryPair

Purpose

To set up a gantry pair.

Syntax

```
int EnableGantryPair(
    int lhs_axis_id,
    int rhs_axis_id
);
```

Parameter

lhs_axis_id [in] Left-hand-side axis index.

rhs_axis_id [in] Right-hand-side axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is applicable only when both axes are enabled.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

7.3 DisableGantryPair

Purpose

To split a gantry pair.

Syntax

```
int DisableGantryPair(
    int axis_id
);
```

Parameter

axis_id [in] Either axis index in a gantry pair.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is applicable only when both axes are disabled.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

(This page is intentionally left blank.)

8. Group Functions

8. Group Functions 8-1
8.1 Overview 8-2
8.1.1 Group motion variables 8-3
8.1.2 Group state diagram 8-4
8.1.3 Example 8-5
8.1.3.1 Basic group setup 8-5
8.1.3.2 Advanced group setup and velocity blending 8-6
8.2 Group Motion Control 8-8
8.2.1 EnableGroup 8-8
8.2.2 DisableGroup 8-9
8.2.3 Basic motion command 8-10
8.2.3.1 LineAbs2D 8-10
8.2.3.2 LineAbs3D 8-12
8.2.3.3 LineRel2D 8-13
8.2.3.4 LineRel3D 8-15
8.2.3.5 Arc2D 8-16
8.2.3.6 Circle2D 8-18
8.2.4 Advanced motion command 8-20
8.2.4.1 Bezier 8-20
8.2.4.2 LinAbs 8-23
8.2.4.3 LinRel 8-25
8.2.4.4 CircAbs 8-27
8.2.5 StopGroup 8-29
8.3 Group Setting 8-30
8.3.1 AddAxisToGrp 8-30
8.3.2 RemoveAxisFromGrp 8-31
8.3.3 SetupGroup 8-32
8.3.4 UngrpAllAxes 8-33
8.3.5 GetGroupID 8-34
8.3.6 SetGrpMotionProfile 8-35
8.3.7 SetGrpKin 8-37
8.3.8 GroupReset 8-38
8.4 Group Status Functions 8-39
8.4.1 IsGrpEnabled 8-39
8.4.2 IsGrpMoving 8-40
8.4.3 IsGrpInPos 8-41

8.1 Overview

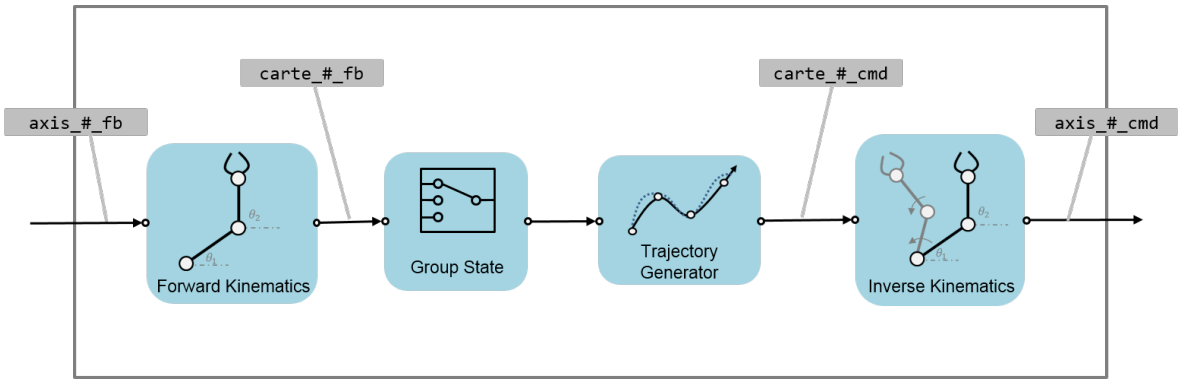


Figure 8.1.1

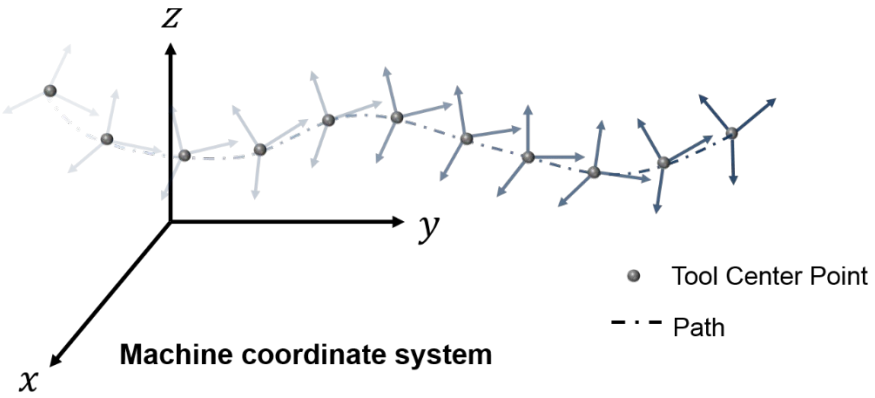


Figure 8.1.2

8.1.1 Group motion variables

Common axis group variables are given in Table 8.1.1.1. To access the variables in iA Studio, add prefix **"hcv.axis_group[&].(variable name)"** in the edit box. "&" stands for the axis group index.

Table 8.1.1.1

Name	Type	Unit	Description
grp_tcp_vel_norm	double	m/s	Tool center point velocity vector norm. An absolute value corresponding to the linear velocity of tool center point in machine coordinated system (MCS).
grp_tcp_acc_norm	double	m/s^2	Tool center point acceleration vector norm. An absolute value corresponding to the linear acceleration of tool center point in machine coordinated system (MCS).
axis_pos_cmd	double	meter or radian	Axis space position command. An array containing the position commands in axis coordinate system (ACS). The values derived from inverse kinematics will be adopted by individual axes as their target set-points.
axis_vel_cmd	double	m/s or rad/s	Axis space velocity command. An array containing the velocity commands in axis coordinate system (ACS). The values derived from inverse kinematics will be adopted by individual axes as their target set-points.
axis_acc_cmd	double	m/s^2 or rad/s^2	Axis space acceleration command. An array containing the acceleration commands in axis coordinate system (ACS). The values derived from inverse kinematics will be adopted by individual axes as their target set-points.

8.1.2 Group state diagram

The finite-state machine and the corresponding commands for an axis group are described in Figure 8.1.2.1.

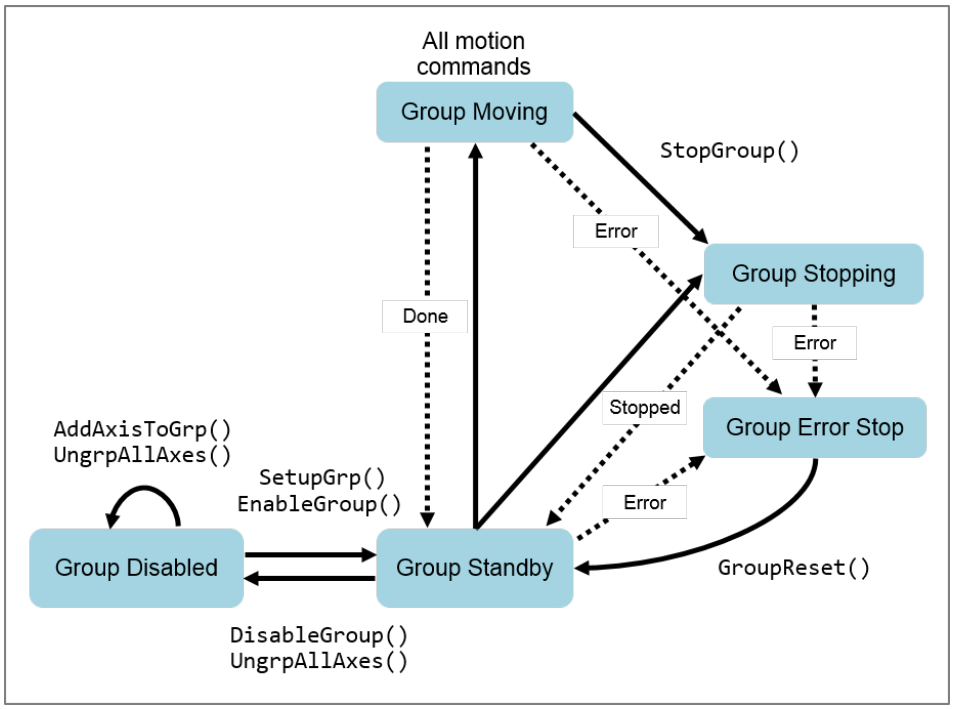


Figure 8.1.2.1

8.1.3 Example

8.1.3.1 Basic group setup

The way to create, enable an axis group and execute coordinated motion commands is shown in the following HMPL task.

```
int main() {

    int gid = 0;  //  group index

    UngrpAllAxes(gid);
    //  Remove all axes from the existing axis group and disable it (optional)

    Enable(0);
    Enable(1);

    Till(IsEnabled(0) && IsEnabled(1));  //  Wait until all axes are enabled

    SetGrpMotionProfile(gid, 0.5, 5, 5, 0.2);
    //  Set a motion profile for LineAbs2D

    SetupGroup(gid, 0, 1);
    //  Add axis 0 and axis 1 to the axis group and enable it

    LineAbs2D(gid, 0.1, 0.1);  //  absolute linear movement
    Till(IsGrpInPos(gid));

    LineAbs2D(gid, 0.0, 0.0);  //  absolute linear movement
    Till(IsGrpInPos(gid));
}
```

The concept is similar to that in PLCopen® Motion Control: Part 4—Coordinated Motion. Please refer to PLCopen® Chapter 4.1 “Creating and using an AxisGroup” for more information.

Note: PLCopen® is a registered trademark licensed by the association PLCopen.

8.1.3.2 Advanced group setup and velocity blending

The way to move tool center point along the two-dimension paths in Figure 8.1.3.2.1 is shown in the following HMPL task. The group motion profiles are configured with "BM_PREV", "BM_NEXT" and "BM_BUFF". The coordinated velocity at p_1 referes to the maximum velocity of Path 1 beause of "BM_PREV"; while the coordinated velocity at p_2 referes to the maximum velocity of Path 3 because of "BM_NEXT".

Note: Refer to Section 19.1 Buffer modes to know more about "BM_PREV" and "BM_NEXT".

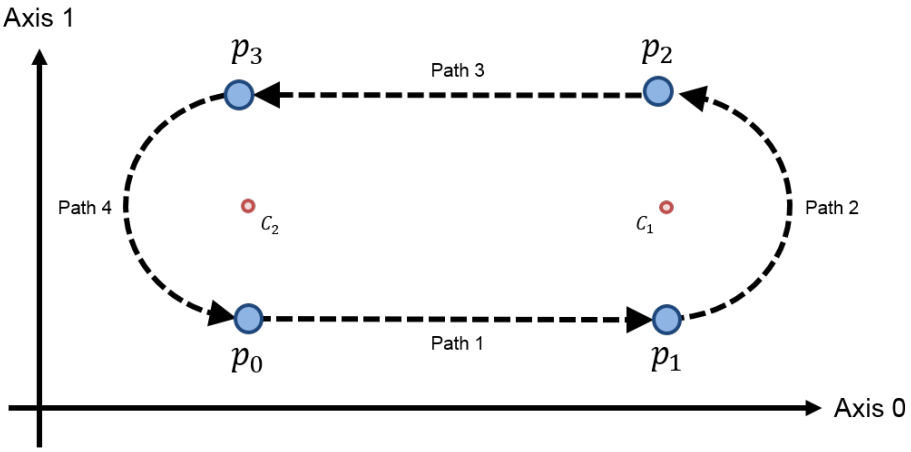


Figure 8.1.3.2.1

```
int main() {

    int axis[2] = {0, 1};  //  axis index
    int gid = 0;  //  group index

    UngrpAllAxes(gid);
    //  Remove all axes from the existing axis group and disable it (optional)

    AddAxisToGrp(gid, axis[0]);  //  Add axis to group 0
    AddAxisToGrp(gid, axis[1]);

    Enable(axis[0]);  //  Enable all axes in group 0
    Enable(axis[1]);

    Till(IsEnabled(axis[0]) && IsEnabled(axis[1]));
    //  Wait until all axes are enabled
```

```
    EnableGroup(gid);  //  Enable group 0

    double c1[3] = {0.1, 0.05, 0.0};
    double c2[3] = {0.0, 0.05, 0.0};
    double p0[6] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
    double p1[6] = {0.1, 0.0, 0.0, 0.0, 0.0, 0.0};
    double p2[6] = {0.1, 0.1, 0.0, 0.0, 0.0, 0.0};
    double p3[6] = {0.0, 0.1, 0.0, 0.0, 0.0, 0.0};

    double norm_ccw[3] = {0, 0, 1};
    double vel[4] = {0.1, 5.0, 5.0, 0.05};

    LinAbs(gid, p0, vel, CS_MCS, BM_BUFF, TM_NONE, 0.0);
    Till(IsGrpInPos(gid));

    //  Blending Next and Blending Previous
    LinAbs(gid, p1, vel, CS_MCS, BM_PREV, TM_NONE, 0.0); //  path 1
    CircAbs(gid, c1, norm_ccw, 0, p2, vel, CS_MCS, BM_NEXT, TM_NONE, 0); //  path 2
    LinAbs(gid, p3, vel, CS_MCS, BM_PREV, TM_NONE, 0.0); //  path 3
    Till(IsGrpInPos(gid));

    //  Buffered
    CircAbs(gid, c2, norm_ccw, 0, p0, vel, CS_MCS, BM_BUFF, TM_NONE, 0); //  path 4
    Till(IsGrpInPos(gid));
}
```

8.2 Group Motion Control

8.2.1 EnableGroup

Purpose

To enable an axis group.

Syntax

```
int EnableGroup(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

All of the axes in the group should be enabled before executing this function.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

8.2.2 DisableGroup

Purpose

To disable an axis group.

Syntax

```
int DisableGroup(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

8.2.3 Basic motion command

8.2.3.1 LineAbs2D

Purpose

To command an interpolated, two-dimensional linear movement on an axis group toward an absolute target position in the machine coordinate system.

Syntax

```
int LineAbs2D(
    int    group_id,
    double end_x,
    double end_y
);
```

Parameter

group_id [in]	Axis group index.
end_x [in]	The value of the absolute target position in X coordinate.
end_y [in]	The value of the absolute target position in Y coordinate.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
void main() {
    //  Assume that axis group 0 which includes two orthogonal axes is enabled
    double target_x = 0.1;
    double target_y = 0.1;
    LineAbs2D (0 /* group_id */, target_x, target_y);
    //  Move to (target_x, target_y)
    //  that is (0.1, 0.1)
}
```

Requirement

Minimum supported version	iA Studio 0.24.2326.0

8.2.3.2 LineAbs3D

Purpose

To command an interpolated, three-dimensional linear movement on an axis group toward an absolute target position in the machine coordinate system.

Syntax

```
int LineAbs3D(
    int    group_id,
    double end_x,
    double end_y,
    double end_z
);
```

Parameter

group_id [in]	Axis group index.
end_x [in]	The value of the absolute target position in X coordinate.
end_y [in]	The value of the absolute target position in Y coordinate.
end_z [in]	The value of the absolute target position in Z coordinate.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.24.2326.0

8.2.3.3 LineRel2D

Purpose

To command an interpolated, two-dimensional linear movement on an axis group toward a relative position in the machine coordinate system.

Syntax

```
int LineRel2D(
    int    group_id,
    double distance_x,
    double distance_y
);
```

Parameter

group_id [in]	Axis group index.
distance_x [in]	The value of the relative distance in X coordinate.
distance_y [in]	The value of the relative distance in Y coordinate.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
void main() {
    //  Assume that axis group 0 which includes two orthogonal axes is enabled
    //  and the starting positions of the two axes are (0.1, 0.2)
    double distance_x = 0.1;
    double distance_y = 0.1;
    LineRel2D (0 /* group_id */, distance_x, distance_y);
    //  Move to (X starting position + distance_x, Y starting position + distance_y)
    //  that is (0.2, 0.3)
}
```

Requirement

Minimum supported version	iA Studio 0.24.2326.0

8.2.3.4 LineRel3D

Purpose

To command an interpolated, three-dimensional linear movement on an axis group toward a relative position in the machine coordinate system.

Syntax

```
int LineRel3D(
    int    group_id,
    double distance_x,
    double distance_y,
    double distance_z
);
```

Parameter

group_id [in]	Axis group index.
distance_x [in]	The value of the relative distance in X coordinate.
distance_y [in]	The value of the relative distance in Y coordinate.
distance_z [in]	The value of the relative distance in Z coordinate.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.24.2326.0

8.2.3.5 Arc2D

Purpose

To command an interpolated, two-dimensional circular movement on an axis group toward an absolute target position in the machine coordinate system.

Syntax

```
int Arc2D(
    int    group_id,
    double border_x,
    double border_y,
    double end_x,
    double end_y
);
```

Parameter

group_id [in]	Axis group index.
border_x [in]	The value of the absolute border position in X coordinate.
border_y [in]	The value of the absolute border position in Y coordinate.
end_x [in]	The value of the absolute end position in X coordinate.
end_y [in]	The value of the absolute end position in Y coordinate.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
void main() {
    //  Assume that axis group 0 which includes two orthogonal axes is enabled
    //  and the starting positions of the two axes are (0, 0)
    double border_x = 0.1;
    double border_y = 0.1;
    double end_x = 0.2;
    double end_y = 0;
    Arc2D(0 /* group_id */, border_x, border_y, end_x, end_y);
    //  Circle to (end_x, end_y) through (border_x , border_y)
    //  that is, circle to (0.2, 0) through (0.1, 0.1)
}
```

Requirement

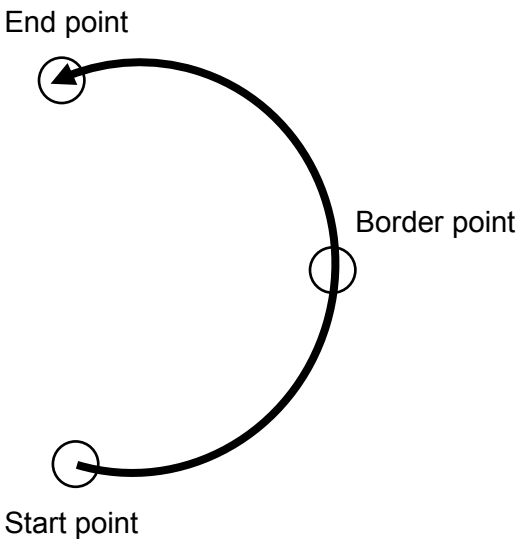
Minimum supported version	iA Studio 0.24.2326.0

Advantage

Users can specify the border point (the farthest point in the movement), and make sure that the machine can reach it.

Disadvange

It is restricted to the angle < 2π in a single command.



8.2.3.6 Circle2D

Purpose

To command an interpolated, two-dimensional circular movement on an axis group toward an absolute target position in the machine coordinate system.

Syntax

```
int Circle2D(
    int    group_id,
    double center_x,
    double center_y,
    double end_x,
    double end_y,
    int    turns
);
```

Parameter

group_id [in]	Axis group index.
center_x [in]	The value of the absolute center position in X coordinate.
center_y [in]	The value of the absolute center position in Y coordinate.
end_x [in]	The value of the absolute end position in X coordinate.
end_y [in]	The value of the absolute end position in Y coordinate.
turns [in]	Number of turns of circular path relative to the start point. It determines the direction and the total angle of circular path.

Return value

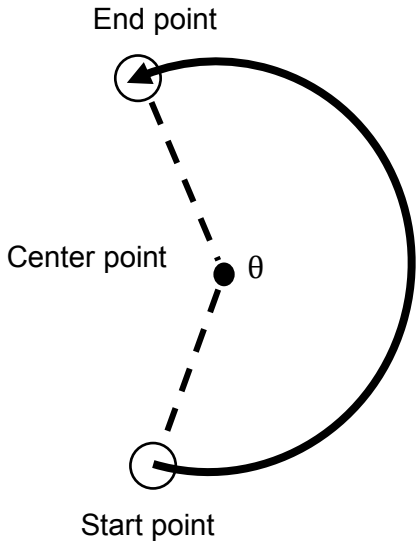
It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
void main() {
    //  Assume that axis group 0 which includes two orthogonal axes is enabled
    //  and the starting positions of the two axes are (0, 0)
    double center_x = 0.1;
    double center_y = 0;
    double end_x = 0.2;
    double end_y = 0;
    int turns = 1;
    Circle2D(0 /* group_id */, center_x, center_y, end_x, end_y, turns);
    //  Take (center_x, center_y) as the center and circle to (end_x, end_y)
    //  that is, take (0.1, 0) as the center and circle to (0.2, 0)
}
```

Requirement

Minimum supported version	iA Studio 1.1.3761.0



Advantage

No restriction to angles.

Disadvantage

Users cannot specify the border point (the farthest point in the movement). Therefore, the machine may not reach the border point.

The value of turns determines the direction of circular movement. ※ $\text{angle} = \theta + \text{turns} \times 360$

$\text{turns} \geq 0$ indicates C.C.W. direction, while $\text{turns} < 0$ indicates C.W. direction. Note that the movement of $\text{turns} = 0$ is the same as that of $\text{turns} = 1$. The following table takes $\theta = 210°$ for example.

Turns	Calculation	Angle (Degree)
-2	$210 - 2 \times 360°$	$-510°$
-1	$210 - 1 \times 360°$	$-150°$
0	$210 + 0 \times 360°$	$210°$
1	$210 + 0 \times 360°$	$210°$
2	$210 + 1 \times 360°$	$570°$

8.2.4 Advanced motion command

8.2.4.1 Bezier

Purpose

To command an interpolated Bézier curve movement on an axis group toward an absolute position in the specific coordinate system.

Syntax

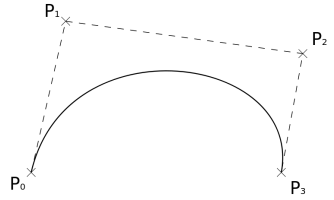
```
int Bezier(
    int group_id,
    int num_point,
    double *x,
    double *y,
    double *z,
    double *abc,
    double *motion_profile
);
```

Parameter

group_id [in] Axis group index.

num_point [in] Number of control points of Bézier curve.
Input range: 2~32

x [in] A pointer to an array which contains X coordinates of the control points.
Parameter unit: meter



y [in] A pointer to an array which contains Y coordinates of the control points.
Parameter unit: meter

z [in] A pointer to an array which contains Z coordinates of the control points.
Parameter unit: meter

abc [in]	A pointer to a three-element array which contains the end-point orientation {A, B, C} of the tool center point. Parameter unit: radian
motion_profile [in]	A pointer to a four-element array which contains the maximum tangential motion profile of TCP on the path. {max_velocity, max_acceleration, max_deceleration, smooth_time} **Refer to Section 8.3.6 SetGrpMotionProfile for details.**

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

The position of the first control point must coincide with the current axis group position.

Example

```
void main() {
    //  Assume that axis group 0 which includes three orthogonal axes is enabled
    //  and the starting positions of the three axes are {0, 0, 0}
    double x[4] = {0, 0.1, 0.3, 0.5};
    double y[4] = {0, 0.5, 0, 0.2};
    double z[4] = {0, 0.1, 0.2, 0.5};
    double abc[3] = {0, 0, 0};
    double profile[4] = {0.1, 5, 5, 0.5};
    Bezier (0 /* group_id */, 4, x, y, z, abc, profile);
}
```

The resulting path is shown in Figure 8.2.4.1.1.

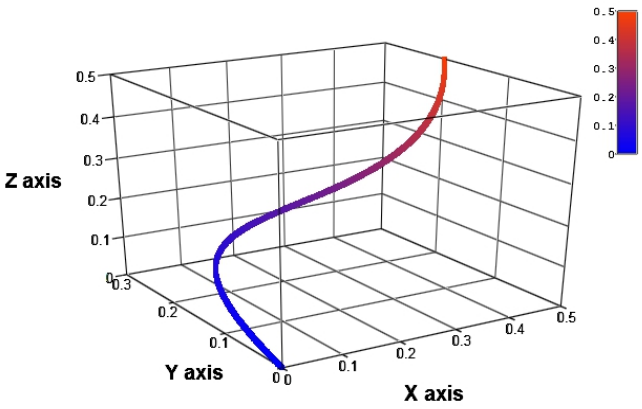


Figure 8.2.4.1.1

Requirement

Minimum supported version	iA Studio 0.23.1892.0

8.2.4.2 LinAbs

Purpose

To command an interpolated linear movement on an axis group toward an absolute position in the specific coordinate system.

Syntax

```
int LinAbs(
    int group_id,
    double *target_pos,
    double *motion_profile,
    int coord_sys,
    int buff_mode,
    int trans_mode,
    double trans_par
);
```

Parameter

group_id [in]	Axis group index.
target_pos [in]	A pointer to a six-element array which contains the absolute target position and orientation in 6-DOF {X, Y, Z, A, B, C}. Parameter unit: meter for X, Y, Z; radian for A, B, C
motion_profile [in]	A pointer to a four-element array which contains the maximum tangential motion profile of TCP on the path. {max_velocity, max_acceleration, max_deceleration, smooth_time} **Refer to Section 8.3.6 SetGrpMotionProfile for details.**
coord_sys [in]	Specify the applicable coordinate system. **Refer to Section 19.3 Coordinate systems for details.**
buff_mode [in]	Specify the path buffer mode. **Refer to Section 19.1 Buffer modes for details.**
trans_mode [in]	Specify the path transition mode. **Refer to Section 19.2 Transition modes for details.**

trans_par [in]	Specify the parameter for specific transition mode. If it is not needed, fill in zero.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.2170.0

8.2.4.3 LinRel

Purpose

To command an interpolated linear movement on an axis group toward a relative position in the specific coordinate system.

Syntax

```
int LinRel(
    int group_id,
    double *relative_dist,
    double *motion_profile,
    int coord_sys,
    int buff_mode,
    int trans_mode,
    double trans_par
);
```

Parameter

group_id [in]	Axis group index.
relative_dist [in]	A pointer to a six-element array which contains the relative distance in 6-DOF {X, Y, Z, A, B, C}. Parameter unit: meter for X, Y, Z; radian for A, B, C
motion_profile [in]	A pointer to a four-element array which contains the maximum tangential motion profile of TCP on the path. {max_velocity, max_acceleration, max_deceleration, smooth_time} **Refer to Section 8.3.6 SetGrpMotionProfile for details.**
coord_sys [in]	Specify the applicable coordinate system. **Refer to Section 19.3 Coordinate systems for details.**
buff_mode [in]	Specify the path buffer mode. **Refer to Section 19.1 Buffer modes for details.**
trans_mode [in]	Specify the path transition mode. **Refer to Section 19.2 Transition modes for details.**

trans_par [in]	Specify the parameter for specific transition mode. If it is not needed, fill in zero.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.2170.0

8.2.4.4 CircAbs

Purpose

To command an interpolated circular movement on an axis group toward an absolute position in the specific coordinate system.

Syntax

```
int CircAbs(
    int group_id,
    double *center_pos,
    double *normal_vector,
    int turns,
    double *target_pos,
    double *motion_profile,
    int coord_sys,
    int buff_mode,
    int trans_mode,
    double trans_par
);
```

Parameter

group_id [in] Axis group index.

center_pos [in] A pointer to a three-element array which contains the absolute center position {X, Y, Z} of the circular path.
Parameter unit: meter

normal_vector [in] A pointer to a three-element array which contains the normal vector {X, Y, Z} of the rotation with the right-hand rule.
Parameter unit: meter

turns [in] Number of turns of circular path relative to the start point.
It determines the direction and the total angle of circular path.

target_pos [in] A pointer to a six-element array which contains the absolute target position and orientation in 6-DOF {X, Y, Z, A, B, C}.
Parameter unit: meter for X, Y, Z; radian for A, B, C

motion_profile [in]	A pointer to a four-element array which contains the maximum tangential motion profile of TCP on the path. {max_velocity, max_acceleration, max_deceleration, smooth_time} **Refer to Section 8.3.6 SetGrpMotionProfile for details.**
coord_sys [in]	Specify the applicable coordinate system. **Refer to Section 19.3 Coordinate systems for details.**
buff_mode [in]	Specify the path buffer mode. **Refer to Section 19.1 Buffer modes for details.**
trans_mode [in]	Specify the path transition mode. **Refer to Section 19.2 Transition modes for details.**
trans_par [in]	Specify the parameter for specific transition mode. If it is not needed, fill in zero.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.2170.0

8.2.5 StopGroup

Purpose

To stop the motion of an axis group.

Syntax

```
int StopGroup(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

The motion queue of the axis group will be cleared.

Requirement

Minimum supported version	iA Studio 0.23.1960.0

8.3 Group Setting

8.3.1 AddAxisToGrp

Purpose

To add an axis to an axis group.

Syntax

```
int AddAxisToGrp(
    int group_id,
    int axis_id
);
```

Parameter

group_id [in] Axis group index.

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

(1) The maximum number of the axes is 9.

(2) The sequence of adding should correspond to {X, Y, Z, A, B, C}.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

8.3.2 RemoveAxisFromGrp

Purpose

To remove the last axis from an axis group.

Syntax

```
int RemoveAxisFromGrp(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.24.2302.0

8.3.3 SetupGroup

Purpose

To set up an axis group with a specific sequence.

Syntax

```
int SetupGroup(
    int group_id,
    int axis_id,
    int axis_id,
    …
    int axis_id
);
```

Parameter

group_id [in]	Axis group index.
axis_id [in]	Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

The maximum number of the axes is 9.

Requirement

Minimum supported version	iA Studio 0.25.2348.0

8.3.4 UngrpAllAxes

Purpose

To ungroup and disable an axis group.

Syntax

```
int UngrpAllAxes(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

8.3.5 GetGroupID

Purpose

To get the axis group ID that the axis belongs to.

Syntax

```
int GetGroupID(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The axis group ID to which the axis belongs.

It will return **-1** if the axis does not belong to any axis group.

Requirement

Minimum supported version	iA Studio 0.23.2096.0

8.3.6 SetGrpMotionProfile

Purpose

To set a max tangential motion profile of TCP on the Path for a group motion.

Syntax

```
int SetGrpMotionProfile(
    int    group_id,
    double max_velocity,
    double max_acceleration,
    double max_deceleration,
    double smooth_time
);
```

Parameter

group_id [in] Axis group index.

max_velocity [in] The new maximum profile velocity of an axis.
Parameter unit: m/s or rad/s
Input range: nonzero positive value

max_acceleration [in] The new maximum profile acceleration of an axis.
Parameter unit: m/s^2 or rad/s^2
Input range: nonzero positive value

max_deceleration [in] The new maximum profile deceleration of an axis.
Parameter unit: m/s^2 or rad/s^2
Input range: nonzero positive value

smooth_time [in] The new profile smooth time of an axis.
Parameter unit: second
Input range: 0.0~0.5

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

The default value of group motion profile is **[0.1, 0.5, 0.5, 0.05]** for velocity, acceleration, deceleration, and smooth time respectively.

Requirement

Minimum supported version	iA Studio 0.24.2326.0

8.3.7 SetGrpKin

Purpose

To set axis group kinematics transformation (between MCS and ACS).

Syntax

```
int SetGrpKin(
    int group_id,
    int kin_type
);
```

Parameter

group_id [in]	Axis group index.
kin_type [in]	Type of kinematics transformation (between MCS and ACS). **Refer to Section 19.3 Coordinate systems and Section 19.4 Kinematics for details.**

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

8.3.8 GroupReset

Purpose

To change an axis group's state from "Group Error Stop" to "Group Standby".

Syntax

```
int GroupReset(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function can only be used after all errors have been cleared.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

8.4 Group Status Functions

8.4.1 IsGrpEnabled

Purpose

To query the “enable” status of an axis group.

Syntax

```
int IsGrpEnabled(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **TRUE** (1) if the axis group is enabled. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

8.4.2 IsGrpMoving

Purpose

To query the “moving” status of an axis group. If the axis group is moving, PG (profile generator) continues outputting new positions.

Syntax

```
int IsGrpMoving(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **TRUE** (1) if the axis group is moving. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

8.4.3 IsGrpInPos

Purpose

To query the “in-position” status of an axis group. If the axis group is in-position, the position error is kept within an error window (target radius) for a specific duration (debounce time).

Syntax

```
int IsGrpInPos(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **TRUE** (1) if the axis group is in-position. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.24.2302.0

(This page is intentionally left blank.)

9. GPIO Functions

9. GPIO Functions 9-1
- 9.1 SetGPO 9-2
- 9.2 ToggleGPO 9-3
- 9.3 IsGPI_On 9-4
- 9.4 IsGPO_On 9-5
- 9.5 HIMC_GPI 9-6
- 9.6 HIMC_GPO 9-7
- 9.7 SetSlvGPO 9-8
- 9.8 ToggleSlvGPO 9-9
- 9.9 IsSlvGPI_On 9-10
- 9.10 IsSlvGPO_On 9-11
- 9.11 SLV_GPI 9-12
- 9.12 SLV_GPO 9-13

9.1 SetGPO

Purpose

To set the state of the controller's general purpose output.

Syntax

```
int SetGPO(
    int gpo_idx,
    int on_off
);
```

Parameter

gpo_idx [in]	General purpose output index.
on_off [in]	The state to be set to the specific output. "1" is for on, and "0" is for off.

Return value

It will return an **int** value **0** if the function succeeds, an **int** value **-1** if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

9.2 ToggleGPO

Purpose

To toggle the state of the controller’s general purpose output.

Syntax

```
int ToggleGPO(
    int gpo_idx
);
```

Parameter

gpo_idx [in] General purpose output index.

Return value

It will return an **int** value **0** if the function succeeds, an **int** value **-1** if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

9.3 IsGPI_On

Purpose

To query the state of the controller’s general purpose input.

Syntax

```
int IsGPI_On(
    int gpi_idx
);
```

Parameter

gpi_idx [in] General purpose input index.

Return value

It will return an **int** value **TRUE** (1) if the input is at the “on” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

9.4 IsGPO_On

Purpose

To query the state of the controller’s general purpose output.

Syntax

```
int IsGPO_On(
    int gpo_idx
);
```

Parameter

gpo_idx [in] General purpose output index.

Return value

It will return an **int** value **TRUE** (1) if the output is at the “on” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

9.5 HIMC_GPI

Purpose

To query the state of the controller's general purpose input.

Syntax

```
HIMC_GPI(
    int gpi_idx
);
```

Parameter

gpi_idx [in] General purpose input index.

Example

```
int main() {
    //  Get the state of the specific general input
    if (HIMC_GPI(4) && HIMC_GPI(6)) {
    //  if both HIMC_GPI(4) and HIMC_GPI(6) are at the “on” state
      //  Do something
    }
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

9.6 HIMC_GPO

Purpose

To query the state of the controller's general purpose output.

Syntax

```
HIMC_GPO(
    int gpo_idx
);
```

Parameter

gpo_idx [in] General purpose output index.

Example

```
int main() {
    //  Get the state of the specific general output
    if (HIMC_GPO(5)) {  //  if HIMC_GPO(5) is at the "on" state
      //  Do something
    }
    HIMC_GPO(1) = HIMC_GPI(4) && HIMC_GPI(6);  //  Set specific general output
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

9.7 SetSlvGPO

Purpose

To set the state of the slave's general purpose output.

Syntax

```
int SetSlvGPO(
    int slave_id,
    int gpo_idx,
    int on_off
);
```

Parameter

slave_id [in]	Slave ID.
gpo_idx [in]	General purpose output index.
on_off [in]	The state to be set to the specific output. "1" is for on, and "0" is for off.

Return value

It will return an **int** value **0** if the function succeeds, an **int** value **-1** if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

9.8 ToggleSlvGPO

Purpose

To toggle the state of the slave's general purpose output.

Syntax

```
int ToggleSlvGPO(
    int slave_id,
    int gpo_idx
);
```

Parameter

slave_id [in]	Slave ID.
gpo_idx [in]	General purpose output index.

Return value

It will return an **int** value **0** if the function succeeds, an **int** value **-1** if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

9.9 IsSlvGPI_On

Purpose

To query the state of the slave's general purpose input.

Syntax

```
int IsSlvGPI_On(
    int slave_id,
    int gpi_idx
);
```

Parameter

slave_id [in]	Slave ID.
gpi_idx [in]	General purpose input index.

Return value

It will return an **int** value **TRUE** (1) if the input is at the "on" state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.1.3761.0

9.10 IsSlvGPO_On

Purpose

To query the state of the slave's general purpose output.

Syntax

```
int IsSlvGPO_On(
    int slave_id,
    int gpo_idx
);
```

Parameter

slave_id [in]	Slave ID.
gpo_idx [in]	General purpose output index.

Return value

It will return an **int** value **TRUE** (1) if the output is at the "on" state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.1.3761.0

9.11 SLV_GPI

Purpose

To query the state of the slave's general purpose input.

Syntax

```
SLV_GPI(
    int slave_id,
    int gpi_idx
);
```

Parameter

slave_id [in]	Slave ID.
gpi_idx [in]	General purpose input index.

Example

```
int main() {
    //  Get the state of the specific general input
    if (SLV_GPI(0, 4) && SLV_GPI(0, 6)) {
    //  if both SLV_GPI(0, 4) and SLV_GPI(0, 6) are at the “on” state
      //  Do something
    }
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

9.12 SLV_GPO

Purpose

To query the state of the slave's general purpose output.

Syntax

```
SLV_GPO(
    int slave_id,
    int gpo_idx
);
```

Parameter

slave_id [in]	Slave ID.
gpo_idx [in]	General purpose output index.

Example

```
int main() {
    //  Get the state of the specific general output
    if (SLV_GPO(0, 1)) {  //  if SLV_GPO(0, 1) is at the "on" state
      //  Set specific general output
      SLV_GPO(0, 1) = 0;
    }
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

(This page is intentionally left blank.)

10. User Table Functions

10. User Table Functions 10-1
10.1 SetUserTable 10-2
10.2 GetUserTable 10-4
10.3 SetTableValue 10-6
10.4 GetTableValue 10-7
10.5 SaveUserTable 10-8
10.6 LoadUserTable 10-10

10.1 SetUserTable

Purpose

To write data to user table.

Syntax

```
int SetUserTable(
    int    start_idx,
    int    num_data,
    double *input
);
```

Parameter

start_idx [in]	Start index of user table.
num_data [in]	Number of elements.
input [in]	A pointer to an array which contains input data.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
int main() {
    //  Write data to user table
    double data[5] = {-2.0, 0.0, 2.0, 6.0, 4.0};
    SetUserTable(
      1,  //  start index of user table
      5,  //  number of elements
      data  //  pointer to input data array
    );
    //  the above script is the same as below
    system_user_table[1] = -2.0;
    system_user_table[2] = 0.0;
    system_user_table[3] = 2.0;
    system_user_table[4] = 6.0;
    system_user_table[5] = 4.0;
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

10.2 GetUserTable

Purpose

To retrieve the user table data.

Syntax

```
int GetUserTable(
    int    start_idx,
    int    num_data,
    double *output
);
```

Parameter

start_idx [in]	Start index of user table.
num_data [in]	Number of elements to be retrieved.
output [out]	A pointer to an array which contains output data.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
int main() {
    //  Write data to user table
    double input[5] = {-2.0, 0.0, 2.0, 6.0, 4.0};
    SetUserTable(
      1,  //  start index of user table
      5,  //  number of elements
      input  //  pointer to input data array
    );
    //  now user table has the value "-2.0", "0.0", "2.0", "6.0", "4.0"
    //  Start from index 1

    //  Read user table
    double output[3];
    GetUserTable(
      3,  //  start index of user table
      3,  //  number of elements
      output  //  pointer to output data array
    );
    //  now output[0] = 2.0;
    //  output[1] = 6.0;
    //  output[2] = 4.0;
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

10.3 SetTableValue

Purpose

To write data to the specific index of user table.

Syntax

```
int SetTableValue(
    int    index,
    double value
);
```

Parameter

index [in]	Index of user table.
value [in]	Input data.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

10.4 GetTableValue

Purpose

To get data from the specific index of user table.

Syntax

```
double GetTableValue(
    int index
);
```

Parameter

index [in] Index of user table.

Return value

Data of the specific index.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

10.5 SaveUserTable

Purpose

Store user table data in RAM to permanent memory.

Syntax

```
int SaveUserTable(
    int start_idx,
    int num_data
);
```

Parameter

start_idx [in]	Start index of user table.
num_data [in]	Number of elements to be stored.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
int main() {
    //  Write data to user table
    system_user_table[3] = 2.0;
    system_user_table[4] = 6.0;
    system_user_table[5] = 4.0;

    SaveUserTable(
      3,  //  start index of user table
      3  //  number of elements
    );
    //  Reboot the controller
    //  the value of system_user_table[3] is 2.0
    //  the value of system_user_table[4] is 6.0
    //  the value of system_user_table[5] is 4.0
}
```

Requirement

Minimum supported version	iA Studio 0.25.2359.0

10.6 LoadUserTable

Purpose

Load user table data from permanent memory to RAM.

Syntax

```
int LoadUserTable(
    int start_idx,
    int num_data
);
```

Parameter

start_idx [in]	Start index of user table.
num_data [in]	Number of elements to be loaded.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
int main() {
    //  Write data to user table
    system_user_table[3] = 2.0;
    system_user_table[4] = 6.0;
    system_user_table[5] = 4.0;
    SaveUserTable(
      3,  //  start index of user table
      3  //  number of elements
    );
    /* Fill in any value in table[3], table[4] and table[5] */
    system_user_table[3] = 999.0;
    system_user_table[4] = 777.0;
    system_user_table[5] = 888.0;
    LoadUserTable(3, 3);
    //  the value of system_user_table[3] is 2.0
    //  the value of system_user_table[4] is 6.0
    //  the value of system_user_table[5] is 4.0
}
```

Requirement

Minimum supported version	iA Studio 0.25.2359.0

(This page is intentionally left blank.)

11. Position Trigger Functions

11. Position Trigger Functions 11-1
11.1 Overview 11-2
11.1.1 PT variables 11-2
11.1.2 Flow of using PT function 11-4
11.2 EnablePT 11-5
11.3 DisablePT 11-6
11.4 IsPTEnabled 11-7
11.5 SetPT_StartPos 11-8
11.6 SetPT_EndPos 11-9
11.7 SetPT_Interval 11-10
11.8 SetPT_PulseWidth 11-11
11.9 SetPT_Polarity 11-12

11.1 Overview

With HMPL commands, users can operate PT (position trigger) related functions in some HIWIN drives. Before operating PT related functions, please consult HIWIN or local distributors for compatible drives.

11.1.1 PT variables

PT related functions are operated based on the variables listed in Table 11.1.1.1.

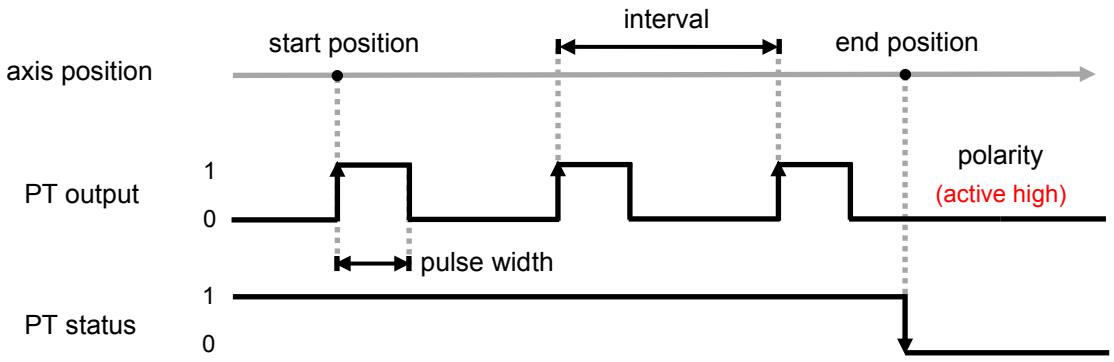


Figure 11.1.1.1

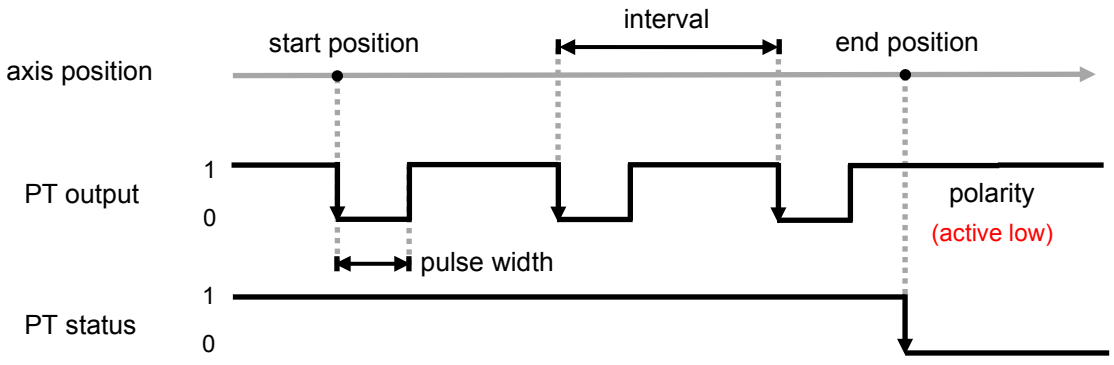


Figure 11.1.1.2

The variable “polarity” can be defined by users. In Figure 11.1.1.1, it is set to be active high. In Figure 11.1.1.2, it is set to be active low. Section 11.9 SetPT_Polarity takes Figure 11.1.1.1 for example.

Table 11.1.1.1

Name	Type	Unit	Description	HMPL functions
status	int	true / false	Status of PT function, which indicates whether PT is still functioning.	EnablePT DisablePT IsPTEnabled
start position	double	meter or radian	Start position of PT function. PT output signal train starts at this point.	SetPT_StartPos
end position	double	meter or radian	End position of PT function. No more PT output signal is sent out after this point.	SetPT_EndPos
interval	double	meter or radian	Position interval between consecutive PT outputs.	SetPT_Interval
pulse width	int	nanosecond	The width of each PT output signal. It ranges from 25 ns to 100,000 ns, with the minimum increment of 25 ns. For example, 25, 50, ... 100,000 ns.	SetPT_PulseWidth
polarity	int	0 / 1	Electronic signal polarity output. “0” is active high, and “1” is active low.	SetPT_Polarity

11.1.2 Flow of using PT function

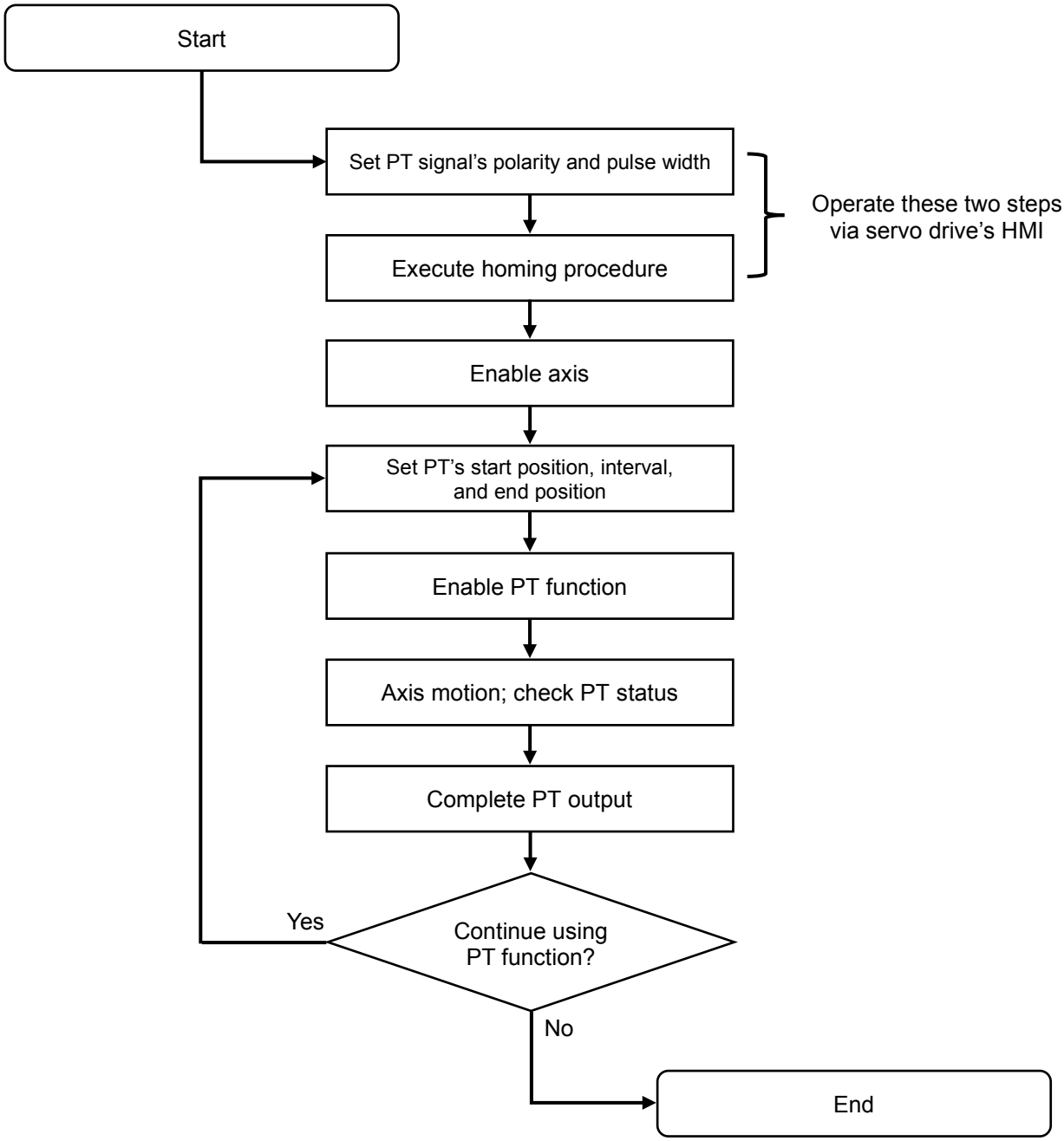


Figure 11.1.2.1

11.2 EnablePT

Purpose

To enable the position trigger function of an axis.

Syntax

```
int EnablePT(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remarks

iA Studio 1.2.4032.0 supports the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23.1993.0

11.3 DisablePT

Purpose

To disable the position trigger function of an axis.

Syntax

```
int DisablePT(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remarks

iA Studio 1.2.4032.0 supports the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23.1993.0

11.4 IsPTEnabled

Purpose

To query whether the position trigger function is enabled.

Syntax

```
int IsPTEnabled(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is at the “PT enabled” state. Otherwise, it will return **FALSE** (0).

Remarks

iA Studio 1.2.4032.0 supports the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23.1993.0

11.5 SetPT_StartPos

Purpose

To set position trigger function's start position.

Syntax

```
int SetPT_StartPos(
    int    axis_id,
    double start_pos
);
```

Parameter

axis_id [in]	Axis index.
start_pos [in]	Start position of PT function. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remarks

iA Studio 1.2.4032.0 supports the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23.1993.0

11.6 SetPT_EndPos

Purpose

To set position trigger function's end position.

Syntax

```
int SetPT_EndPos(
    int    axis_id,
    double end_pos
);
```

Parameter

axis_id [in]	Axis index.
end_pos [in]	End position of PT function. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remarks

iA Studio 1.2.4032.0 supports the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23.1993.0

11.7 SetPT_Interval

Purpose

To set position trigger function's position interval.

Syntax

```
int SetPT_Interval(
    int    axis_id,
    double interval
);
```

Parameter

axis_id [in]	Axis index.
interval [in]	Position interval between consecutive PT outputs. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remarks

iA Studio 1.2.4032.0 supports the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23.1993.0

11.8 SetPT_PulseWidth

Purpose

To set position trigger function's pulse width.

Syntax

```
int SetPT_PulseWidth(
    int axis_id,
    int width_ns
);
```

Parameter

axis_id [in]	Axis index.
width_ns [in]	The width of each PT output signal. Parameter unit: nanosecond Input range: 25~100,000 (with the minimum increment of 25)

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remarks

This function does not support E1 series servo drive. Position trigger function's pulse width must be set when setting sero drive's parameters.

Requirement

Minimum supported version	iA Studio 0.23.1993.0

11.9 SetPT_Polarity

Purpose

To set position trigger function's polarity.

Syntax

```
int SetPT_Polarity(
    int axis_id,
    int is_active_low
);
```

Parameter

axis_id [in]	Axis index.
is_active_low [in]	Electronic signal polarity output. "0" is active high, and "1" is active low.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remarks

This function does not support E1 series servo drive. Position trigger function's polarity must be set when setting sero drive's parameters.

Requirement

Minimum supported version	iA Studio 0.23.1993.0

12. Touch Probe Functions

12. Touch Probe Functions 12-1
12.1 EnableTouchProbe 12-2
12.2 DisableTouchProbe 12-3
12.3 IsTouchProbeEnabled 12-4
12.4 IsTouchProbeTriggered 12-5
12.5 GetTouchProbePos 12-6

12.1 EnableTouchProbe

Purpose

To enable the touch probe function of an axis.

Syntax

```
int EnableTouchProbe(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

12.2 DisableTouchProbe

Purpose

To disable the touch probe function of an axis.

Syntax

```
int DisableTouchProbe(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

12.3 IsTouchProbeEnabled

Purpose

To query whether the touch probe function is enabled.

Syntax

```
int IsTouchProbeEnabled(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is at the “touch probe enabled” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

12.4 IsTouchProbeTriggered

Purpose

To query whether the touch probe function is triggered.

Syntax

```
int IsTouchProbeTriggered(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis is at the “touch probe triggered” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1892.0

12.5 GetTouchProbePos

Purpose

To get the touch probe position of an axis.

Syntax

```
int GetTouchProbePos(
    int    axis_id,
    double *output
);
```

Parameter

axis_id [in]	Axis index.
output [out]	A pointer to the buffer to receive the touch probe position of an axis. Parameter unit: meter or radian

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the axis is in the gantry mode.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

13. Dynamic Error Compensation Functions

13. Dynamic Error Compensation Functions 13-1
13.1 EnableComp 13-2
13.2 DisableComp 13-3
13.3 SetupComp 13-4
13.4 SetupComp2D 13-7

13.1 EnableComp

Purpose

To enable the dynamic error compensation of an axis.

Syntax

```
int EnableComp(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the axis is enabled.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

13.2 DisableComp

Purpose

To disable the dynamic error compensation of an axis.

Syntax

```
int DisableComp(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

(1) The reference position of the axis will be reset as current feedback.

(2) This function is not applicable when the axis is enabled.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

13.3 SetupComp

Purpose

To set up one-dimensional dynamic error compensation of an axis.

Syntax

```
int SetupComp(
    int    axis_id,
    int    start_idx,
    double base_val,
    double interval,
    int    num_pt,
    int    ref_axis_id
);
```

Parameter

axis_id [in] Axis index.

start_idx [in] Start index of map point in the user table.

base_val [in] Base value (the smallest value of map input)
Parameter unit: meter or radian

interval [in] Constant interval between adjacent map points.
Parameter unit: meter or radian

num_pt [in] Number of map points.

ref_axis_id [in] Index of reference axis.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the compensated axis is disabled.

Example

In this example, the position of axis 0 (output) is compensated according to the position set-point of axis 1 (input). Their relationship is shown in Figure 13.1.1.

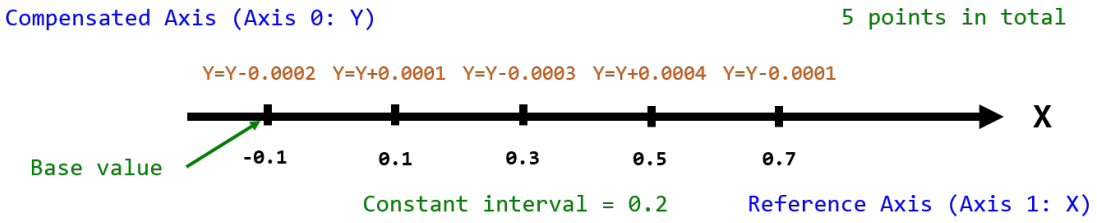


Figure 13.3.1

Set up and enable the compensation with the following HMPL. After that, enable the compensated axis (axis 0) and move the reference axis (axis 1) to observe the compensation result.

```
void main() {
    //  Set up user map table
    double data[5] = {-2e-4, 1e-4, -3e-4, 4e-4, -1e-4};
    SetUserTable(1, 5, data);

    SetupComp(
      0,    //  axis to be compensated
      1,    //  start index in user table
      -0.1, //  base value
      0.2,  //  interval
      5,    //  number of points (base data included)
      1     //  reference axis (input)
    );
    EnableComp(0);  //  Enable compensation on axis 0
    Enable(0);  //  Enable axis 0 to activate compensation
}
```

When users disable the compensation, the reference position of the axis will be reset as current feedback.

```
void main() {
    DisableComp(0);  //  Disable compensation on axis 0
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

13.4 SetupComp2D

Purpose

To set up two-dimensional dynamic error compensation of an axis.

Syntax

```
int SetupComp2D(
    int    axis_id,
    int    start_idx,
    double *base_val,
    double *interval,
    int    *num_pt,
    int    *ref_axis_id
);
```

Parameter

axis_id [in]	Axis index.
start_idx [in]	Start index of map point in the user table.
base_val [in]	A pointer to a two-element array which contains each dimension's base value (the smallest value of map input). Parameter unit: meter or radian
interval [in]	A pointer to a two-element array which contains each dimension's constant interval between adjacent map points. Parameter unit: meter or radian
num_pt [in]	A pointer to a two-element array which contains each dimension's number of map points.
ref_axis_id [in]	A pointer to a two-element array which contains each dimension's index of reference axis.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the compensated axis is disabled.

Example

In this example, the position of axis 2 (output) is compensated according to the position set-points of axis 0 (input 0) and axis 1 (input 1). Their relationship is shown in Figure 13.4.1.

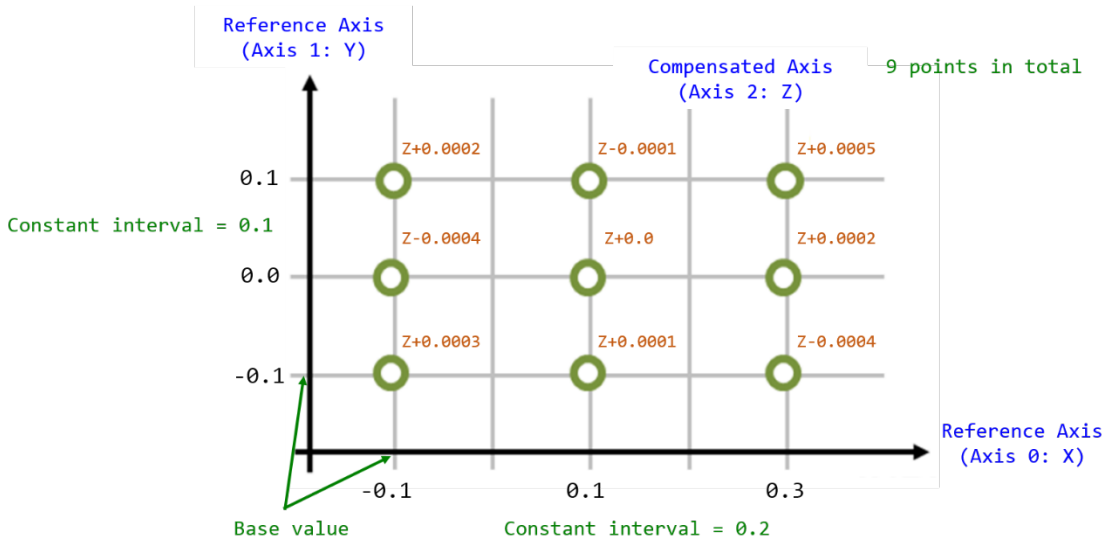


Figure 13.4.1

Set up and enable the compensation with the following HMPL. After that, enable the compensated axis (axis 2) and move the reference axes (axis 0 and axis 1) to observe the compensation result.

```
void main() {
    //  Set up user map table
    double data[9] = {
      3e-4, 1e-4,-4e-4,
      -4e-4,  0.0, 2e-4,
      2e-4,-1e-4, 5e-4
    };
    SetUserTable(3, 9, data);

    double base[2] = {-0.1, -0.1};
    double interval[2] = {0.2, 0.1};
    int num_pt[2] = {3, 3};
    int ref_axis[2] = {0, 1};

    SetupComp2D(
```

```
      2,     //  axis to be compensated
      3,     //  start index in user table
      base,  //  base values
      interval,    //  intervals
      num_pt,      //  number of points (base data included)
      ref_axis     //  reference axes (input)
    );
    EnableComp(2);  //  Enable compensation on axis 2
    Enable(2);  //  Enable axis 2 to activate compensation
}
```

When users disable the compensation, the reference position of the axis will be reset as current feedback.

```
void main() {
    DisableComp(2);  //  Disable compensation on axis 2
}
```

Requirement

Minimum supported version	iA Studio 0.23.1892.0

(This page is intentionally left blank.)

14. Filter Functions

14. Filter Functions 14-1
14.1 Overview 14-2
14.1.1 Example 14-2
14.2 EnableAxisVsf 14-3
14.3 DisableAxisVsf 14-4
14.4 SetAxisVsf 14-5
14.5 EnableAxisInShape 14-6
14.6 DisableAxisInShape 14-7
14.7 SetAxisInShape 14-8
14.8 EnableGrpInShape 14-10
14.9 DisableGrpInShape 14-11
14.10 SetGrpInShape 14-12

14.1 Overview

Filter functions are used to revise profile generator's position command. Currently, HMPL provides three kinds of filters, smooth time, vibration suppression filter (VSF), and input shaping filter (InShape).

Smooth time makes the motor accelerate smoothly to achieve smooth movement, while VSF and InShape suppresses the vibration of the motor (especially when the load of the mechanism is cantilever) during the movement. By tuning "frequency" and "damping ratio", the effect of vibration suppression can be achieved.

VSF and InShape cannot be used at the same time, but either of them can be used with smooth time.

Besides, when it comes to coordinated motion, Axis InShape function is useless. Users must adopt Group InShape function to suppress the vibration.

Note: Using filters will increase move time and decrease debounce time.

14.1.1 Example

The way to set up an input shaping filter (InShape) is shown in the following HMPL task.

```
void main()
{
    SetAxisInShape(0, 5.5, 0.03, Shaper_Normal);
    //  axis_id, frequency, damping_ratio, shaper_type
    EnableAxisInShape(0);  //  Enable InShape filter of axis 0
}
```

14.2 EnableAxisVsf

Purpose

To enable VSF filter of an axis.

Syntax

```
int EnableAxisVsf(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the motor is moving. Otherwise, the motor will generate an unexpected vibration.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

14.3 DisableAxisVsf

Purpose

To disable VSF filter of an axis.

Syntax

```
int DisableAxisVsf(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

14.4 SetAxisVsf

Purpose

To set VSF filter's parameters of an axis.

Syntax

```
int SetAxisVsf(
    int axis_id,
    double frequency,
    double damping_ratio
);
```

Parameter

axis_id [in]	Axis index.
frequency [in]	System frequency. Parameter unit: Hz Input range: 0.1~200
damping_ratio [in]	Damping ratio. Input range: 0.7~1.5 (**1.0** is recommended)

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the motor is moving. Otherwise, the motor will generate an unexpected vibration.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

14.5 EnableAxisInShape

Purpose

To enable InShape filter of an axis.

Syntax

```
int EnableAxisInShape(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the motor is moving. Otherwise, the motor will generate an unexpected vibration.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

14.6 DisableAxisInShape

Purpose

To disable InShape filter of an axis.

Syntax

```
int DisableAxisInShape(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

14.7 SetAxisInShape

Purpose

To set InShape filter's parameters of an axis.

Syntax

```
int SetAxisInShape(
    int axis_id,
    double frequency,
    double damping_ratio,
    int shaper_type
);
```

Parameter

axis_id [in]	Axis index.
frequency [in]	System frequency. Parameter unit: Hz Input range: 1.5~300
damping_ratio [in]	Damping ratio. Input range: 0.0~0.3
shaper_type [in]	Shaper type. "1" is for **Shaper_Normal**, and "0" is for **Shaper_Robust**. Shaper_Robust is more robust than Shaper_Normal, but Shaper_Normal is strong enough to suppress vibration.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

(1) This function is not applicable when the motor is moving. Otherwise, the motor will generate an unexpected vibration.

(2) The default value for frequency and damping ratio is 5.5Hz and 0.03 respectively.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

14.8 EnableGrpInShape

Purpose

To enable InShape filter of an axis group.

Syntax

```
int EnableGrpInShape(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

This function is not applicable when the motor is moving. Otherwise, the motor will generate an unexpected vibration.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

14.9 DisableGrpInShape

Purpose

To disable InShape filter of an axis group.

Syntax

```
int DisableGrpInShape(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

14.10 SetGrpInShape

Purpose

To set InShape filter's parameters of an axis group.

Syntax

```
int SetGrpInShape(
    int group_id,
    double frequency,
    double damping_ratio,
    int shaper_type
);
```

Parameter

group_id [in]	Axis group index.
frequency [in]	System frequency. Parameter unit: Hz Input range: 3.0~300
damping_ratio [in]	Damping ratio. Input range: 0.0~0.3
shaper_type [in]	Shaper type. "1" is for **Shaper_Normal**, and "0" is for **Shaper_Robust**. Shaper_Robust is more robust than Shaper_Normal, but Shaper_Normal is strong enough to suppress vibration.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Remark

(1) This function is not applicable when the motor is moving. Otherwise, the motor will generate an unexpected vibration.

(2) The default value for frequency and damping ratio is 5.5Hz and 0.03 respectively.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

(This page is intentionally left blank.)

15. HMPL Task Functions

15. HMPL Task Functions 15-1
15.1 StartTask 15-2
15.2 StartTaskFunc 15-3
15.3 StopTask 15-4
15.4 StopAllTask 15-5
15.5 IsTaskStop 15-6

15.1 StartTask

Purpose

To start the execution of a HMPL task.

Syntax

```
int StartTask(
    int task_id
);
```

Parameter

task_id [in] HMPL task ID.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.22.1862.0

15.2 StartTaskFunc

Purpose

To start the execution of a function in a HMPL task.

Syntax

```
int StartTaskFunc(
    int   task_id,
    char *func_name
);
```

Parameter

task_id [in] HMPL task ID.

func_name [in] A pointer to the buffer to store the function name in the HMPL task.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.22.1862.0

15.3 StopTask

Purpose

To stop the execution of a HMPL task.

Syntax

```
int StopTask(
    int task_id
);
```

Parameter

task_id [in] HMPL task ID.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.22.1862.0

15.4 StopAllTask

Purpose

To stop the execution of all HMPL tasks (the caller included).

Syntax

```
void StopAllTask();
```

Parameter

N/A

Return value

N/A

Requirement

Minimum supported version	iA Studio 0.23.2114.0

15.5 IsTaskStop

Purpose

To query whether the execution of a HMPL task is stopped.

Syntax

```
int IsTaskStop(
    int task_id
);
```

Parameter

task_id [in] HMPL task ID.

Return value

It will return an **int** value **TRUE** (1) if the execution of a HMPL task is stopped. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23.1905.0

16. Variables Operating Functions

16. Variables Operating Functions 16-1
16.1 GetSlvVar 16-2
16.2 GetSlvVarEx 16-3
16.3 SetSlvVar 16-4
16.4 GetSlvSt 16-5
16.5 SetSlvSt 16-6
16.6 GetConfigVar 16-7
16.7 SetConfigVar 16-8

16.1 GetSlvVar

Purpose

To get the variable value of the slave.

Syntax

```
double GetSlvVar(
    int slave_id,
    const char *var_name
);
```

Parameter

slave_id [in]	Slave index.
var_name [in]	A pointer to the buffer to store the variable name.

Return value

The value of the variable.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

16.2 GetSlvVarEx

Purpose

To get the variable value of the slave and the execution result.

Syntax

```
double GetSlvVarEx(
    int slave_id,
    const char *var_name,
    int *result
);
```

Parameter

slave_id [in]	Slave index.
var_name [in]	A pointer to the buffer to store the variable name.
result [out]	A pointer to the buffer to receive the execution result. It will return an **int** value **0** if the execution succeeds, a **nonzero** value if it fails.

Return value

The value of the variable.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

16.3 SetSlvVar

Purpose

To set the variable value of the slave.

Syntax

```
int SetSlvVar(
    int slave_id,
    const char *var_name,
    double value
);
```

Parameter

slave_id [in]	Slave index.
var_name [in]	A pointer to the buffer to store the variable name.
value [in]	The new value of the variable.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 0.23.1892.0

16.4 GetSlvSt

Purpose

To get the state of the slave.

Syntax

```
int GetSlvSt(
    int slave_id,
    const char *st_name
);
```

Parameter

slave_id [in] Slave index.

st_name [in] A pointer to the buffer to store the state name.

Return value

The state of the slave.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

16.5 SetSlvSt

Purpose

To set the state of the slave.

Syntax

```
int SetSlvSt(
    int slave_id,
    const char *st_name,
    int on_off
);
```

Parameter

slave_id [in]	Slave index.
st_name [in]	A pointer to the buffer to store the state name.
on_off [in]	The new state of the slave.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

16.6 GetConfigVar

Purpose

To get the variable value of the controller.

Syntax

```
double GetConfigVar(
    int hcv_id,
    int *result
);
```

Parameter

hcv_id [in]	hcv ID. **Note: Get the ID from `hcv_id.h` file in iA_Studio folder.**
result [out]	It will return **-1** if the function fails.

Return value

The value of the variable.

Example

```
void main()
{
    int result = 0;
    double SW_RL = GetConfigVar(0x83000065, &result);
    Print("SW_RL = %f", SW_RL);
}
```

Requirement

Minimum supported version	iA Studio 1.1.3761.0

16.7 SetConfigVar

Purpose

To set the variable value of the controller.

Syntax

```
int SetConfigVar(
    int hcv_id,
    double value
);
```

Parameter

hcv_id [in]	hcv ID. **Note: Get the ID from `hcv_id.h` file in iA_Studio folder.**
value [in]	The new value of the variable.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Example

```
void main()
{
    int result = 0;
    SetConfigVar(0x83000065, 0.0);  //  0x83000065 is axis 0 software right limit
    Print("SW_RL = %f", GetConfigVar(0x83000065, &result));
}
```

Requirement

Minimum supported version	iA Studio 1.1.3761.0

17. Error Functions

17. Error Functions 17-1
17.1 GetSystemLastErr 17-2
17.2 GetAxisLastErr 17-3
17.3 ClearAxisLastErr 17-4
17.4 GetGrpLastErr 17-5
17.5 ClearGrpLastErr 17-6

17.1 GetSystemLastErr

Purpose

To get the latest error code of the controller.

Syntax

```
int GetSystemLastErr();
```

Parameter

N/A

Return value

The latest error code of the controller.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

17.2 GetAxisLastErr

Purpose

To get the latest error code of an axis.

Syntax

```
int GetAxisLastErr(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The latest error code of the axis.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

17.3 ClearAxisLastErr

Purpose

To clear the latest error code of an axis.

Syntax

```
int ClearAxisLastErr(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

17.4 GetGrpLastErr

Purpose

To get the latest error code of an axis group.

Syntax

```
int GetGrpLastErr(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

The latest error code of the axis group.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

17.5 ClearGrpLastErr

Purpose

To clear the latest error code of an axis group.

Syntax

```
int ClearGrpLastErr(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

It will return an **int** value **0** if the function succeeds, a **nonzero** value if the function fails.

Requirement

Minimum supported version	iA Studio 1.1.3761.0

18. Homing Procedure Examples

18. Homing Procedure Examples 18-1
18.1 Basic: index signal only 18-2
18.2 Advanced: index signal & limit switch 18-4
18.3 For E1 series servo drive gantry mode 18-12

18.1 Basic: index signal only

```
//  standard procedure for axis homing --- basic version
//  single axis homing (touch probe only)
//  HIMC version 1.0

/* Set parameters */
int axis = 1;
double Home_vel = -0.02;  //  the direction depends on + or -
double ind_offset = 0.0;  //  index position relative to 0 after homing

void main()
{
    DisableTouchProbe(axis);
    Till(!IsTouchProbeEnabled(axis));
    EnableTouchProbe(axis);
    Till(IsTouchProbeEnabled(axis));

    Enable(axis);
    Till(IsEnabled(axis));
    IgnoreSWL(axis, true);

    Print("Search index...");

    MoveVel(axis,-Home_vel);

    Till(IsTouchProbeTriggered(axis) || !IsMoving(axis) || IsHWLL(axis));
    Stop(axis);
    Till(!IsMoving(axis));

    if (!IsEnabled(axis)) {goto Error;}
    else if (IsHWLL(axis)) {goto Error;}
    else if (IsTouchProbeTriggered(axis)){
            Print("Index found.");
            double pos1, pos2;
            GetTouchProbePos(axis, &pos1);
            pos2 = GetPosFb(axis) - pos1 + ind_offset;
            SetPos(axis, pos2);
```

```
            Print("Go to zero...");
            MoveAbs(axis, 0.0);  //  go to zero
            Till(!IsMoving(axis));
            if (GetRefPos(axis)==0.0 && IsEnabled(axis)) {goto HomeSuccess;}
            else {goto Error;}
            }

    Error:
    Print("Axis homing fails.");
    goto RestoreFaultResponse;

    HomeSuccess:
    Print("Axis homing succeeds.");
    goto RestoreFaultResponse;

    RestoreFaultResponse:
    IgnoreSWL(axis, false);
}
```

18.2 Advanced: index signal & limit switch

```
//  standard procedure for axis homing --- advanced version
//  single axis and gantry homing (touch probe and limit switch)
//  HIMC version 1.0

/* Set parameters */
int Homing_Type=0;  //  homing method
int axis = 1;  //  Configure the axis
int axis_1 = 2;  //  Configure the axes to a gantry pair
double Home_vel = -0.02;  //  the direction depends on + or -

//  Homing_Type=0 : Find the index directly without searching limit switch
//  Homing_Type=1 : Find the left limit switch and index
//  Homing_Type=2 : Set the middle point of switch(L,R) as the home position

//  Type 3 & 4 is for gantry mode
//  Homing_Type=3 : the motion for homing is the same as Homing_Type=0
//  Homing_Type=4 : the motion for homing is the same as Homing_Type=1

double ind_offset = 0.0;  //  index position relative to 0 after homing

int Homing_Type_0(void);
int Homing_Type_1(void);
int Homing_Type_2(double *);
int Homing_Type_3(void);
int Homing_Type_4(void);

void main()
{
    int err=0;
    if (Homing_Type==0 || Homing_Type==1)
    {
        Print("Start single axis homing...");
        if (Homing_Type==0){
            err=Homing_Type_0();
            if (err==1) {goto Error;}
            }
```

```
        if (Homing_Type==1){
            err=Homing_Type_1();
            if (err==1) {goto Error;}
        }
        if (!IsEnabled(axis)) {goto Error;}
        else if (IsHWLL(axis)) {goto Error;}
        else if (IsTouchProbeTriggered(axis)) {

            Print("Index found.");
            double pos1, pos2;
            GetTouchProbePos(axis, &pos1);
            pos2 = GetPosFb(axis) - pos1 + ind_offset;
            SetPos(axis, pos2);

            Print("Go to zero...");
            MoveAbs(axis, 0.0);  //  go to zero

            Till(!IsMoving(axis));
            if (GetRefPos(axis)==0.0 && IsEnabled(axis)) {goto HomeSuccess;}
            else {goto Error;}
        }
        else {goto Error;}
    }

    if (Homing_Type==2)
    {
        double home_pos;
        Print("Start single axis homing...");
        if (Homing_Type==2){
            err=Homing_Type_2(&home_pos);
            if (err==1) {goto Error;}
        }

        if (!IsEnabled(axis)) {goto Error;}
        else {
            Print("Hardware limit found.");
            Print("Go to home position...");
            MoveAbs(axis, home_pos);  //  go to zero
```

```
            Till(!IsMoving(axis));
            SetPos(axis, 0.0);
            if (GetRefPos(axis)==0.0 && IsEnabled(axis)) {goto HomeSuccess;}
            else {goto Error;}
        }
    }

    if (Homing_Type==3 || Homing_Type==4)
    {
        Print("Start gantry pair homing...");
        if (Homing_Type==3){
            err=Homing_Type_3();
            if (err==1) {goto Error;}
        }
        if (Homing_Type==4){
            err=Homing_Type_4();
            if (err==1) {goto Error;}
        }

        if (!IsEnabled(axis)) {goto Error;}
        else if (IsHWLL(axis)) {goto Error;}
        else if (IsTouchProbeTriggered(axis))
        {
            Print("Index found.");
            double pos1, pos2, pos3, pos4;
            GetTouchProbePos(axis, &pos1);
            GetTouchProbePos(axis_1, &pos3);

            pos2 = GetPosFb(axis) - pos1 + ind_offset;
            pos4 = GetPosFb(axis_1) - pos3 + ind_offset;

            SetPos(axis, pos2);
            SetPos(axis_1, pos4);

            //  Enable gantry pair
            Disable(axis); Disable(axis_1);
            Till(!IsEnabled(axis)&& !IsEnabled(axis_1));
            EnableGantryPair(axis, axis_1);
```

```
            Till(IsGantry(axis) && IsGantry(axis_1));

            Enable(axis);
            Till(IsEnabled(axis) && IsEnabled(axis_1));

            Print("Go to zero...");
            MoveAbs(axis, 0.0);

            Till(!IsMoving(axis));
            if (0.0 == GetRefPos(axis) && IsEnabled(axis)) {goto HomeSuccess;}
            else {goto Error;}
        }
        else {goto Error;}
    }

    Error:
    Print("Axis homing fails.");
    goto RestoreFaultResponse;

    HomeSuccess:
    Print("Axis homing succeeds.");
    goto RestoreFaultResponse;

    RestoreFaultResponse:
    IgnoreSWL(axis, false);
    IgnoreSWL(axis_1, false);
}

int Homing_Type_0()
{
    DisableTouchProbe(axis);
    Till(!IsTouchProbeEnabled(axis));
    EnableTouchProbe(axis);
    Till(IsTouchProbeEnabled(axis));

    Enable(axis);
    Till(IsEnabled(axis));
    IgnoreSWL(axis, true);
```

```
    Print("Search index...");

    MoveVel(axis, -Home_vel);

    Till(IsTouchProbeTriggered(axis) || !IsMoving(axis) || IsHWLL(axis));

    Stop(axis);
    Till(!IsMoving(axis));
    return 0;
}

int Homing_Type_1()
{
    Enable(axis);
    Till(IsEnabled(axis));
    IgnoreSWL(axis, true);
    IgnoreHWL(axis, true);

    //  Find limit switch
    if (!IsHWLL(axis)) {
        Print("Search hardware left limit...");
        MoveVel(axis, Home_vel);
    }
    Till(!IsMoving(axis) || IsHWLL(axis));
    Stop(axis);
    Till(!IsMoving(axis));
    if(IsHWLL(axis)==false) {return 1;}
    Print("Hardware left limit found.");

    DisableTouchProbe(axis);
    Till(!IsTouchProbeEnabled(axis));
    EnableTouchProbe(axis);
    Till(IsTouchProbeEnabled(axis));

    Print("Search Index...");

    MoveVel(axis, -Home_vel);
```

```
    Till(IsTouchProbeTriggered(axis) || !IsMoving(axis) || IsHWRL(axis));
    Stop(axis);
    Till(!IsMoving(axis));
    return 0;
}

int Homing_Type_2(double *home_pos)
{
    Enable(axis);
    Till(IsEnabled(axis));
    IgnoreSWL(axis, true);
    IgnoreHWL(axis, true);
    double pos1, pos2;

    //  Find left limit switch
    if (!IsHWLL(axis)) {
        Print("Search hardware left limit...");
        MoveVel(axis, Home_vel);
    }
    Till(IsHWLL(axis)) {
      pos1 = GetPosFb(axis);
    };

    Stop(axis);
    Till(!IsMoving(axis));
    if (IsHWLL(axis)==false) {return 1;}
    Print("Hardware left limit found.");

    //  Find right limit switch
    if (!IsHWRL(axis)) {
        Print("Search hardware right limit...");
        MoveVel(axis, -Home_vel);
    }
    Till(IsHWRL(axis)) {
      pos2 = GetPosFb(axis);
    };
```

```
    Stop(axis);
    Till(!IsMoving(axis));
    if (IsHWRL(axis)==false) {return 1;}
    Print("Hardware right limit found.");

    *home_pos = (pos2+pos1)/2.0;  //  pos3 is home position
    return 0;
}

int Homing_Type_3()
{
    DisableGantryPair(axis);
    Till(!IsGantry(axis) && !IsGantry(axis_1));

    DisableTouchProbe(axis); DisableTouchProbe(axis_1);
    Till(!IsTouchProbeEnabled(axis) && !IsTouchProbeEnabled(axis_1));
    EnableTouchProbe(axis); EnableTouchProbe(axis_1);
    Till(IsTouchProbeEnabled(axis) && IsTouchProbeEnabled(axis_1));

    Enable(axis); Enable(axis_1);
    Till(IsEnabled(axis) && IsEnabled(axis_1));

    IgnoreSWL(axis, true); IgnoreSWL(axis_1, true);

    MoveVel(axis, -Home_vel); MoveVel(axis_1, -Home_vel);

    Till((IsTouchProbeTriggered(axis) && IsTouchProbeTriggered(axis_1)) ||
        (!IsMoving(axis) || !IsMoving(axis_1) || IsHWLL(axis) || IsHWLL(axis_1)));

    Stop(axis); Stop(axis_1);
    Till(!IsMoving(axis) && !IsMoving(axis_1));
    return 0;
}

int Homing_Type_4()
{
    DisableGantryPair(axis);
    Till(!IsGantry(axis) && !IsGantry(axis_1));
```

```
    DisableTouchProbe(axis); DisableTouchProbe(axis_1);
    Till(!IsTouchProbeEnabled(axis) && !IsTouchProbeEnabled(axis_1));
    EnableTouchProbe(axis); EnableTouchProbe(axis_1);
    Till(IsTouchProbeEnabled(axis) && IsTouchProbeEnabled(axis_1));

    Enable(axis); Enable(axis_1);
    Till(IsEnabled(axis) && IsEnabled(axis_1));

    IgnoreHWL(axis, true); IgnoreHWL(axis_1, true);
    IgnoreSWL(axis, true); IgnoreSWL(axis_1, true);

    //  Find limit switch
    if (!IsHWLL(axis) || !IsHWLL(axis_1)) {
        Print("Search hardware left limit...");
        MoveVel(axis, Home_vel);
        MoveVel(axis_1, Home_vel);
    }

    Till(!IsMoving(axis) || !IsMoving(axis_1) || IsHWLL(axis) || IsHWLL(axis_1));

    Stop(axis); Stop(axis_1);

    Till(!IsMoving(axis) && !IsMoving(axis_1));
    if (IsHWLL(axis)==false && IsHWLL(axis_1)==false) {return 1;}
    Print("Hardware left limit found.");

    MoveVel(axis, -Home_vel); MoveVel(axis_1, -Home_vel);

    Till((IsTouchProbeTriggered(axis) && IsTouchProbeTriggered(axis_1)) ||
        (!IsMoving(axis) || !IsMoving(axis_1) || IsHWLL(axis) || IsHWLL(axis_1)));

    Stop(axis); Stop(axis_1);
    Till(!IsMoving(axis) && !IsMoving(axis_1));
    return 0;
}
```

18.3 For E1 series servo drive gantry mode

```
//  Homing program for E1 series servo drive gantry axis
//  Using homing method 34

#define INNER_POS 0
int axis_gantry = 0;
int oper_mode1_backup = 0;
int oper_mode2_backup = 0;

int GetSlvType(int axis_id, int* result)
{
    int hcv_id = ((0x8400+axis_id)<<16)+0x14;
    return (int)(GetConfigVar(hcv_id, result));
}

void main()
{
    int drive = 0;
    int result = 0;
    drive = GetSlvType(axis_gantry, &result);

    if (drive==4) {
        Disable(axis_gantry);
        Till(!IsEnabled(axis_gantry));

        //  Clear pos offset to zero
        double pos_fb = GetPosFb(axis_gantry);
        double pos_off = GetPosOffset(axis_gantry);
        SetPos(axis_gantry, pos_fb + pos_off);

        oper_mode1_backup = GetSlvVar(axis_gantry, "app.oper_mode1");
        oper_mode2_backup = GetSlvVar(axis_gantry, "app.oper_mode2");

        SetSlvVar(axis_gantry, "app.oper_mode1", INNER_POS);
        SetSlvVar(axis_gantry, "app.oper_mode2", INNER_POS);
        Till(INNER_POS==GetSlvVar(axis_gantry, "X_oper_mode_act"));
```

```
int home_method = 34;           //  Using homing method 34
int home_speed = 10;            //  Homing velocity
int home_time = 100;            //  Homing timeout
int home_acc_time = 100;        //  Homing acceleration
int home_dcc_time = 100;        //  Homing deceleration
int home_kill_dcc_time = 10;    //  Homing emergency stop deceleration

Enable(axis_gantry);
Till(IsEnabled(axis_gantry));

SetSlvVar(axis_gantry, "Pt700", home_method);
SetSlvVar(axis_gantry, "Pt703", home_time);
SetSlvVar(axis_gantry, "Pt706", home_speed);
SetSlvVar(axis_gantry, "Pt707", home_acc_time);
SetSlvVar(axis_gantry, "Pt708", home_dcc_time);
SetSlvVar(axis_gantry, "Pt709", home_kill_dcc_time);

//  Execute homing procedure
SetSlvVar(axis_gantry, "app.home_cmd", 1);

//  Wait for homing procedure to fail or succeed
Till(-1 == GetSlvVar(axis_gantry, "app.I_flag") ||
    2 == GetSlvVar(axis_gantry, "app.I_flag"));
Sleep(100);

if (-1 == GetSlvVar(axis_gantry, "app.I_flag")) {
    Print("Home Failed");
} else if (2 == GetSlvVar(axis_gantry, "app.I_flag")) {
    Print("Home Success");
}

Disable(axis_gantry);
Till(!IsEnabled(axis_gantry));
Sleep(1000);

SetSlvVar(axis_gantry, "app.oper_mode1", oper_mode1_backup);
SetSlvVar(axis_gantry, "app.oper_mode2", oper_mode2_backup);
```

```
        Sleep(1000);
        Enable(axis_gantry);
        Till(IsEnabled(axis_gantry));

        //  Move to absolute position 0
        MoveAbs(axis_gantry, 0);
        Till(IsInPos(axis_gantry));
    } else {
        Print("Please use E1 series servo drive.");
    }
}
```

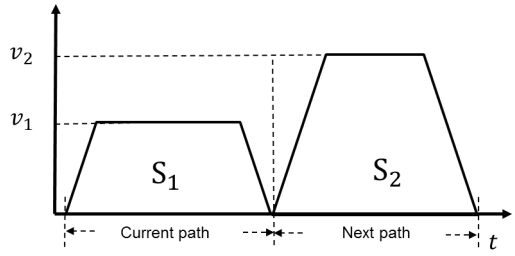
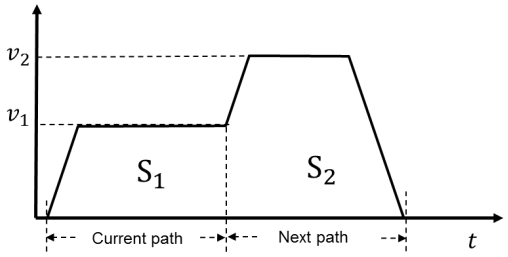
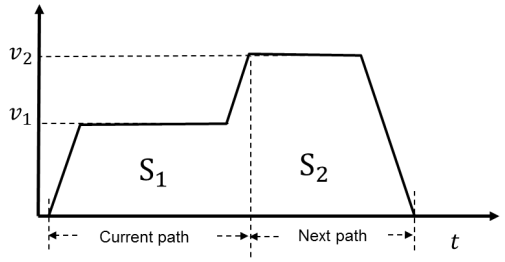
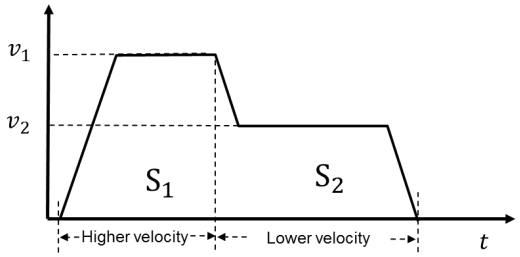
19. Appendix

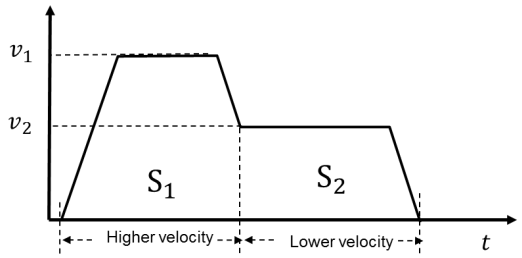
19. Appendix 19-1
19.1 Buffer modes 19-2
19.2 Transition modes 19-3
19.3 Coordinate systems 19-4
19.4 Kinematics 19-5
19.5 Math constants 19-6
19.6 System variables 19-6
19.7 Bit manipulation 19-7

19.1 Buffer modes

Buffer mode determines the velocity profile at the end-points of adjacent paths.

Table 19.1.1

HMPL definition	Description
BM_ABORT	Abort the ongoing motion and immediately start the next one.
BM_BUFF	Start next path after current path is done. 
BM_PREV	The velocity is blended with current path's velocity. (Blending) 
BM_NEXT	The velocity is blended with next path's velocity. (Blending) 
BM_HIGH	The velocity is blended with the higher velocity between the two paths. (Blending) 

BM_LOW	The velocity is blended with the lower velocity between the two paths. (Blending) 

19.2 Transition modes

Transition mode determines the type of the transition curve between adjacent paths.

Table 19.2.1

HMPL definition	Description	Corresponding parameter
TM_NONE	None: Insert no transition curve (default mode)	N/A
TM_START_VEL	Start velocity (not in use)	TPStartVelocity; Unit: m/s or rad/s
TM_CONST_VEL	Constant velocity (not in use)	TPVelocity; Unit: m/s or rad/s
TM_CORNER_DIST	Corner distance (not in use)	TPCornerDistance; Unit: meter or radian
TM_MAX_CORNER_DEV	Max. corner deviation (not in use)	TPCornerDeviation; Unit: meter or radian

19.3 Coordinate systems

Table 19.3.1

HMPL definition	Description
CS_ACS	Axis Coordinate System. It is related to the motion of individual motors.
CS_MCS	Machine Coordinate System. (sometimes called “World Coordinate System” or “Base Coordinate System”) It is a coordinate system with a fixed origin on the machine, and it is linked to ACS via a kinematic transformation (forward and backward conversion). It represents an imaginable space with up to 6 dimensions (3 translational, 3 rotational).
CS_PCS	Product Coordinate System (or “Program Coordinate System” in CNC program). It is attached to the product or the workpiece.

Figure 19.3.1 shows an example of the relationship among ACS, MCS and PCS for a SCARA robot with two rotary axes.

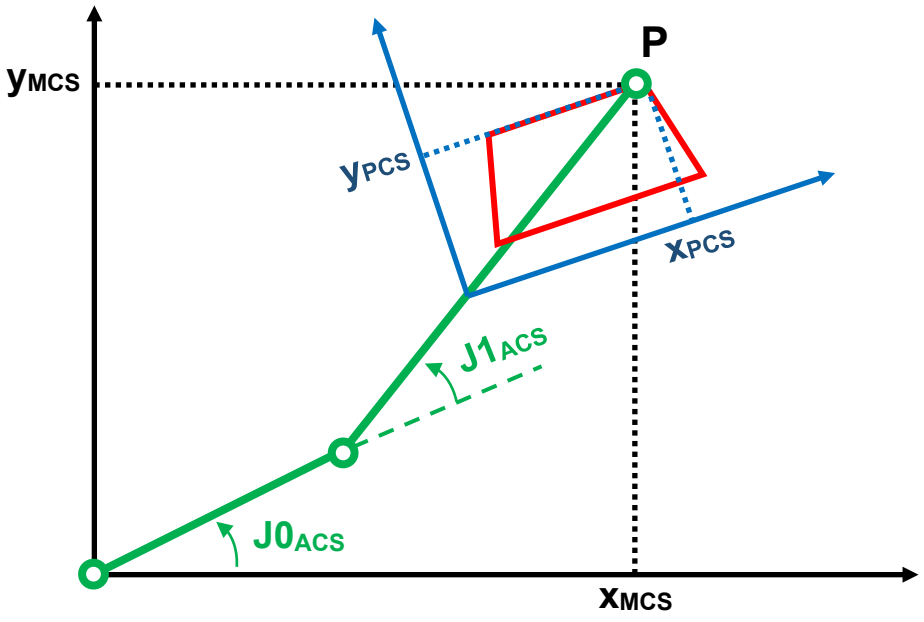


Figure 19.3.1

19.4 Kinematics

Table 19.4.1

ID	Name	Description
0	Independent	Single axis.
1	Cartesian	Map each axis in the coordinated motion group to X, Y, Z, A, B, C axis of Cartesian coordinate respectively. The maximum allowable number of axes in joint space is 6. (Default for axis group)
2	X-Y-C Gantry	(Not in use)
3	Gantry master-master	(Not in use)
4	SCARA	(Not in use)
5	DELTA	(Not in use)
6	6-axis articulated robot (PUMA)	(Not in use)
7	Custom	(Not in use)

19.5 Math constants

Table 19.5.1

Name	Description	Defined value
PI	The ratio of a circle's circumference to its diameter.	3.14159265358979323846
SQRT2	Square root of 2.	1.41421356237309504880
SQRT1_2	The reciprocal of the square root of 2, i.e., square root of 1/2.	0.70710678118654752401

19.6 System variables

Table 19.6.1

Name	Type	Description
system_timeInMs	int	A variable which stores the system time of HIMC, expressed in milliseconds.
system_fclk	int	A variable which increases by 1 every controller cycle.
system_user_table[512000]	double	An array for users. It can be stored to permanent memory. **Refer to Section 10.5 SaveUserTable.**
system_ltest0 system_ltest1 ... system_ltest9	int	Variables for users.
system_dtest0 system_dtest1 ... system_dtest9	double	Variables for users.
system_mtest[10]	double	An array for users.

19.7 Bit manipulation

There is no built-in function to set, clear, flip, or check a bit within a variable. However, users can copy the following code to the HMPL task for bit manipulation.

```
#define BIT_SET(a, idx)       ((a) |= (1<<(idx)))
#define BIT_CLEAR(a, idx)     ((a) &= ~(1<<(idx)))
#define BIT_FLIP(a, idx)      ((a) ^= (1<<(idx)))
#define BIT_CHECK(a, idx)     ((a) & (1<<(idx)))
```

Example

```
#define BIT_SET(a, idx)       ((a) |= (1<<(idx)))
#define BIT_CLEAR(a, idx)     ((a) &= ~(1<<(idx)))
#define BIT_FLIP(a, idx)      ((a) ^= (1<<(idx)))
#define BIT_CHECK(a, idx)     ((a) & (1<<(idx)))

void main() {
    int bits_value = 0;

    BIT_SET(bits_value, 0);  //  now the value of bits_value is 1
    BIT_SET(bits_value, 3);  //  now the value of bits_value is 9
    BIT_CLEAR(bits_value, 0);  //  now the value of bits_value is 8
    BIT_FLIP(bits_value, 4);  //  now the value of bits_value is 24

    bits_value = 684;
    if (BIT_CHECK(bits_value, 5)) {
      Print("bit 5 is 1");
    } else {
      Print("bit 5 is 0");
    }
    //  the output is: bit 5 is 1
}
```

(This page is intentionally left blank.)