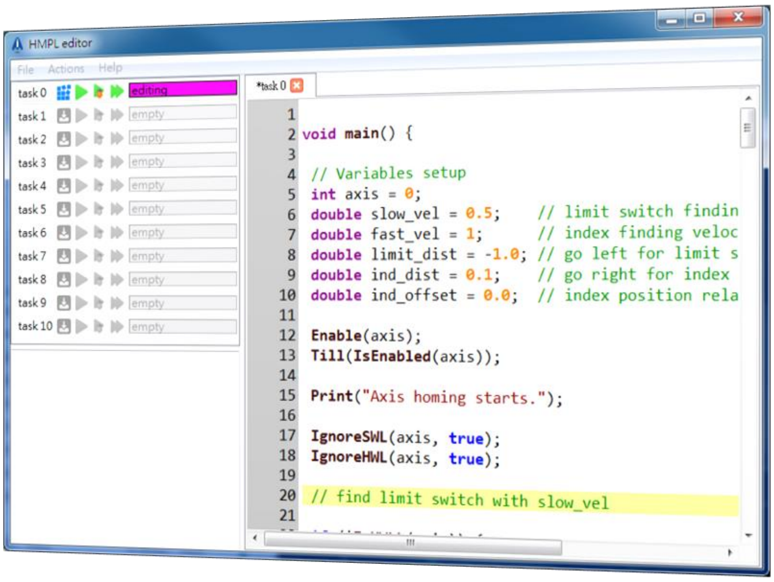
HIMC
HMPL User Guide

```
Print("Hello World");
```



Revision History

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

MH06UE01-2303_V1.0

Version

Release Date

Release Date	Version	Applicable Software Version	Revision Contents
Mar. 15^{th,} 2023	1.0	iA Studio 3.0.0	1. HIMC supports CoE communication：Add Read/Write SDO and PDO function.; remove related function of Get/Set Slave Var and Run PDL. 2. Add function ■ Chapter 2 IsSystemInit, GetECATSt, GetSlvECATSt, ScanNetwork ■ Chapter 5 SetOpMode, SetBufferMode ■ Chapter 9 SetSlvAOHex, GetSlvAOHex ■ Chapter 13 SetTouchProbeFunc ■ Chapter 18 GetDriveErr ■ Chapter 20 MoveHome, SetHomeSwitchVel, SetHomeZeroVel, SetHomeAcc, IsHomed, IsHoming 3. Add example: Section 8.1.6 4. Remove chapter 12, related function of Random PT
Jun. 30th, 2022	0.9	iA Studio 2.0	1. Add chapter/section: ■ Chapter 10 – AIO Functions ■ Section 21.3 – ■ Modbus communication 2. Add function:

Release Date	Version	Applicable Software Version	Revision Contents
			■ Chapter 2 SendMsgEvent ■ Chapter 5 MoveTrq, MovePVT, IsAcc ■ Chapter 8 ArcAngle2D, SetGrpAngMotionProfile, GetGrpCoordTrans, SetGrpCoordTrans, GetGrpPoseCmd, GetGrpPoseFb, CircleRel ■ Chapter 9 SetGPIInvert, SetGPOInvert, BindEMO, GetAllGPIInvertSt, GetAllGPOInvertSt ■ Chapter 14 SetCompAlgType ■ Chapter 20 AxisHome, SetHomeType, SetHomeMethod, SetHomeProfile, SetHomeOffset, SetHomeTimeout, SetEndStopPosErr, SetEndStopDist 3. Add example: Section 8.1.6 4. Add variable: Section 5.1.1 Section 8.1.1 5. Add error message: Section 18.1.1, Section 18.1.2, Section 18.1.3
Dec. 24th, 2021	0.8	iA Studio 1.4	1. Add function: ■ Chapter 11 SetPT_PosArray, SetPT_StateArray, SetPT_StartIndex, SetPT_EndIndex 2. Add example: Section 11.1.3 3. Remove example: Section 8.1.6
Sep. 15th, 2021	0.7	iA Studio 1.4	1. Add chapter/section: ■ Section 9.1.1 – GPIO variables

Release Date	Version	Applicable Software Version	Revision Contents
			■ Chapter 20 – Communication Functions 2. Add function: ■ Chapter 2 GetFirmwareVer ■ Chapter 5 GetVelFb, GetVelErr, GetCurrFb, SetVelScale, GetVelScale, SetRollover, GetRolloverTurns, IsDriveErr, IsPosErr ■ Chapter 8 JogGroup, JogGroupAxis, SetGrpVelScale, GetGrpVelScale ■ Chapter 13 SetupComp3D 3. Add example: SetHMIScope, Section 8.1.6, Section 9.1.2, Section 13.1.1, Section 19.3.1 4. Add variable: Section 5.1.1, Section 8.1.1, Section 16.1.2 5. Add error message: Section 17.1.1, Section 17.1.2, Section 17.1.3 6. Add homing method: Section 19.1
Sep. 16th, 2020	0.6	iA Studio 1.3	1. Section 13.1.1: Modify figures and codes. 2. Section 15.1: Add description. 3. Section 16.1.2: Modify the note of Table 16.1.2.3.
Jun. 30th, 2020	0.5	iA Studio 1.3	1. Change unit system: meter-radian-second → mm-deg-ms 2. Add chapter: ■ Chapter 18 Marco Definition and Functions ■ Chapter 19

Release Date	Version	Applicable Software Version	Revision Contents
			Homing Procedure 3. Add / Modify overview of chapter 5, 7, 8, 9, 10, 11, 12, 13, 15, 16 and 17. 4. Add function: ■ Chapter 5 Halt, Resume, SetAccTime, SetDecTime, IgnorePE ■ Chapter 6 IsInGear, IsGearMaster, IsGearSlave ■ Chapter 7 GetGantryPairID, IsGantryPair ■ Chapter 8 HaltGroup, ResumeGroup, ArcCW2D, ArcCCW2D, GetGrpKin, GetGrpMaxVel, SetGrpVel, GetGrpMaxAcc, SetGrpAcc, SetGrpAccTime, GetGrpMaxDec, SetGrpDec, SetGrpDecTime, GetGrpSMTime, SetGrpSMTime, GetGrpCoordSys, SetGrpCoordSys, GetGrpBufferMode, SetGrpBufferMode, GetGrpTransMode, SetGrpTransMode, SetGrpTransPrm, GetGrpCmdNum ■ Chapter 9 SetAllGPO, SetSlvAllGPO, GetAllGPI, GetAllGPO, GetSlvAllGPI, GetSlvAllGPO ■ Chapter 13 GetCompPos ■ Chapter 16 RunSlvPdlFunc, StopSlvPdlFunc, IsSlvPdlFuncRunning ■ Chapter 19 Home 5. Remove function:

Release Date	Version	Applicable Software Version	Revision Contents
			■ Chapter 8 Bezier ■ Chapter 12 SetPT_Polarity 6. Rename function: ■ Chapter 8 GroupReset → ResetGroup
Apr. 28th, 2020	0.4	iA Studio 1.2.4107.1	1. Chapter 18: Modify homing procedure examples of Basic, Advanced and E1 series servo drive gantry mode.
Nov. 29th, 2019	0.3	iA Studio 1.2.4032.0	1. Section 2.10: Add remark 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 ResetGroup ■ Chapte r 9

Release Date	Version	Applicable Software Version	Revision Contents
			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.

Related Documents

The figure and table of the documents related to the product are shown below. Refer to these documents as required.

Product		Doc. Name	Doc. No.	Content
Controller	HIMC Motion Controller	HIMC Installation Guide	MH07UE01-□□□□	Provides detailed information on installing and connecting HIMC motion controller.
		HIMC iA Studio User Guide	MH01UE01-□□□□	Provides detailed information on the human machine interface operation of HIMC motion controller.
		HIMC Modbus TCP User Guide	MH02UE01-□□□□	Provides detailed information on the way Modbus TCP communication protocol applied to HIMC motion controller.
		HIMC HMPL User Guide	MH06UE01-□□□□	Provides detailed information on HMPL library of HIMC motion controller.
		HIMC API Reference Guide	MH05UE01-□□□□	Provides detailed information on API library of HIMC motion controller.
		HIOM Installation Guide	MH03UE01-□□□□	Provides detailed information on installing and connecting HIOM (HIWIN mega-ulink IO module).
		ETA3 Installation Guide	MH09UE01-□□□□	Provides detailed information on installing and connecting ETA3 (HIMC remote module).
Servo Drive	E Series Servo Drive	E1 Series Servo Drive User Manual	MD09UE01-□□□□	Provides detailed information on selecting, installing, connecting, setting, performing test run for, tuning, and monitoring E1 series servo drive.
		E2 Series Servo Drive User Manual	MD28UE01-□□□□	Provides detailed information on selecting, installing, connecting, setting, performing test run for, tuning, and monitoring E2 series servo drive.
		E1 Series Servo Drive Thunder Software Operation Manual	MD12UE01-□□□□	Provides detailed information on the human machine interface operation of E1 series servo drive.
		E1 Series Servo Drive EtherCAT(CoE) Communications Command Manual	MD08UE01-□□□□	Provides detailed information on the way EtherCAT communication protocol applied to E1 series servo drive.
		E1 Series Servo Drive MECHATROLINK-III Communication Command Manual	MD24UE01-□□□□	Provides detailed information on the way MECHATROLINK-III communication protocol applied to E1 series servo drive.
		E1 Series Servo Drive PROFINET Communication Command Manual	MD02UE01-□□□□	Provides detailed information on the way PROFINET communication protocol applied to E1 series servo drive.
		E1 Series Servo Drive Gantry Control System User Manual	MD22UE01-□□□□	Provides detailed information on the usage of E1 series servo drive gantry control system.
		E1 Series Servo Drive Electronic Cam Control System User Manual	MD27UE01-□□□□	Provides detailed information on the usage of E1 series servo drive electronic cam control system.
		E1 Series Servo Drive Multi-Motion Function User Manual	MD32UE01-□□□□	Provides detailed information on the usage of E1 series servo drive multi-motion function.
		MPI Library Reference Manual	MD19UE01-□□□□	Provides detailed information on MPI library of E1 series servo drive and D series servo drive.
		MPI Examples	MD18UE01-□□□□	Provides detailed information on MPI examples of E1 series servo drive and D series servo drive.
		API Library Reference Manual for Servo Drives	MD23UE01-□□□□	Provides detailed information on API library of E1 series servo drive and D series servo drive.
		PDL Examples for E1 Series Servo Drive	MD25UE01-□□□□	Provides detailed information on PDL examples of E1 series servo drive.

Product		Doc. Name	Doc. No.	Content
Servo Drive	E Series Servo Drive	Application Note E1 PROFINET Drive Complete Setup with Siemens TIA Portal	MD30UE01-□□□□	Provides detailed information on the operation of PLC software TIA Portal when E1 PROFINET drive is used with Siemens S7 series PLC.
		Application Note E1 MECHATROLINK-III Drive Complete Setup with YASKAWA MPE720	MD31UE01-□□□□	Provides detailed information on the operation of machine controller software MPE720 when E1 MECHATROLINK-III drive is used with YASKAWA MP3000 series machine controller.
Servo Drive	D Series Servo Drive	D1 Servo Drive User Manual	MD20UE01-□□□□	Provides detailed information on selecting, installing, connecting, setting, performing test run for, tuning, and monitoring D1 servo drive.
		D2 Series Servo Drive User Manual	MD07UE01-□□□□	Provides detailed information on selecting, installing, connecting, setting, performing test run for, tuning, and monitoring D2T servo drive.
		D2T-LM Series Servo Drive User Manual	MD11UE01-□□□□	Provides detailed information on selecting, installing, connecting, setting, performing test run for, tuning, and monitoring D2T-LM servo drive.
		MPI Library Reference Manual	MD19UE01-□□□□	Provides detailed information on MPI library of E1 series servo drive and D series servo drive.
		MPI Examples	MD18UE01-□□□□	Provides detailed information on MPI examples of E1 series servo drive and D series servo drive.
		API Library Reference Manual for Servo Drives	MD23UE01-□□□□	Provides detailed information on API library of E1 series servo drive and D series servo drive.
		PDL Examples for D-series Drives User Manual	MD13UE01-□□□□	Provides detailed information on PDL examples of D series servo drive.
Motor	Linear Motor	Linear Motor User Manual	MP99UE01-□□□□	Provides detailed information on selecting, installing, and connecting linear motor.
	Direct Drive Motor	DMN Series Direct Drive Motor User Manual	MR01UE01-□□□□	Provides detailed information on selecting, installing, and connecting DMN series direct drive motor.
		DMT Series Direct Drive Motor User Manual	MR03UE01-□□□□	Provides detailed information on selecting, installing, and connecting DMT series direct drive motor.
		DMY Series Direct Drive Motor User Manual	MR04UE01-□□□□	Provides detailed information on selecting, installing, and connecting DMY series direct drive motor.
		DMS Series Direct Drive Motor User Manual	MR05UE01-□□□□	Provides detailed information on selecting, installing, and connecting DMS series direct drive motor.
		DMR Series Direct Drive Motor User Manual	MR06UE01-□□□□	Provides detailed information on selecting, installing, and connecting DMR series direct drive motor.
	Torque Motor	Torque Motor User Manual	MW99UE01-□□□□	Provides detailed information on selecting, installing, and connecting torque motor.
	AC Servo Motor	AC Servo Motor User Manual	MC03UE01-□□□□	Provides detailed information on selecting, installing, and connecting AC servo motor.
	IM-1 Series Spindle Motor	IM-1 Series Spindle Motor User Manual	MS01UE01-□□□□	Provides detailed information on selecting and installing IM-1 series spindle motor.
Linear Motor Stage	Standard Single-Axis Linear Motor	Standard Single-Axis Linear Motor Stage User Manual	MM06UE01-□□□□	Provides detailed information on selecting, installing, and connecting standard single-axis linear motor stage.

Product		Doc. Name	Doc. No.	Content
	Stage			
Actuator	Linear Actuator	Linear Actuator User Manual	MA99UE01-□□□□	Provides detailed information on selecting, installing, and connecting linear actuator.

Table of Contents

(This page is intentionally left blank.)

1. Introduction

1.1 How HMPL works

HIWIN Motion Programming Language (HMPL) constructs independent tasks with C-like syntax at users' disposal. Based on the application, users can edit the motion control logic and program in iA Studio's HMPL Editor, and the program will be complied and loaded to HIMC via HMPL complier. The real-time procedure in HIMC will execute a fixed number of basic command unit in every communication cycle.

Note :

The icon indicates that the function can be used in iA Studio's Message Window or self-installed terminal application via ASCII TCP communication (refer to chapter 21).

1.2 Version description

When HIMC controller is applied with software version iA Studio 3.0 (included) and above, it supports HIMC controller (product model: MC-XX-XX-01-XX) with CoE communication function, but it is not compatible with HIMC controller (product model: MC-XX-XX-00-XX) with MoE communication function. Users must adopt software version iA Studio 2.X (included) and below when using HIMC controller with MoE communication function.

The unit of motion variables adopted by iA Studio 1.3 (included) and above:
linear motion (mm), rotary motion (deg), time (ms)
The unit of motion variables adopted by iA Studio 1.2 (included) and below:
linear motion (m), rotary motion (rad), time (s)

1.3 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.

1.4 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 1.4.1

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

1.5 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 1.5.1

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);
}
```

Example 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

2.1 IsSystemInit

Purpose

To query whether HIMC system is at “initializing” state.

Syntax

```
int IsSystemInit();
```

Parameter

N/A

Return value

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

Requirement

Minimum supported version	iA Studio 3.0

2.2 IsSystemOper

Purpose

To query whether HIMC system is at “operation” state. If it is, the communication between HIMC and the slaves is established.

Syntax

```
int IsSystemOper();
```

Parameter

N/A

Return value

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

Requirement

Minimum supported version	iA Studio 3.0

2.3 IsSystemError

Purpose

To query whether HIMC system is at “error” state.

Syntax

```
int IsSystemError();
```

Parameter

N/A

Return value

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

Requirement

Minimum supported version	iA Studio 3.0

2.4 GetECATSt

Purpose

To get EtherCAT State Machine of the controller.

Syntax

```
int GetECATSt();
```

Parameter

N/A

Return value

EtherCAT State Machine of the controller: 1：Init, 2：Pre-Operation, 4：Safe-Operation, 8：Operation.

Requirement

Minimum supported version	iA Studio 3.0

2.5 GetSlvECATSt

Purpose

To get EtherCAT State Machine of the slave.

Syntax

```
int GetSlvECATSt(
    int slv_id
);
```

Parameter

N/A

Return value

EtherCAT State Machine of the slave: 1：Init, 2：Pre-Operation, 4：Safe-Operation, 8：Operation.

Requirement

Minimum supported version	iA Studio 3.0

2.6 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

2.7 StopAll

Purpose

To stop all axes and all axis groups with kill deceleration and make them stay at “enable” status.

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

2.8 EStop

Purpose

To stop all the execution programs in the controller (including all HMPL tasks), and disable all axes and all axis groups.

Syntax

```
void EStop();
```

Parameter

N/A

Return value

N/A

Requirement

Minimum supported version	iA Studio 0.23

2.9 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

2.10 GetFirmwareVer

Purpose

To get the firmware version of the controller.

Syntax

```
int GetFirmwareVer(
    char *ver_buf
);
```

Parameter

ver_buf [out] A pointer to the buffer to receive the returned string of the firmware version.

Return value

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

Example

```
void main() {
    char ver_buf[30] = {0};
    GetFirmwareVer(ver_buf);
    Print("%s", ver_buf);
}
```

Requirement

Minimum supported version	iA Studio 1.4

2.11 ScanNetwork

Purpose

To rescan the connection between the controller and the slave.

Syntax

```
int ScanNetwork();
```

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 3.0

2.12 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.

Example

```
void main() {
    //  Enable axis
    int axis_id = 0;
    Enable(axis_id);        Till(IsEnabled(axis_id));
    //  Start executing the scope
    SetHMIScope(1);
    //  P2P motion
    MoveAbs(axis_id, 100);  Till(IsInPos(axis_id));
    MoveAbs(axis_id, 0);    Till(IsInPos(axis_id));
    //  Stop executing the scope
    SetHMIScope(0);
}
```

Requirement

Minimum supported version	iA Studio 1.1

2.13 Sleep

Purpose

Stop executing the HMPL task for a specific period.

Syntax

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

Parameter

ms [in] Time in milliseconds.

Return value

N/A

Requirement

Minimum supported version	iA Studio 0.23

2.14 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

2.15 SendMsgEvent

Purpose

To send a string message to host PC.

Syntax

```
int SendMsgEvent(
    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

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 string message with the function “HIMC_SetHmplMsgEvtCallback” 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.
(3) The maximum length of the string message is 128 characters.

Example

```
void main()
{
    SendMsgEvent("variable: %d", 88);
}
```

Requirement

Minimum supported version	iA Studio 2.0

2.16 RunScheduler

Purpose

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

Syntax

```
void RunScheduler();
```

Parameter

N/A

Return value

N/A

Requirement

Minimum supported version	iA Studio 0.23

2.17 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

2.18 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

2.19 TON

Purpose

Rising-edge timer function. When the IN input condition is established, the return value of this function will change from 0 to a nonzero 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: ms

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

```
void main()
{
    //  Initialize counter
    Timer timer1 = TimerInit;
    Timer timer2 = TimerInit;

    int counter = 0;
    int target_cnt = 1000;
    int cnt_reach_delay_1s = 0;
    int cnt_reach_delay_3s = 0;

    for (;;) {
        Sleep(1);
        counter = counter + 1;
        //  After TON condition is satisfied, cnt_reach_delay_1s will change from
            0 to 1 in 1 second.
        cnt_reach_delay_1s = TON(&timer1, counter >= target_cnt, 1000);
        //  After TON condition is satisfied, cnt_reach_delay_3s will change from
            0 to 1 in 2 seconds.
        cnt_reach_delay_3s = TON(&timer2, cnt_reach_delay_1s, 2000);

        //  Save as system parameters to observe changes in value via Scope Manager.
        system_dtest0 = counter;
        system_dtest1 = target_cnt;
        system_dtest2 = cnt_reach_delay_1s;
        system_dtest3 = cnt_reach_delay_3s;

        //  After the condition is satisfied, counter will be reset in 0.5 second.
        if(cnt_reach_delay_3s){
            Sleep(500);
            counter = 0;
        }
    }
}
```

Requirement

Minimum supported version	iA Studio 0.23

2.20 TOF

Purpose

Falling-edge timer function. When the IN input condition is established, the return value of this function will change from a nonzero value to 0 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: ms

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

```
void main()
{
    //  Initialize counter
    Timer timer1 = TimerInit;
    Timer timer2 = TimerInit;

    int counter = 10000;
    int target_cnt = 8000;
    int cnt_reach_delay_1s = 1;
    int cnt_reach_delay_3s = 1;

    for (;;) {
        Sleep(1);
        counter = counter - 1;
        //  After TOF condition is satisfied, cnt_reach_delay_1s will change from
            1 to 0 in 1 second.
        cnt_reach_delay_1s = TOF(&timer1, counter >= target_cnt, 1000);
        //  After TOF condition is satisfied, cnt_reach_delay_3s will change from
            1 to 0 in 2 seconds.
        cnt_reach_delay_3s = TOF(&timer2, cnt_reach_delay_1s, 2000);

        //  Save as system parameters to observe changes in value via Scope Manager.
        system_dtest0 = counter;
        system_dtest1 = target_cnt;
        system_dtest2 = cnt_reach_delay_1s;
        system_dtest3 = cnt_reach_delay_3s;

        //  After the condition is satisfied, counter will be reset in 0.5 second.
        if(!cnt_reach_delay_3s){
        Sleep(500);
            counter = 10000;
        }
    }
}
```

Requirement

Minimum supported version	iA Studio 0.23

3. String functions

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

```
int 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

3.3 StringPrint

Purpose

To write a formatted string to the buffer.

Syntax

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

Parameter

destination [out]	A pointer to the buffer to receive 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

3.4 StringLen

Purpose

To get the length of a string.

Syntax

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

Parameter

str [in] The string.

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

3.5 IsStringEqual

Purpose

To check whether the two strings are the same.

Syntax

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

Parameter

str1 [in]	The string 1.
str2 [in]	The string 2.

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

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

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

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

3.9 StringCopy

Purpose

To copy a string.

Syntax

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

Parameter

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

Return value

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

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 1.3

3.10 StringCopyEx

Purpose

To copy a substring.

Syntax

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

Parameter

destination [out]	A pointer to the buffer to receive 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

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

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 1.3

3.11 StringCat

Purpose

To concatenate two strings.

Syntax

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

Parameter

destination [out]   A pointer to the buffer to receive the result of the string.

source [in]   A string to be appended.

Return value

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

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 1.3

3.12 StringCatEx

Purpose

To concatenate substrings.

Syntax

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

Parameter

destination [out]	A pointer to the buffer to receive 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

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

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 1.3

3.13 StringToDouble

Purpose

To convert a string to double type.

Syntax

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

Parameter

str [in] The string.

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

3.14 MemoryCopy

Purpose

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

Syntax

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

Parameter

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

Return value

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

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 1.3

3.15 MemorySet

Purpose

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

Syntax

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

Parameter

destination [out]	A pointer to the buffer to receive 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

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

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 = 0x01010101
}
```

Requirement

Minimum supported version	iA Studio 1.3

3.16 IsMemoryEqual

Purpose

To check whether the two memory blocks are the same.

Syntax

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

Parameter

memory_ptr1 [in]	The memory block 1.
memory_ptr2 [in]	The memory block 2.
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

(This page is intentionally left blank.)

4. Math functions

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

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

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

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

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

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

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

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

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 %f.", fabs(-35.91));
    //  The absolute value of -35.91 is 35.91.
}
```

Requirement

Minimum supported version	iA Studio 0.23

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

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

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

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

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

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

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

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

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

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

5. Axis functions

5.1 Overview

HIMC provides single axis motion commands, including point-to-point (P2P), JOG and synchronized motion. Users can use the related motion functions based on application and requirement. Figure 5.1.1 is the flow diagram of HIMC axis motion control. Reference position will be generated after motion command goes through the built-in profile generator (PG), and the position output for servo drive will be generated after the output reference position adds error compensation value.

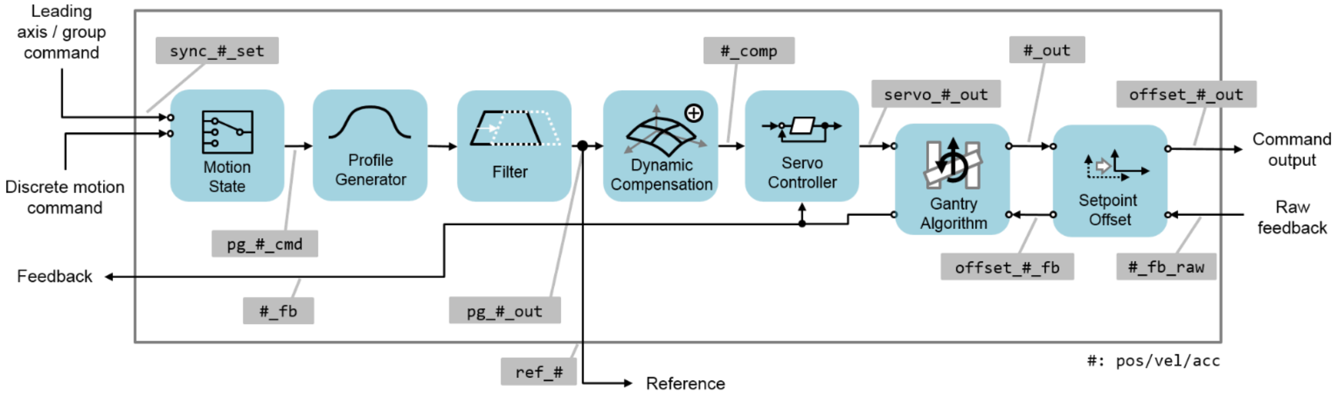


Figure 5.1.1

HIMC's profile generator has built-in S-Curve velocity planning, as Figure 5.1.2 shows. Users can set profile generator's maximum velocity, maximum acceleration, maximum deceleration and smooth time.

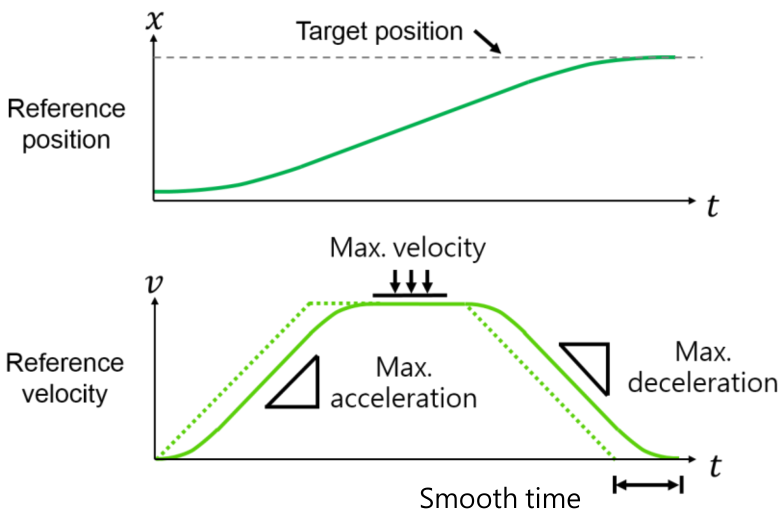


Figure 5.1.2

As Figure 5.1.3 shows, adding smooth time will delay the reference velodity, but it can effectively lower the jerk generated by high acceleration to increase the stability of system. The relationship of jerk, maximum acceleration and smooth time is shown as follows.

Jerk = Maximum acceleration / Smooth time (Ts)

As for total acceleration time, it can be obtained via the following formula.

Total acceleration time (T) = T-Curve acceleration time (Ta) + Smooth time (Ts)

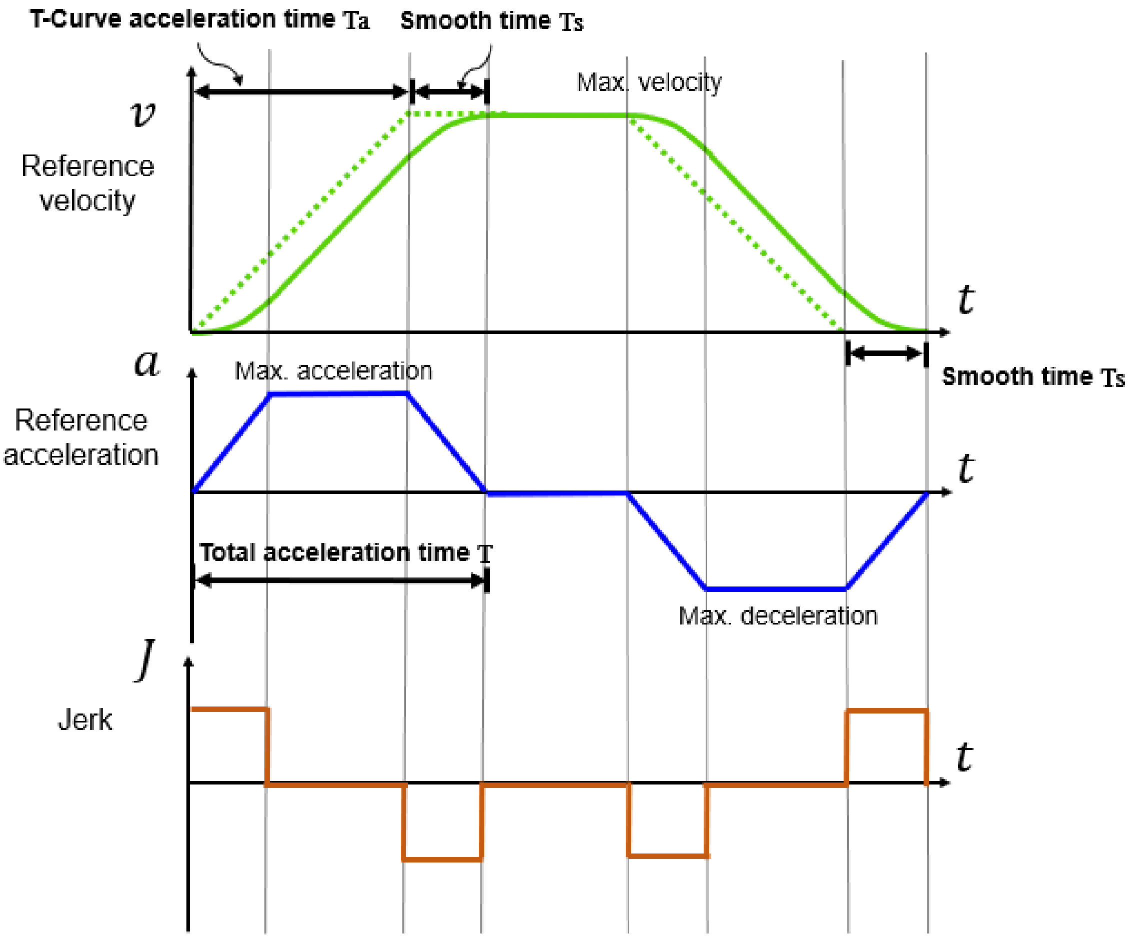


Figure 5.1.3

Besides, HIMC axis motion commands support the change of dynamic target position and velocity planning (On the Fly Modification). During axis motion, users can change axis' target position, maximum velocity and maximum acceleration / deceleration. HIMC's profile generator will move to the new target position based on the new commands and the motion parameters.

Axis motion status can be divided into “moving” and “in-positiion”, as Figure 5.1.4 shows. Continue to send axis position planning commands in section I, and end it before entering section II. Controller will judge whether the axis is in-position based on the set target radius and debounce time.

If axis position feedback remains in reference position’s target radius after debounce time, axis position will be viewed as in-position. At this time, the status of “axis is in-position” is established inside the controller. However, if axis position feedback exceeds reference position’s target radius during debounce time, the calculation of settling time will be reset. Not until the next time axis position feedback enters target radius will the in-position condition of debounce time be checked.

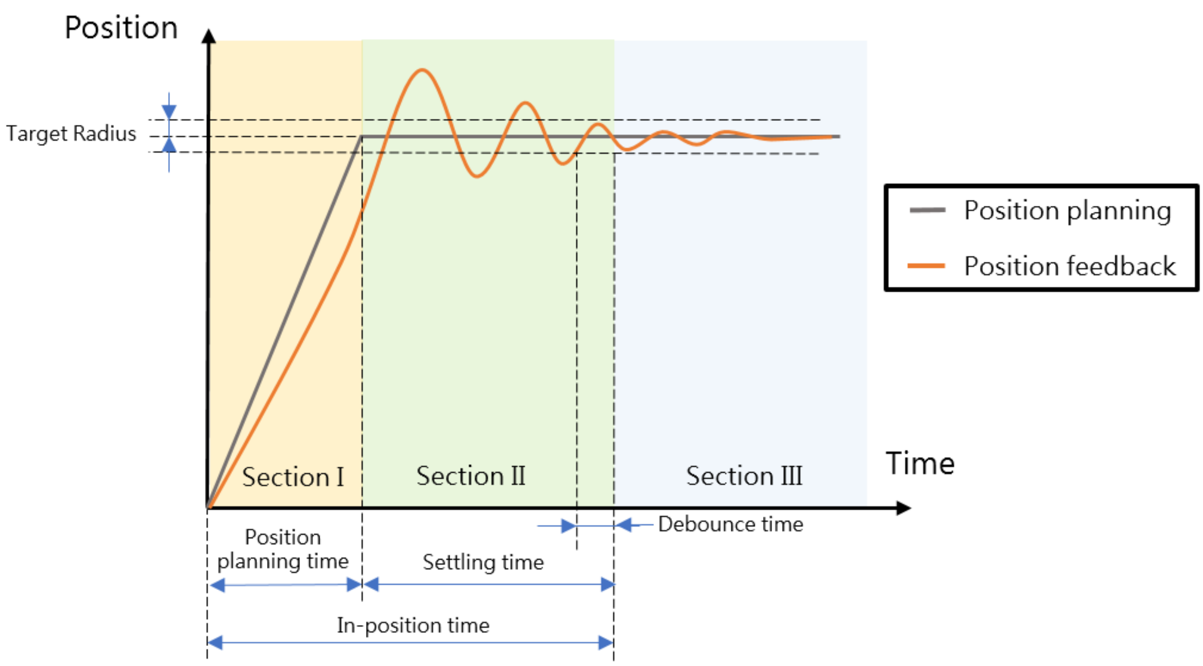


Figure 5.1.4

Refer to Figure 5.1.4, axis motion status is described as follows.

1. Section I: Axis is moving and not in-position.
2. Section II: Axis is not moving but not in-position.
3. Section III: Axis is not moving and in-position.

If the axis is at the “Synchronized” state, axis group or master axis will generate the motion profile for axis motion. Axis itself does not generate the motion profile; instead, it just follows the reference position planned by axis group or master axis.

5.1.1 Axis variables

Axis variables are divided into three categories, motion command variables, profile generator variables and status variables. Users can select the desired variables via Scope Manager in iA Studio (refer to section 4.8 in “iA Studio User Guide”). Detailed descriptions are shown in Table 5.1.1.1 to Table 5.1.1.5.

Table 5.1.1.1 Motion command variables for axis

Name	Variable	Unit	Description
Sync Position Setpoint	sync_pos_set	mm or deg	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).
Position Command	pg_pos_cmd	mm or deg	Profile generator position command. It is the target position to be followed when the axis is in discrete motion (point-to-point).
Reference Position	ref_pos	mm or deg	Reference position. It is the position set-point generated from the profile generator according to predefined motion profile.
Reference Velocity	ref_vel	mm/s or deg/s	Reference velocity. It is the velocity set-point generated from the profile generator according to predefined motion profile.
Reference Acceleration	ref_acc	mm/s^2 or deg/s^2	Reference acceleration. It is the acceleration set-point generated from the profile generator according to predefined motion profile.
Position Compensation	pos_comp_set	mm or deg	Position compensation value. It is the output of error map of dynamic error compensation function. If the function is disabled, it will be zero.
Compensated Position	pos_comp	mm or deg	Compensated position command value. It is the sum of reference position and position compensation value.
Position Offset	pos_offset	mm or deg	Position offset. The default value is zero. If users set a new axis position without moving the motor, it will be nonzero.
Position Output	pos_out	mm or deg	Position output. It is the axis position command without position offset.
Velocity Output	vel_out	mm/s or deg/s	Velocity output. It is the axis velocity command without velocity offset.
Acceleration Output	acc_out	mm/s^2 or deg/s^2	Acceleration output. It is the axis acceleration command without acceleration offset.
Offsetted Position Output	offset_pos_out	mm or deg	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.
Raw Position Feedback	pos_fb_raw	mm or deg	Raw position feedback. It is the position feedback read from the slave.
Offsetted Position Feedback	offset_pos_fb	mm or deg	Offsetted position feedback. It is the position feedback with position offset.
Position Feedback	pos_fb	mm or deg	Position feedback. It is in an axis coordinate system.
Velocity Feedback	vel_fb	mm/s or deg/s	Velocity feedback. It is in an axis coordinate system.
Position Error	pos_err	mm or deg	Position error. It is the difference between position output and raw position feedback.

Name	Variable	Unit	Description
Velocity Error	vel_err	mm/s or deg/s	Velocity error. It is the difference between velocity output and raw velocity feedback.
Move Time	movetime	ms	Move time.
Settling Time	settlingtime	ms	Settling time.

Table 5.1.1.2 Profile generator variables for axis

Name	Variable	Unit	Description
Max. Profile Velocity	max_vel	mm/s or deg/s	Maximum profile velocity. Not necessarily reached.
Max. Profile Acceleration	max_acc	mm/s^2 or deg/s^2	Maximum profile acceleration. Not necessarily reached.
Profile Deceleration	max_dec	mm/s^2 or deg/s^2	Maximum profile deceleration. Not necessarily reached.
Smooth Time	sm_factor	ms	Smooth time. Its input range is from 0 to 500. Increasing the value can reduce mechanical vibration during motion, but the total motion time will be affected.

Table 5.1.1.3 Status variables for axis

Name	Variable	Unit	Description
Fault Status	fault_status	N/A	Error status of axis; refer to Table 5.1.1.4 for bit definition.
Motion Status	motion_status	N/A	Motion status of axis; refer to Table 5.1.1.5 for bit definition.

Table 5.1.1.4 Bit definition for axis error status

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.

Table 5.1.1.5 Bit definition for axis motion status

Bit	Name	Description	Remark
0	Enabled	The axis is enabled.	N/A
1	Moving	The axis is moving.	Refer to section 5.1.
2	In Position	The axis is in-position.	Refer to section 5.1.
3	Synchronous	The axis is at the “Synchronized” state.	The axis is in an axis group or is the slave axis in synchronized motion.
4	Group	The axis is in an axis group.	Refer to section 8.1.
5	Gantry	The axis is a gantry axis.	Refer to section 7.1.
6	Input Shape	Enable Input Shape filter.	Refer to section 15.1.
7	VSF	Enable VSF filter.	Refer to section 15.1.
8	Gear	The axis is the electronic gear slave axis.	Refer to section 6.1.
9	Cam	The axis is the electronic cam slave axis.	Not supported yet.
10	Accelerating	The axis is accelerating.	Refer to section 5.1.

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.

Remark

Users must configure object 0x6040(Control word) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

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

(1) The motion queue of the axis will be cleared.

(2) Users must configure object 0x6040(Control word) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

5.2.3 Reset

Purpose

To clear the errors when the axis 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.

Remark

(1) Operate this function when the axis is at the error stop state.
(2) Users must configure object 0x6040(Control word) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

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: mm or deg

Return value

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

Remark

Users must configure the corresponding command as PDO when using this function. For example, to configure CSP mode as 0x607A (Target position).

Requirement

Minimum supported version	iA Studio 0.23

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: mm or deg

Return value

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

Remark

Users must configure the corresponding command as PDO when using this function. For example, to configure CSP mode as 0x607A (Target position).

Requirement

Minimum supported version	iA Studio 0.23

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. The direction of the motion is decided by the positive / negative value. Parameter unit: mm/s or deg/s

Return value

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

Remark

Users must configure the corresponding command as PDO when using this function. For example, to configure CSP mode as 0x607A (Target position).

Requirement

Minimum supported version	iA Studio 0.23

5.2.7 MoveTrq

Purpose

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

Syntax

```
int MoveTrq(
    int    axis_id,
    double torque_cmd
);
```

Parameter

axis_id [in]	Axis index.
torque_cmd [in]	Torque command. Parameter unit: N-m

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 only applicable to “Profile Torque” mode.

(2) If the torque command is larger than the continuous torque of motor, the motor will move with the value of continuous torque.

(3) Users must configure object 0x6071 (Target torque) as PDO and set the force constant of the motor.

Requirement

Minimum supported version	iA Studio 2.0

5.2.8 MovePVT

Purpose

To move the axis to the designated position based on the given position (P), velocity (V) and time (T).

Syntax

```
int MovePVT(
    int axis_id,
    double *time,
    double *pos,
    double *vel,
    int num_pt
);
```

Parameter

axis_id [in]	Axis index.
time [in]	A pointer to the time array given by users. Parameter unit: ms
pos [in]	A pointer to the position array given by users. Parameter unit: mm or deg
vel [in]	A pointer to the velocity array given by users. Parameter unit: mm/s or deg/s
num_pt [in]	The number of PVT motion points. Its maximum value is 50. The length of time, position and velocity array must be the same as this parameter.

Return value

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

Remark

Users must configure the corresponding command as PDO when using this function. For example, to configure CSP mode as 0x607A (Target position).

Example

```
void main() {
    int axis_id = 0;
    //  Position, velocity and time array of PVT motoin
    double point[6] = {0.0, 153.333, 42.123, 161.21, 177.0, 83.333};
    double velocity[6] = {0.0, 1000.0, 800.0, 1660.0, 450.0, 0.0};
    double time[6] = {0.0, 2000.0, 3000.0, 4000.0, 6000.0, 11000.0};
    Enable(axis_id);
    Till(IsEnabled(axis_id));
    //  Start PVT motion
    MovePVT(axis_id, time, point, velocity, 6);
}
```

Requirement

Minimum supported version	iA Studio 2.0

5.2.9 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

5.2.10 Halt

Purpose

To halt the motion of an axis; its velocity will be set as 0.

Syntax

```
int Halt(
    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

In PP/PV/PT mode, users must configure object 0x6040(Control word) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 1.3

5.2.11 Resume

Purpose

To resume the motion of an axis from “halt” status.

Syntax

```
int Resume(
    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

In PP/PV/PT mode, users must configure object 0x6040(Control word) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 1.3

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: mm/s or deg/s

Requirement

Minimum supported version	iA Studio 0.23

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: mm/s or deg/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

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: mm/s^2 or deg/s^2

Requirement

Minimum supported version	iA Studio 0.23

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: mm/s^2 or deg/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

5.3.5 SetAccTime

Purpose

To set the acceleration time of an axis.

Syntax

```
int SetAccTime(
    int    axis_id,
    double acc_time
);
```

Parameter

axis_id [in]	Axis index.
acc_time [in]	The acceleration time of an axis. Parameter unit: ms 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 1.3

5.3.6 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: mm/s^2 or deg/s^2

Requirement

Minimum supported version	iA Studio 0.23

5.3.7 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: mm/s^2 or deg/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

5.3.8 SetDecTime

Purpose

To set the deceleration time of an axis.

Syntax

```
int SetDecTime(
    int    axis_id,
    double dec_time
);
```

Parameter

axis_id [in]	Axis index.
dec_time [in]	The deceleration time of an axis. Parameter unit: ms 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 1.3

5.3.9 GetKillDec

Purpose

To get the kill deceleration of an axis.

Syntax

```
double GetKillDec(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The kill deceleration of the axis.

Unit: mm/s^2 or deg/s^2

Requirement

Minimum supported version	iA Studio 1.3

5.3.10 SetKillDec

Purpose

To set the kill deceleration of an axis.

Syntax

```
int SetKillDec(
    int    axis_id,
    double kill_dec
);
```

Parameter

axis_id [in]	Axis index.
kill_dec [in]	The new kill deceleration of an axis. Parameter unit: mm/s^2 or deg/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 1.3

5.3.11 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: mm or deg

Requirement

Minimum supported version	iA Studio 1.1

5.3.12 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: mm or deg

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

5.3.13 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: mm or deg

Requirement

Minimum supported version	iA Studio 1.1

5.3.14 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: mm or deg

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

5.3.15 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: ms

Requirement

Minimum supported version	iA Studio 0.23

5.3.16 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: ms Input range: 0 ~ 500

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 moving.

Requirement

Minimum supported version	iA Studio 0.23

5.3.17 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: ms

Requirement

Minimum supported version	iA Studio 0.24

5.3.18 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: ms

Requirement

Minimum supported version	iA Studio 0.24

5.3.19 SetPos

Purpose

To set the position of an axis and change home offset.

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: mm or deg

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 at the "Synchronized" state, added to an axis group, or at the "error" state.

Requirement

Minimum supported version	iA Studio 0.23

5.3.20 GetPosFb

Purpose

To get the position feedback of an axis.

Syntax

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

Parameter

axis_id [in] Axis index.

Return value

The position feedback of the axis.

Unit: mm or deg

Remark

Users must configure object 0x6064(Position actual value) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

5.3.21 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: mm or deg

Requirement

Minimum supported version	iA Studio 0.23

5.3.22 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: mm or deg

Requirement

Minimum supported version	iA Studio 0.23

5.3.23 GetVelFb

Purpose

To get the velocity feedback of an axis.

Syntax

```
double GetVelFb(
    int axis_id
);
```

Parameter

axis_id [in]   Axis index.

Return value

The velocity feedback of the axis.

Unit: mm/s or deg/s

Requirement

Minimum supported version	iA Studio 1.4

5.3.24 GetVelErr

Purpose

To get the velocity error of an axis.

Syntax

```
double GetVelErr(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The velocity error of the axis.

Unit: mm/s or deg/s

Requirement

Minimum supported version	iA Studio 1.4

5.3.25 GetCurrFb

Purpose

To get the current feedback of an axis.

Syntax

```
int GetCurrFb(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The current feedback of the axis.

Parameter unit: 0.1% rated current.

Remark

Users must configure object 0x6077(Torque actual value) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 3.0

5.3.26 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: mm or deg

Requirement

Minimum supported version	iA Studio 0.23

5.3.27 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: mm/s or deg/s

Requirement

Minimum supported version	iA Studio 0.23

5.3.28 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: mm/s^2 or deg/s^2

Requirement

Minimum supported version	iA Studio 0.23

5.3.29 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: mm or deg

Requirement

Minimum supported version	iA Studio 0.23

5.3.30 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: mm/s or deg/s

Requirement

Minimum supported version	iA Studio 0.23

5.3.31 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: mm/s^2 or deg/s^2

Requirement

Minimum supported version	iA Studio 0.23

5.3.32 IgnoreHWL

Purpose

To 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

5.3.33 IgnoreSWL

Purpose

To 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

5.3.34 IgnorePE

Purpose

To ignore the warning messages of position error limit.

Syntax

```
int IgnorePE(
    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 1.3

5.3.35 GetAxisNum

Purpose

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

Syntax

```
int GetAxisNum();
```

Parameter

N/A

Return value

The number of the axes connected to the controller.

Requirement

Minimum supported version	iA Studio 0.23

5.3.36 SetVelScale

Purpose

To set the velocity scale of axis motion.

Syntax

```
int SetVelScale(
    int    axis_id,
    double vel_scale
);
```

Parameter

axis_id [in]	Axis index.
vel_scale [in]	The new velocity scale of axis motion. Input range: 0 ~ 100

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 1.4

5.3.37 GetVelScale

Purpose

To get the velocity scale of axis motion.

Syntax

```
double GetVelScale(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

The velocity scale of axis motion. Its range is from 0 to 100.

Requirement

Minimum supported version	iA Studio 1.4

5.3.38 SetRollover

Purpose

To set the position rollover value of an axis.

Syntax

```
int SetRollover(
    int    axis_id,
    double rollover_val
);
```

Parameter

axis_id [in]	Axis index.
rollover_val [in]	The position rollover value of an axis. Parameter unit: mm or deg 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.

Remark

(1) If parameter “rollover_val” is set as 0, the function is closed.
(2) This function is applicable only when the axis is disabled.
(3) This function is not applicable when the axis is added to an axis group.

Requirement

Minimum supported version	iA Studio 1.4

5.3.39 GetRolloverTurns

Purpose

To get the number of turns when an axis is on rollover mode.

Syntax

```
int GetRolloverTurns(
    int axis_id
);
```

Parameter

axis_id [in]   Axis index.

Return value

The number of turns when the axis is on rollover mode.

Requirement

Minimum supported version	iA Studio 1.4

5.3.40 SetOpMode

Purpose

To set the operation mode of an axis.

Syntax

```
int SetOpMode(
    int    axis_id,
    int    op_mode
);
```

Parameter

axis_id [in] Axis index.

op_mode [in] New operation mode of an axis.
Input range: 1(PP), 3(PV), 4(PT), 8(CSP), 9(CSV), 10(CST)

Return value

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

Remark

Users must configure object 0x6060(Mode of operation) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 3.0

5.3.41 SetBufferMode

Purpose

To set buffer mode of an axis.

Syntax

```
int SetBufferMode(
    int    axis_id,
    int    buf_mode
);
```

Parameter

axis_id [in]	Axis index.
buf_mode [in]	New buffer mode of an axis. Input range: 0 (immediate stop mode), 1(buffer mode), 2(continuous motion mode)

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 3.0

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 at the “Enabled” state. Otherwise, it will return **FALSE** (0).

Remark

Users must configure object 0x6041(Stratus word) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

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 at the "Moving" state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23

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 at the "InPos" state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23

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 “ErrorStop” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23

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

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

5.4.7 IsSync

Purpose

To query whether the axis is at the “Synchronized” state. If the axis is at the “Synchronized” 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

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).

Remark

When using this function, users must configure object 0x60FD(Digital inputs) as PDO and specify bit 0 as left limit input.

Requirement

Minimum supported version	iA Studio 0.23

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).

Remark

When using this function, users must configure object 0x60FD(Digital inputs) as PDO and specify bit 1 as right limit input.

Requirement

Minimum supported version	iA Studio 0.23

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

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

5.4.12 IsDriveErr

Purpose

To query whether the axis triggers drive alarms.

Syntax

```
int IsDriveErr(
    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 “DriveErr” state. Otherwise, it will return **FALSE** (0).

Remark

Users must configure object 0x6041(Status word) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 1.4

5.4.13 IsPosErr

Purpose

To query whether the position error of an axis exceeds the protection limit.

Syntax

```
int IsPosErr(
    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 “PosErr” state. Otherwise, it will return **FALSE** (0).

Remark

The error protection limit indicates the position error tolerance of an axis in controller.

Requirement

Minimum supported version	iA Studio 1.4

5.4.14 IsCompActive

Purpose

To query whether the compensation function is activated.

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 “CompActive” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23

5.4.15 IsAcc

Purpose

To query whether the axis is accelerating.

Syntax

```
int IsAcc(
    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 “Acc” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 2.0

6. Synchronized Motion functions

6.1 Overview

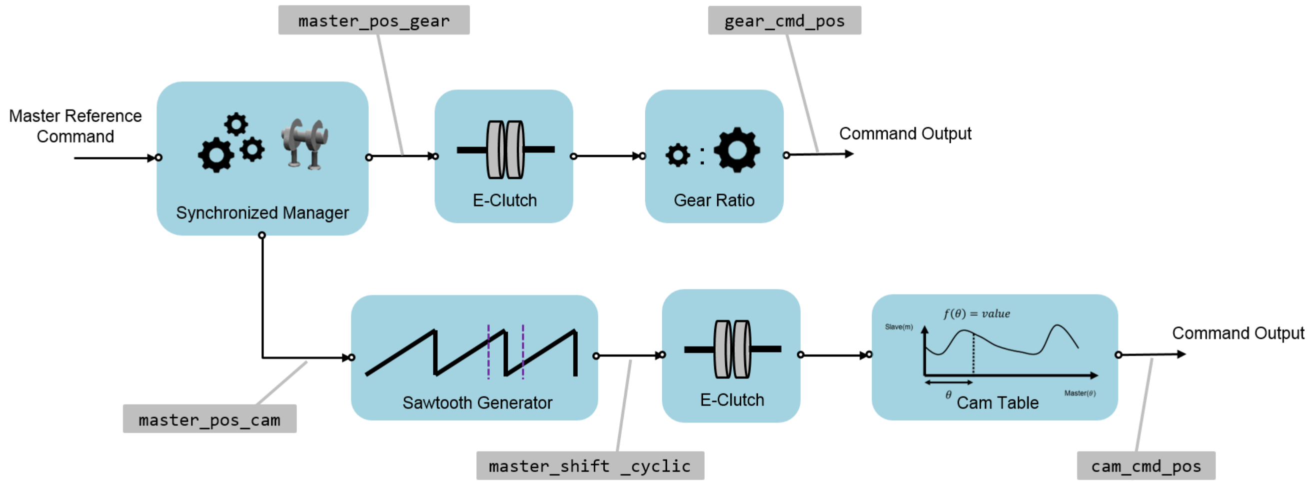


Figure 6.1.1

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.

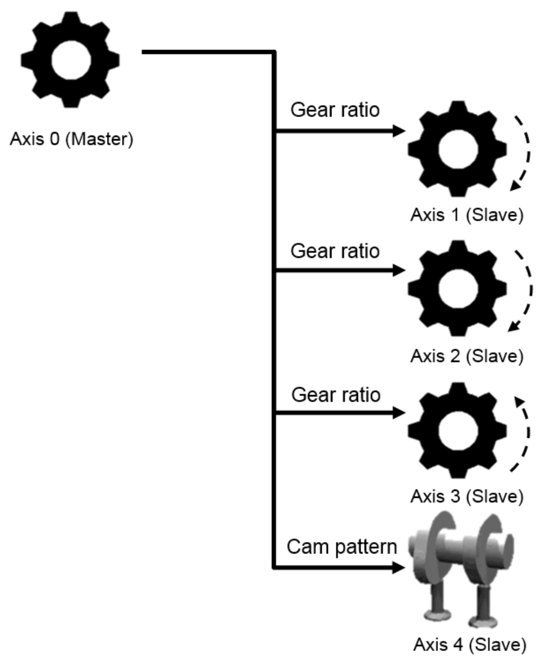


Figure 6.1.2

6.1.1 Synchronized motion variables

Common synchronized motion variables are given in Table 6.1.1.1. Users can select the desired variables via Scope Manager in iA Studio (refer to section 4.8 in "iA Studio User Guide").

Table 6.1.1.1

Name	Variable	Unit	Description
Raw Master Position	master_pos_gear	mm or deg	Position command of master axis.
Gear Command Position	gear_cmd_pos	mm or deg	Slave axis outputs position command.
Gear Ratio	gear_ratio	mm or deg	Gear ratio.

6.1.2 Example

```
void main()
{

    double target = 100;
    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

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

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

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

6.6 GetGearRatio

Purpose

To get the gear ratio of slave axis.

Syntax

```
double GetGearRatio(
    int axis_slave_id
);
```

Parameter

axis_slave_id [in]   Slave axis index.

Return value

The gear ratio of slave axis.

Requirement

Minimum supported version	iA Studio 1.3

6.7 IsInGear

Purpose

To query whether the slave axis is at the “engaged” state.

Syntax

```
int IsInGear(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

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

Requirement

Minimum supported version	iA Studio 1.3

6.8 IsGearMaster

Purpose

To query whether the axis is master axis.

Syntax

```
int IsGearMaster(
    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 "GearMaster" state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.3

6.9 IsGearSlave

Purpose

To query whether the axis is slave axis.

Syntax

```
int IsGearSlave(
    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 “GearSlave” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.3

7. Gantry functions

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, as Figure 7.1.1 shows. After establishing gantry configuration, users can give RHS axis a linear-axis-direction command to drive both RHS and LHS axes in the same direction, and give LHS axis a rotary motion command of yaw-axis-direction.

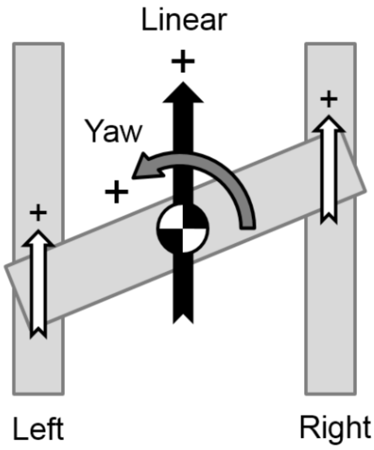


Figure 7.1.1

In gantry configuration, linear axis' and yaw axis' position feedback are defined as follows.

$$Pos_{linear} = \frac{Pos_{RHS} + Pos_{LHS}}{2}; \qquad Pos_{yaw} = \frac{Pos_{RHS} - Pos_{LHS}}{2}$$

Pos_{linear}	: Linear axis' position feedback	Pos_{yaw}	: Yaw axis' position feedback
Pos_{RHS}	: RHS axis' position feedback	Pos_{LHS}	: LHS axis' position feedback

Figure 7.1.2 is a position feedback schematic of linear axis, yaw axis, RHS axis and LHS axis.

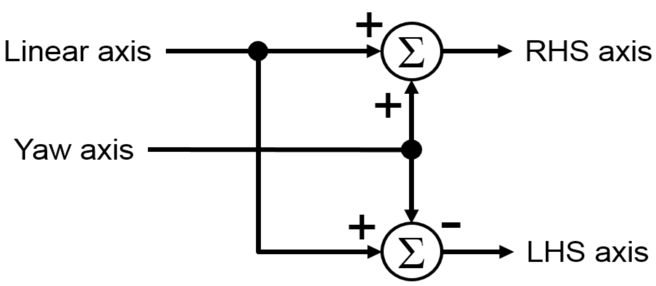


Figure 7.1.2

7.1.1 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 disabled.

Requirement

Minimum supported version	iA Studio 0.23

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

7.4 GetGantryPairID

Purpose

To get the gantry pair ID of any gantry axis.

Syntax

```
int GetGantryPairID(
    int axis_id
);
```

Parameter

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

Return value

The gantry pair ID.

If the input axis is not gantry axis, it will return -1.

Requirement

Minimum supported version	iA Studio 1.3

7.5 IsGantryPair

Purpose

To query whether the two axes are a gantry pair.

Syntax

```
int IsGantryPair(
    int axis_id_1
    int axis_id_2
);
```

Parameter

axis_id_1 [in]	Axis index 1.
axis_id_2 [in]	Axis index 2.

Return value

It will return an **int** value **TRUE** (1) if the two axes are at the “GantryPair” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.3

(This page is intentionally left blank.)

8. Group functions

8.1 Overview

HIMC provides axis group motion commands of multi-axis linear and circular simultaneous interpolation function, including LineAbs2D / 3D, LineRel2D / 3D, Arc2D, Circle2D, etc. Compared with axis motion commands, axis group motion commands ensure the synchronization of each axis in the group. The start time and the stop time of each axis motion are the same, and the controller will adjust the motion velocity of each axis based on the reference velocity given by users. The basic function of HIMC controller supports up to 4 axes for axis group motion commands (product model: MC-XX-XX-XX-00). If there is a need for 5-axis (or above axes') simultaneous machining for axis group motion function, please contact HIWIN Mikrosystem or local distributors for further information.

Figure 8.1.1 is the parameter flow diagram of HIMC axis group motion command. As the position feedback of each axis goes through the calculation of forward kinematics, axis group's position feedback in machine coordinate system (Cartesian Position Feedback) will be obtained. Based on the target command given by users, controller will plan the interpolation command in space (Cartesian Position Command) due to axis group's motion profile (as Figure 8.1.2 shows), and calculate the corresponding position command of each axis with inverse kinematics.

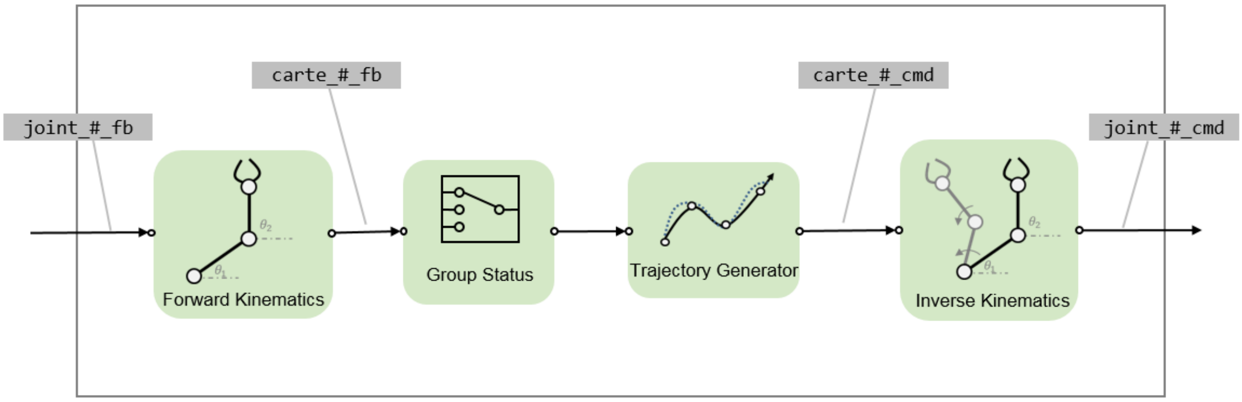


Figure 8.1.1

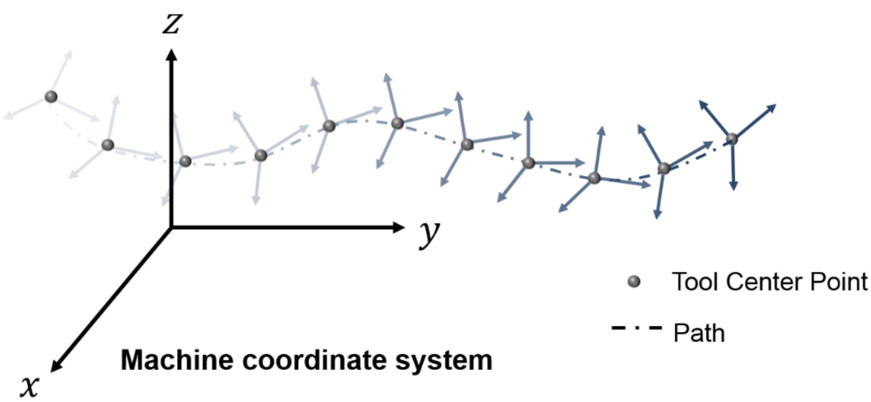


Figure 8.1.2

In axis group motion commands, HIMC will calculate the moving distance of each segment in space. Different from axis motion commands, the velocity planning is planned along axis group's moving direction in space, and the moving direction will change based on the direction of moving command.

Axis group motion commands are similar to axis motion commands; it also adopts S-Curve velocity planning, as Figure 8.1.3 shows. Axis group motion in space consists of two parts, translation and rotation. Translation command consists of the position commands of XYZ, while rotation command consists of the rotation commands of ABC. With axis group, users can set velocity planning parameters of translation and rotation, including profile generator's maximum velocity, maximum acceleration, maximum deceleration and smooth time.

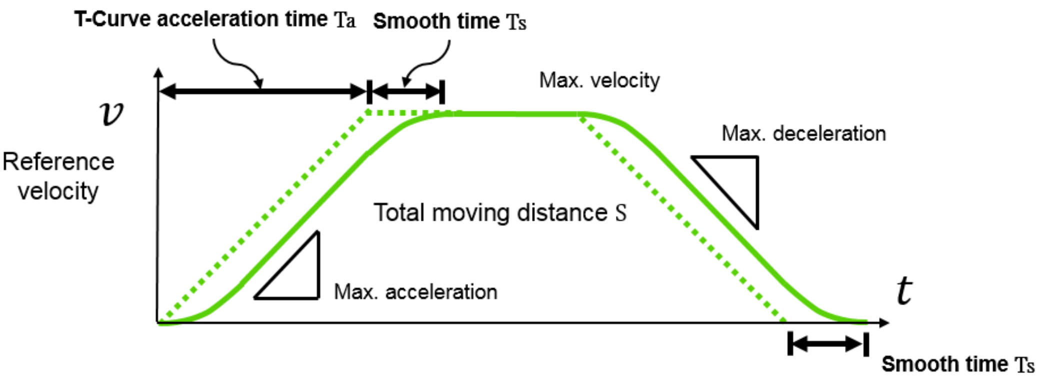


Figure 8.1.3

Each axis group motion command will be viewed as one segment, as Figure 8.1.4 shows. During the motion, based on the translation command and rotation command of each segment as well as the velocity planning parameters set by users, HIMC will calculate the move time of translation command and rotation command. The velocity planning parameters of the longer move time will be viewed as the feed rate of axis group. As for the shorter move time, it will move according to the motion command apportioned by the feed rate command.

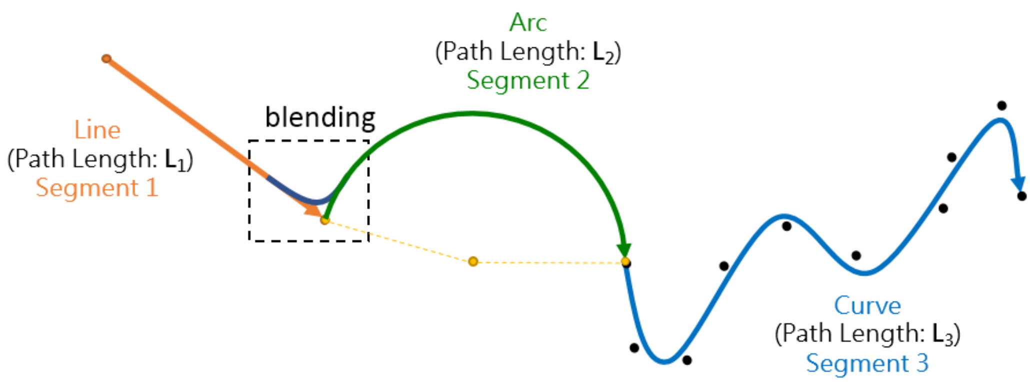


Figure 8.1.4

In HIMC, there is a built-in buffer for axis group commands. The segment of each motion command will be placed in this buffer; up to 512 motion commands can be accepted at the same time. Motion commands larger than this capacity limit will be discarded by the controller, and an error message will be displayed. Between the motion commands of segments, velocity and path will be planned based on the buffer mode and transition mode set by users. The planned velocity and path may change due to the selected mode. Take Figure 8.1.4 as an example, if buffer modes are used to set the velocity handover of each segment, the total length of this axis motion will be "$S = L_1(Line) + L_2(Arc) + L_3(Curve)$". Refer to section 8.1.4 and 8.1.5 for details.

Axis group motion status is similar to axis motion status; it can also be divided into "moving" and "in-positiion". During the motion, there are three phases like Figure 5.1.4 shown, including:

1. Axis group is moving and not in-position.
2. Axis group is not moving but not in-position.
3. Axis group is not moving and in-position.

Unlike axis motion command using target radius and debounce time to check whether the axis is in-position, axis group motion command checks whether all axes in the group are in-position. That is, if the axis group is in-position, all axes in the group are at the "in-position" state.

8.1.1 Group variables

Axis group variables are divided into three categories, motion command variables, profile generator variables and status variables. Users can select the desired variables via Scope Manager in iA Studio (refer to section 4.8 in "iA Studio User Guide"). Detailed descriptions are shown in Table 8.1.1.1 to Table 8.1.1.5.

Table 8.1.1.1 Motion command variables for axis group

Name	Variable	Unit	Description
Cartesian Position Command	carte_pose_cmd	mm or deg	Space position command for axis group in machine coordinated system (MCS). It is an array containing the value of [X Y Z A B C].
Cartesian Velocity Command	carte_vel_cmd	mm/s or deg/s	Space velocity command for axis group in machine coordinated system (MCS). It is an array containing the value of [X Y Z A B C].
Cartesian Position Feedback	carte_pose_fb	mm or deg	Space position feedback for axis group in machine coordinated system (MCS). It is an array containing the value of [X Y Z A B C].
Axis Position Command	joint_pos_cmd	mm or deg	Axis position command for axis group in axis coordinate system (ACS). It is an array.
Axis Velocity Command	joint_vel_cmd	mm/s or deg/s	Axis velocity command for axis group in axis coordinate system (ACS). It is an array.
Axis Acceleration Command	joint_acc_cmd	mm/s^2 or deg/s^2	Axis acceleration command for axis group in axis coordinate system (ACS). It is an array.
Axis Position Feedback	joint_pos_fb	mm or deg	Axis position feedback for axis group in axis coordinate system (ACS). It is an array.
Cartesian Position Error	carte_pose_err	mm or deg	Space position error for axis group in machine coordinated system (MCS). It is an array containing the value of [X Y Z A B C].
Reference Group Position	grp_pg_pos	mm or deg	Reference position for axis group. It is the position set-point generated from the profile generator according to axis group command's motion profile.
Reference Group Velocity	grp_pg_vel	mm/s or deg/s	Reference velocity for axis group. It is the velocity set-point generated from the profile generator according to axis group command's motion profile.
Reference Group Acceleration	grp_pg_acc	mm/s^2 or deg/s^2	Reference acceleration for axis group. It is the acceleration set-point generated from the profile generator according to axis group command's motion profile.

Table 8.1.1.2 Profile generator variables for axis group

Name	Variable	Unit	Description
Group Max. Linear Profile Velocity	grp_lin_vel	mm/s	Maximum linear profile velocity for axis group. Not necessarily reached.
Group Max. Linear Profile Acceleration	grp_lin_acc	mm/s^2	Maximum linear profile acceleration for axis group. Not necessarily reached.
Group Max. Linear Profile Deceleration	grp_lin_dec	mm/s^2	Maximum linear profile deceleration for axis group. Not necessarily reached.
Group Linear Smooth Time	grp_lin_sf	ms	Linear profile smooth time for axis group. Its input range is from 0 to 500. Increasing the value can reduce mechanical vibration during motion, but the total motion time will be affected.
Group Max. Angular Profile Velocity	grp_ang_vel	deg/s	Maximum angular profile velocity for axis group. Not necessarily reached.
Group Max. Angular Profile Acceleration	grp_ang_acc	deg/s^2	Maximum angular profile acceleration for axis group. Not necessarily reached.
Group Max. Angular Profile Deceleration	grp_ang_dec	deg/s^2	Maximum angular profile deceleration for axis group. Not necessarily reached.
Group Angular Smooth Time	grp_ang_sf	ms	Angular profile smooth time for axis group. Its input range is from 0 to 500. Increasing the value can reduce mechanical vibration during motion, but the total motion time will be affected.

Table 8.1.1.3 Status variables for axis group

Name	Variable	Unit	Description
Group Fault Status	grp_fault_status	N/A	Error status of axis group; refer to Table 8.1.1.4 for bit definition.
Group Motion Status	grp_motion_status	N/A	Motion status of axis group; refer to Table 8.1.1.5 for bit definition.

Table 8.1.1.4 Bit definition for axis group error status

Bit	Name	Description	Default Response
0	Error Stop	Axis group at “error stop” state	N/A
1	Axis Fault	Slave drive fault	Controller disables the axis; axis group is out of sync.
2	Software Limit	Axis software limit triggered	Controller stops the motion; axis group is out of sync.

Table 8.1.1.5 Bit definition for axis group motion status

Bit	Name	Description	Remark
0	Enabled	The axis group is enabled.	N/A
1	Moving	The axis group is moving.	N/A
2	In Position	The axis group is in-position.	All the axes in the axis group are in-position.
3	Input Shape	Enable axis group’s Input Shape filter.	Refer to section 15.1.

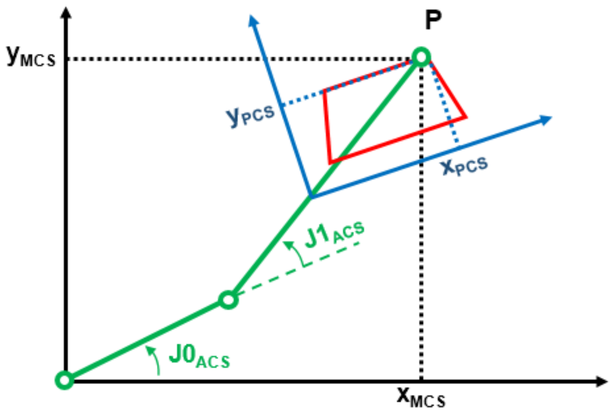
8.1.2 Coordinate systems

Table 8.1.2.1 shows the definition and description of HIMC's coordinate systems, including Axis Coordinate System (ACS), Machine Coordinate System (MCS), Product Coordinate System (PCS), Workpiece Coordinate System (WCS), Global Coordinate System and Coordinate Offset.

Table 8.1.2.1

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 kinematics transformation (refer to section 8.1.3). It has 6 dimensions in total to indicate the position and orientation in space (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, and it can set coordinate transformation parameters.
`CS_WCS#` `(#=1~15)`	Workpiece Coordinate System. It is used to set workpiece zero point, and it provides up to 15 independent workpiece coordinate systems. The default is no offset. It depends on Product Coordinate System, so it can set coordinate transformation parameters.
`CS_GLOBAL`	Global Coordinate System. It is used to set global zero point, and it can establish the global spatial relationship of each axis group. Not supported yet.
`CS_OFFSET`	Coordinate Offset. It is used to set temporary zero point. The default is no offset, that is, the coordinate origin of offset is the coordinate origin of machine. It depends on Product Coordinate System, so it can set coordinate transformation parameters.

Figure 8.1.2.1 shows an example of the relationship among ACS, MCS and PCS for a SCARA robot with two rotary axes. ACS and MCS are transformed through forward and inverse kinematics (refer to section 8.1.3). On the other hand, there is a coordinate transformation relationship between MCS and PCS. The position on the coordinate system is obtained through the translation and the rotation of the coordinate.



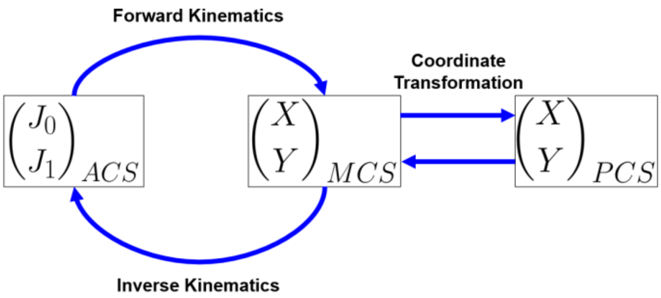


Figure 8.1.2.1

When transforming MCS into PCS, HIMC can set machine's workpiece coordinate (WCS1~15) and coordinate offset (OFFSET) based on requirement. For the setting of coordinate system, 3 translational degrees of freedom (X, Y, Z) and 3 rotational degrees of freedom (A, B, C) are used to indicate the pose in space.

HIMC adopts the "Roll-Pitch-Yaw" rotation convention in fixed angle. As Figure 8.1.2.2 shows, the degree of freedom to rotate along the X axis is **Roll**, angle **A**; the degree of freedom to rotate along the Y axis is **Pitch**, angle **B**; the degree of freedom to rotate along the Z axis is **Yaw**, angle **C**. This rotation convention is the same as using the sequence of ZYX in Tait-Bryan angles to indicate the orientation of object in space, as Figure 8.1.2.3 shows.

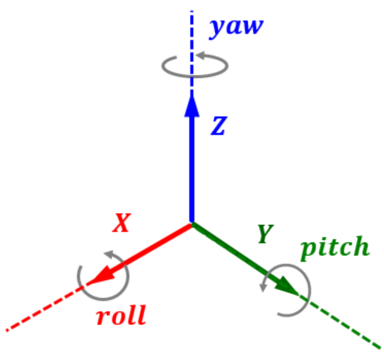


Figure 8.1.2.2

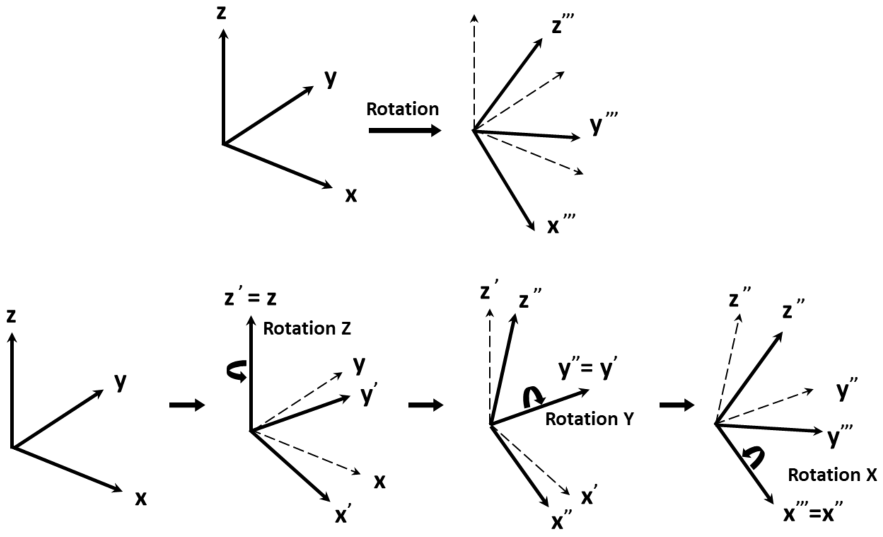


Figure 8.1.2.3

If there is no coordinate offset, the relationship between each WCS and MCS is shown in Figure 8.1.2.4.

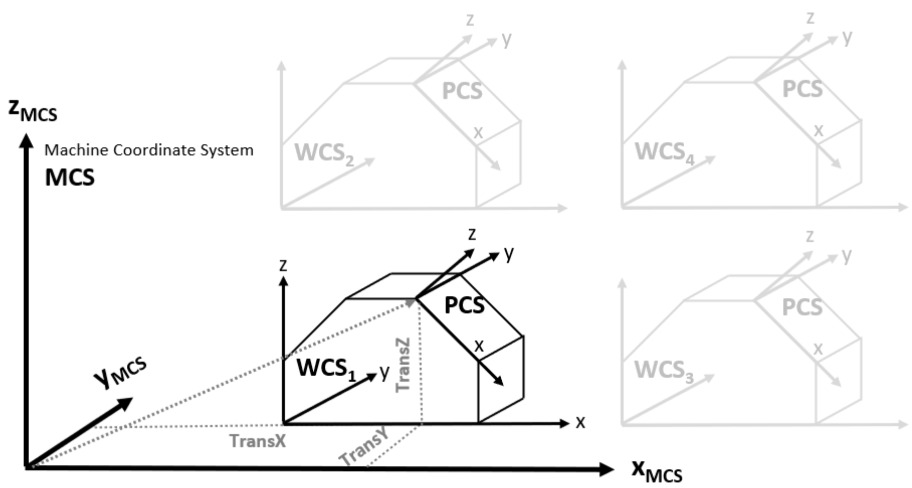


Figure 8.1.2.4

If coordinate offset is added, the relationship between WCS and MCS is shown in Figure 8.1.2.5. The transformation of coordinate offset will be added.

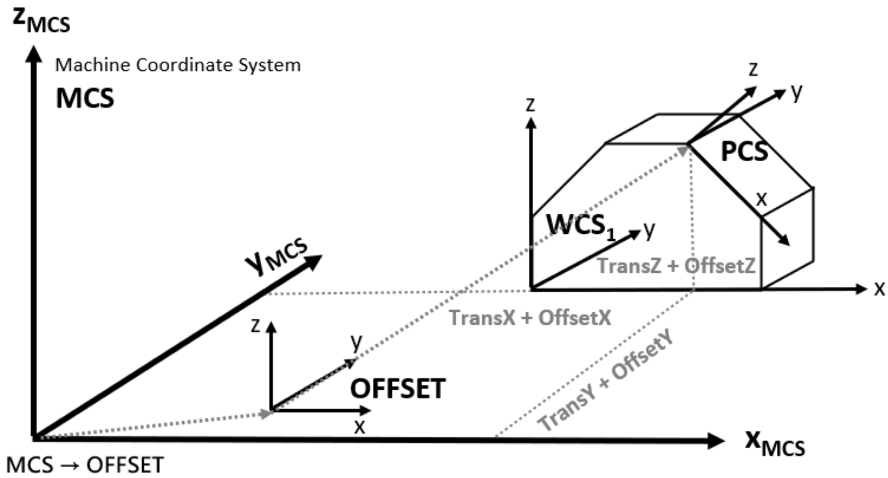


Figure 8.1.2.5

Based on the functions mentioned above, users can define the parameters for coordinate transformation in HIMC and establish the transformation relationship among the coordinate systems. Figure 8.1.2.6 shows the relationship among the coordinate systems. To help users understand easily, the figure only shows the coordinate of XY plane. In actual application, users can set 6 degrees of freedom (X, Y, Z, A, B, C) for coordinate transformation.

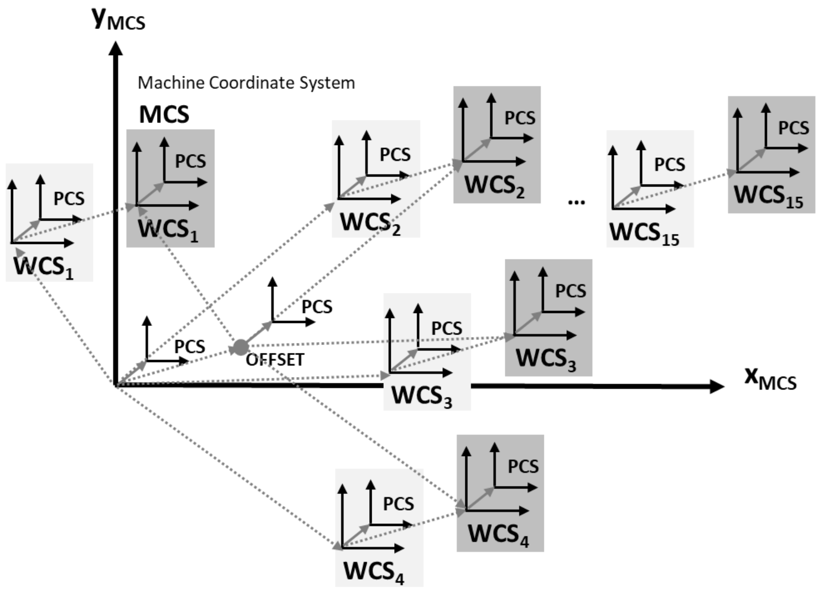


Figure 8.1.2.6

8.1.3 Kinematics

Kinematics mainly deals with the transformation between ACS (Axis Coordinate System) and MCS (Machine Coordinate System). Forward kinematics is the calculation from each axis' position feedback in ACS to the coordinate position of MCS. On the contrary, inverse kinematics is the calculation from coordinate position of MCS to each axis' position in ACS. Table 8.1.3.1 shows the definition of kinematics configuration provided by HIMC.

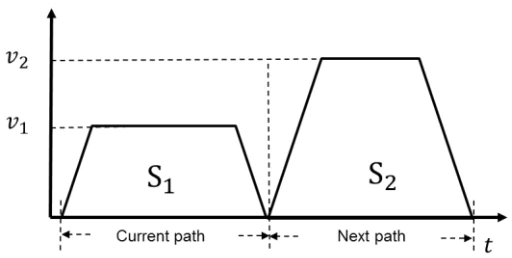
Table 8.1.3.1

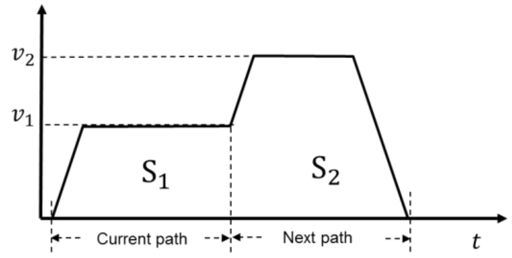
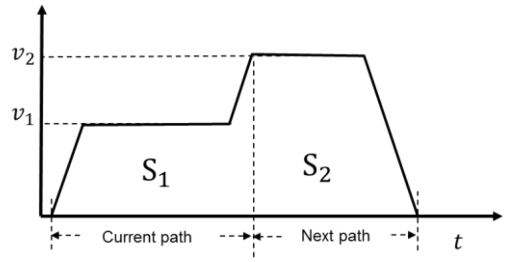
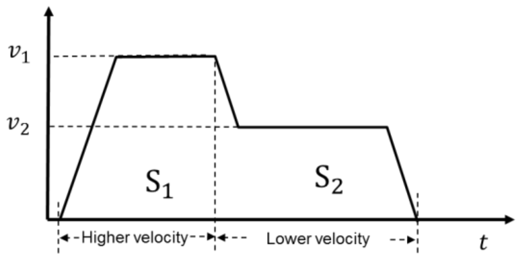
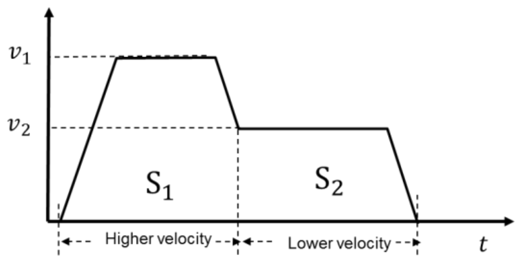
ID	Name	Description
1	Cartesian	Map each axis in the coordinated motion group to X, Y, Z, A, B, C axis of Cartesian coordinate respectively. The maximum allowable number of axes in joint space is 6. (Default for axis group)
2	SCARA	(Not supported)
3	WAFER	(Not supported)
4	6-Axis Articulated Robot	(Not supported)

8.1.4 Buffer modes

Buffer mode determines the velocity profile at the end-points of adjacent paths. Users can use this setting to plan the profile velocity of two adjacent paths. Table 8.1.4.1 shows the definition of buffer modes provided by HIMC.

Table 8.1.4.1

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) 

8.1.5 Transition modes

Transition mode determines the type of the transition curve between adjacent paths. With this setting, HIMC will do the calculation of corner smoothing between two linear motion commands based on the set parameter (refer to Table 8.1.5.1). To achieve a better planning path, this will affect the original motion profile.

Table 8.1.5.1

HMPL definition	Description	Corresponding parameter	Unit
`TM_NONE`	None: Insert no transition curve (Default mode)	N/A	N/A
`TM_START_VEL` (Not supported)	Start velocity	TPStartVelocity	mm/s or deg/s
`TM_CONST_VEL`	Constant velocity	TPVelocity	mm/s or deg/s
`TM_CORNER_DIST`	Corner distance	TPCornerDistance	mm or deg
`TM_MAX_CORNER_DEV`	Max. corner deviation	TPCornerDeviation	mm or deg
`TM_MAX_CORNER_CURV`	Max. corner curvature	TPCornerCurv	K

When using advanced group motion control’s command “LinAbs” and “LinRel” with transition mode `TM_CONST_VEL` to assign the velocity of circular motion, the distance from circular start position to corner is determined by transition mode’s distance parameter set in section 8.3.26, as Figure 8.1.5.1 shows. With transition mode `TM_CORNER_DIST` to assign the distance from circular start position to corner, the velocity of circular motion is determined by transition mode’s velocity parameter set in section 8.3.26.

If the smoothing circular radius calculated by transition mode exceeds the length of any segment, the transition mode function between the segments will be ignored.

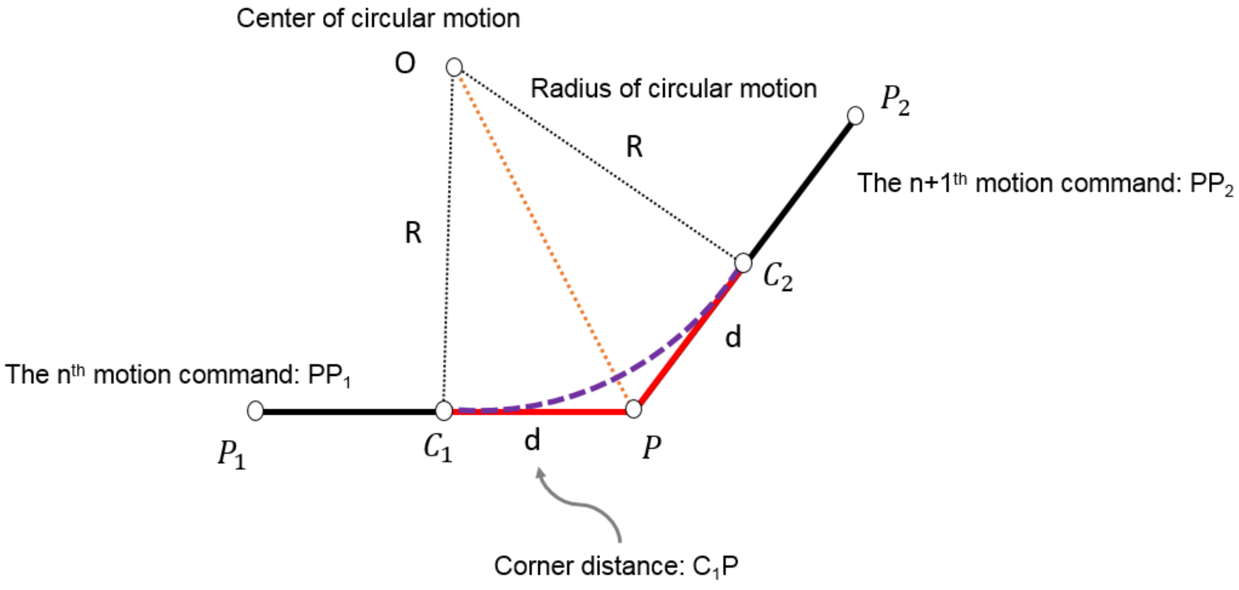

Center of circular motion
O
Radius of circular motion
R
R
P_2
The n+1th motion command: PP2
C_2
d
The nth motion command: PP1
P_1
C_1
d
P
Corner distance: C1P

Figure 8.1.5.1

8.1.6 Example

Example 1: Basic group setup and linear motion

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

```
void 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, 100, 5000, 5000, 200);
    //  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, 100, 100);  //  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.

Example 2: Advanced group setup and velocity blending

The way to move tool center point along the two-dimensional paths in Figure 8.1.6.1 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 8.1.4 to know more about “BM_PREV” and “BM_NEXT”.

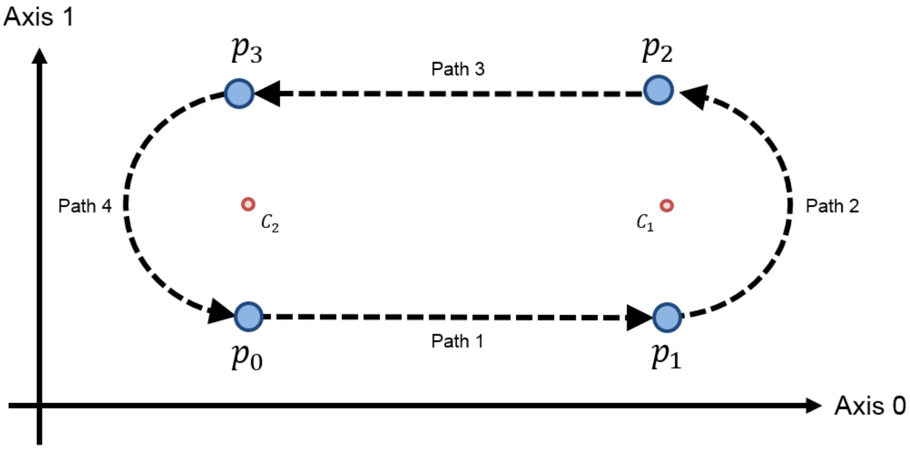


Figure 8.1.6.1

```
void 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] = {100, 50, 0};
    double c2[3] = {0, 50, 0};
    double p0[6] = {0, 0, 0, 0, 0, 0};
    double p1[6] = {100, 0, 0, 0, 0, 0};
    double p2[6] = {100, 100, 0, 0, 0, 0};
    double p3[6] = {0, 100, 0, 0, 0, 0};

    double norm_ccw[3] = {0, 0, 1};
    double vel[4] = {100, 5000, 5000, 50};

    double trans_para[4] = {0, 0, 0, 0};

    LineAbs(gid, p0, vel, CS_MCS, BM_BUFF, TM_NONE, trans_para);
    Till(IsGrpInPos(gid));

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

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

Example 3-1: Transition (line-to-line)

The way to move tool center point along the two-dimensional paths in Figure 8.1.6.2 is shown in the following HMPL task. The group motion profile starts from p_0. After it goes through p_1, p_2, p_3 and gets back to p_0, set the transition mode as "TM_CORNER_DIST", and set its velocity and distance. When the second motion goes through p_1, p_2, p_3, p_0, the paths will be automatically modified to the red solid lines in Figure 8.1.6.2.

Note: Refer to section 8.1.5 to know more about "TM_CORNER_DIST".

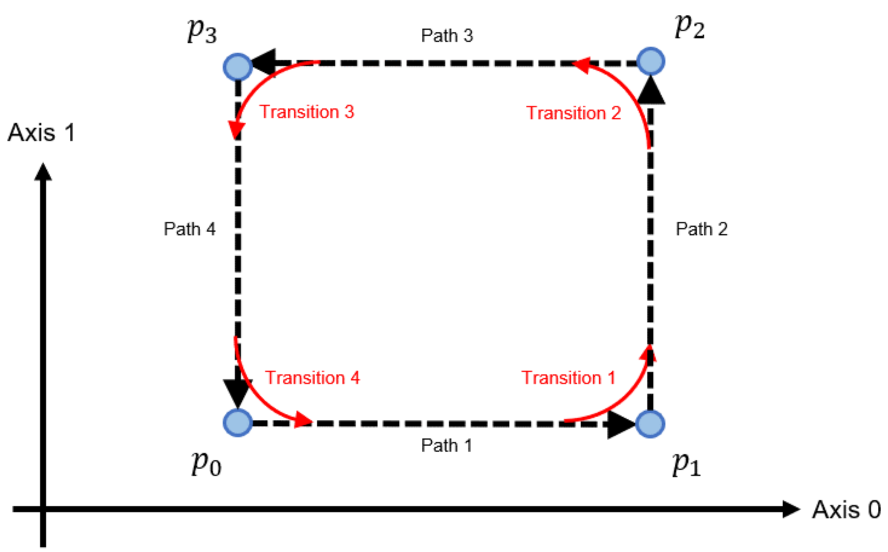


Figure 8.1.6.2

```
typedef struct {
    double x, y;
} Point2D;

//  Set space points
Point2D p0 = {.x = 0, .y = 0};
Point2D p1 = {.x = 100, .y = 0};
Point2D p2 = {.x = 100, .y = 100};
Point2D p3 = {.x = 0, .y = 100};

void RectangularMotion(int gid) {
    LineAbs2D(gid, p0.x, p0.y);
    LineAbs2D(gid, p1.x, p1.y);
    LineAbs2D(gid, p2.x, p2.y);
    LineAbs2D(gid, p3.x, p3.y);
```

```
    LineAbs2D(gid, p0.x, p0.y);
}

void main() {
    int axis[2] = {0, 1};   //  axis index
    int gid = 0;            //  group index
    double round_vel = 50;  //  transition velocity
    double round_dis = 20;  //  transition distance
    double round_dev = 1;   //  transition maximum deviation
    double round_curv = 5;  //  transition curvature


    //  Create and enable an axis group
    UngrpAllAxes(gid);
    Enable(axis[0]);       Enable(axis[1]);
    Till(IsEnabled(axis[0]) && IsEnabled(axis[1]));
    SetupGroup(gid, axis[0], axis[1]);

    //  rectangular motion profile
    RectangularMotion(gid);
    Till(IsGrpInPos(gid));

    //  Enable transition function and set its velocity and distance
    SetGrpTransMode(gid, TM_CORNER_DIST);
    SetGrpTransPrm(gid, round_vel, round_dis, round_dev, round_curv);

    //  rectangular motion profile
    RectangularMotion(gid);

    //  Set end position
    LineAbs2D(gid, p0.x + round_dis, p0.y);
    Till(IsGrpInPos(gid));

    //  Disable transition function
    SetGrpTransMode(gid, TM_NONE);
}
```

Example 3-2: Transition (circular-to-line, line-to- circular, circular-to- circular)

The way to move tool center point along the two-dimensional paths in Figure 8.1.6.3 is shown in the following HMPL task. The group motion profile starts from p_0. After it goes through p_1, p_2, p_3, p_4 and gets back to p_0, set the transition mode as "TM_CORNER_DIST", and set its velocity and distance. When the second motion goes through p_1, p_2, p_3, p_4, p_0, the paths will be automatically modified to the red solid lines in Figure 8.1.6.3.

Note: Refer to section 8.1.5 to know more about "TM_CORNER_DIST".

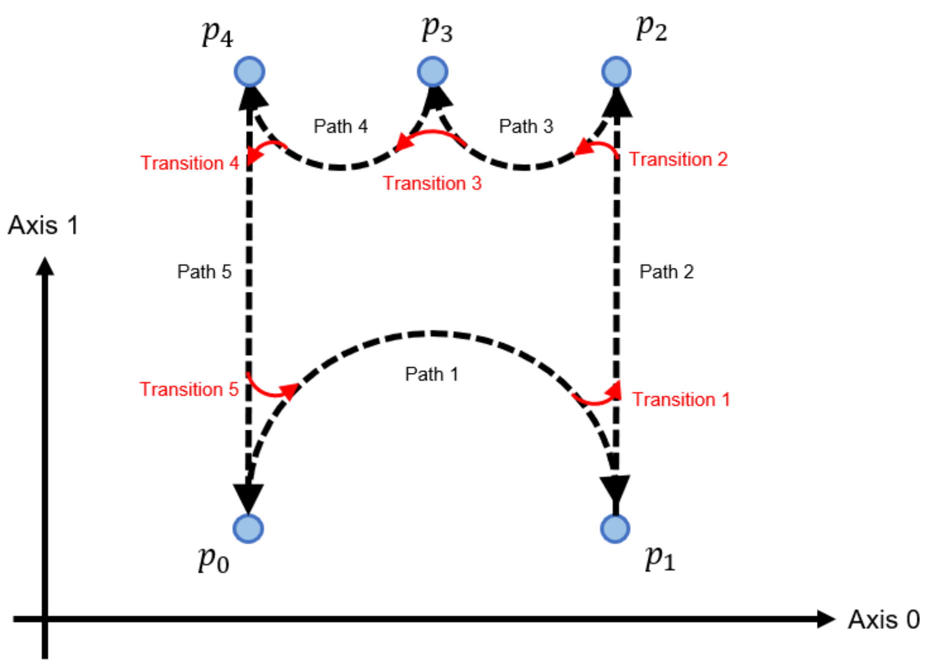


Figure 8.1.6.3

```
typedef struct {
    double x, y;
} Point2D;

//  Set space points
Point2D pt0 = {.x = 0, .y = 0};
Point2D pt1 = {.x = 100, .y = 0};
Point2D pt2 = {.x = 100, .y = 100};
Point2D pt3 = {.x = 50, .y = 100};
Point2D pt4 = {.x = 0, .y = 100};

Point2D c1 = {.x = 50, .y = 0};
Point2D c2 = {.x = 75, .y = 100};
Point2D c3 = {.x = 25, .y = 100};
```

```
void Motion(int gid) {
    LineAbs2D(gid, pt0.x, pt0.y);
    Circle2D(gid, c1.x, c1.y, pt1.x, pt1.y, -1);
    LineAbs2D(gid, pt2.x, pt2.y);
    Circle2D(gid, c2.x, c2.y, pt3.x, pt3.y, -1);
    Circle2D(gid, c3.x, c3.y, pt4.x, pt4.y, -1);
    LineAbs2D(gid, pt0.x, pt0.y);
}

void main() {
    int axis[2] = {0, 1};    //  axis index
    int gid = 0;            //  group index
    double round_vel = 50;  //  transition velocity
    double round_dis = 20;  //  transition distance

    //  Create and enable an axis group
    UngrpAllAxes(gid);
    Enable(axis[0]); Enable(axis[1]);
    Till(IsEnabled(axis[0]) && IsEnabled(axis[1]));
    SetupGroup(gid, 0, 1);

    //  Disable transition function
    SetGrpTransMode(gid, TM_NONE);

    //  Motion profile
    Motion(gid);
    Till(IsGrpInPos(gid))

    //  Enable transition function and set its velocity and distance
    SetGrpTransMode(gid, TM_CORNER_DIST);
    SetGrpTransPrm(gid, round_vel, round_dis, 0, 0);

    //  Motion profile
    Motion(gid);
    Circle2D(gid, c1.x, c1.y, 0.5*(pt0.x + pt1.x), 0.5*(pt0.x + pt1.x), -1);
    Till(IsGrpInPos(gid))
```

```
    //  Disable transition function
    SetGrpTransMode(gid, TM_NONE);
}
```

Example 4: Three-dimensional circular motion and spiral motion

```
void main() {

    int axis[3] = {0, 1, 2};  //  axis index
    int gid = 0;  //  group index

    double center1[3] = {0, 50, 0};          //  center of circle 1
    double center2[3] = {0, 0, 50};          //  center of circle 2
    double end_pos[6] = {0, 0, 0, 0, 0, 0};  //  end position
    double norm_x[3] = {1, 0, 0};            //  normal vector x
    double norm_y[3] = {0, 1, 0};            //  normal vector y
    double norm_z[3] = {0, 0, 1};            //  normal vector z
    double vel[4] = {100, 5000, 5000, 50};

    //  Create and enable an axis group
    UngrpAllAxes(gid);
    Enable(axis[0]); Enable(axis[1]); Enable(axis[2]);
    Till(IsEnabled(axis[0]) && IsEnabled(axis[1]) && IsEnabled(axis[2]));
    SetupGroup(gid, axis[0], axis[1], axis[2]);

    //  Move to coordinate (0, 0, 0)
    LineAbs3D(gid, 0, 0, 0);

    //  three-dimensional circular motion
    CircleAbs(gid, center1, norm_z, 1, end_pos, vel, CS_MCS, BM_BUFF, TM_NONE, 0);
    CircleAbs(gid, center2, norm_y, 1, end_pos, vel, CS_MCS, BM_BUFF, TM_NONE, 0);
    CircleAbs(gid, center2, norm_x, 1, end_pos, vel, CS_MCS, BM_BUFF, TM_NONE, 0);
    end_pos[2] = 100;

    //  spiral motion
    CircleAbs(gid, center1, norm_z, 5, end_pos, vel, CS_MCS, BM_BUFF, TM_NONE, 0);

    //  Move to coordinate (0, 0, 0)
    LineAbs3D(gid, 0, 0, 0);
    Till(IsGrpInPos(gid));
}
```

Example 5-1: Coordinate transformation (MCS→PCS)

Generate an arrow motion profile on Machine Coordinate System (MCS). Instead of changing the values of the original motion profile, set the reference coordinate as Product Coordinate System (PCS) and its transformation parameters. By doing so, MCS can be transformed into PCS, as Figure 8.1.6.4 shows.

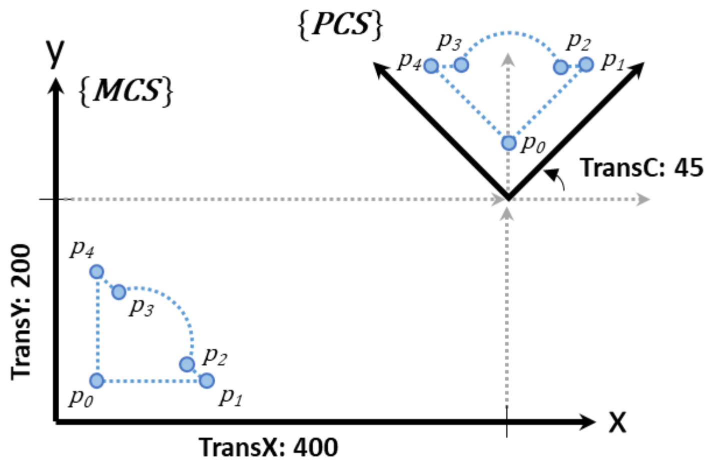


Figure 8.1.6.4

```
typedef struct {
    double x, y;
} Point2D;

//  Set space points
Point2D p0 = {.x = 10, .y = 10};
Point2D p1 = {.x = 110, .y = 10};
Point2D p2 = {.x = 100, .y = 40};
Point2D p3 = {.x = 40, .y = 100};
Point2D p4 = {.x = 10, .y = 110};
Point2D center = {.x = 60, .y = 60};

void ArrowMotion(int gid) {
    LineAbs2D(gid, p0.x, p0.y);
    LineAbs2D(gid, p1.x, p1.y);
    LineAbs2D(gid, p2.x, p2.y);
    Circle2D(gid, center.x, center.y, p3.x, p3.y, 0);
    LineAbs2D(gid, p4.x, p4.y);
    LineAbs2D(gid, p0.x, p0.y);
}
```

```
void main() {
    int axis[2] = {0, 1};   //  axis index
    int gid = 0;            //  group index

    //  Create and enable an axis group
    UngrpAllAxes(gid);
    Enable(axis[0]);       Enable(axis[1]);
    Till(IsEnabled(axis[0]) && IsEnabled(axis[1]));
    SetupGroup(gid, axis[0], axis[1]);   //  axis 0 is X axis, axis 1 is Y axis

    //  arrow motion profile
    ArrowMotion(gid);
    Till(IsGrpInPos(gid));

    //  Set Product coordinate transformation parameters:
    //  X translates 400 mm, Y translates 200 mm, Z rotates 45 deg
    double transfer[6] = {400, 200, 0, 0, 0, 45};
    SetGrpCoordTrans(gid, CS_PCS, transfer);

    //  Refer to Product Coordinate System
    SetGrpCoordSys(gid, CS_PCS);

    //  arrow motion profile
    ArrowMotion(gid);

    //  Refer to Machine Coordinate System
    SetGrpCoordSys(gid, CS_MCS);

    //  Set end position
    LineAbs2D(gid, 0, 0);
    Till(IsGrpInPos(gid));

    //  Clear coordinate transformation parameters
    double zeros[6] = {0, 0, 0, 0, 0, 0};
    SetGrpCoordTrans(gid, CS_PCS, zeros);
}
```

Example 5-2: Coordinate transformation (MCS→WCSn→PCS)

The concept is the same as Example 5-1. Generate an arrow motion profile on Machine Coordinate System (MCS). Instead of changing the values of the original motion profile, add Workpiece Coordinate System (WCS) to the reference coordinate. By doing so, the motion profile of the original Product Coordinate System (PCS) can be transferred to each workpiece coordinate, as Figure 8.1.6.5 shows.

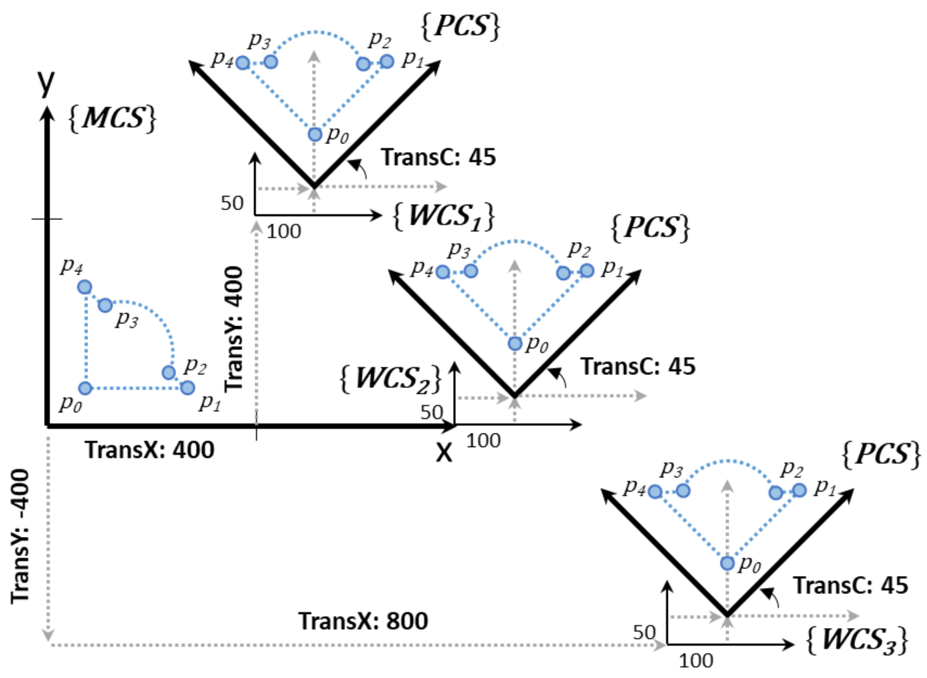


Figure 8.1.6.5

```
typedef struct {
    double x, y;
} Point2D;

//  Set space points
Point2D p0 = {.x = 10, .y = 10};
Point2D p1 = {.x = 110, .y = 10};
Point2D p2 = {.x = 100, .y = 40};
Point2D p3 = {.x = 40, .y = 100};
Point2D p4 = {.x = 10, .y = 110};
Point2D center = {.x = 60, .y = 60};

void ArrowMotion(int gid) {
    LineAbs2D(gid, p0.x, p0.y);
    LineAbs2D(gid, p1.x, p1.y);
```

```
    LineAbs2D(gid, p2.x, p2.y);
    Circle2D(gid, center.x, center.y, p3.x, p3.y, 0);
    LineAbs2D(gid, p4.x, p4.y);
    LineAbs2D(gid, p0.x, p0.y);
}

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

    //  Create and enable an axis group
    UngrpAllAxes(gid);
    Enable(axis[0]);      Enable(axis[1]);
    Till(IsEnabled(axis[0]) && IsEnabled(axis[1]));
    SetupGroup(gid, axis[0], axis[1]);   //  axis 0 is X axis, axis 1 is Y axis

    //  arrow motion profile
    ArrowMotion(gid);
    Till(IsGrpInPos(gid));

    //  Set Product coordinate transformation parameters:
    //  X translates 100 mm, Y translates 50 mm, Z rotates 45 deg
    double transfer[6] = {100, 50, 0, 0, 0, 45};

    //  Set transformation parameters of workpiece coordinate 1~3
    double wcs1[6] = {400, 400, 0, 0, 0, 0};
    double wcs2[6] = {600, 0, 0, 0, 0, 0};
    double wcs3[6] = {800, -400, 0, 0, 0, 0};

    SetGrpCoordTrans(gid, CS_PCS, transfer);
    SetGrpCoordTrans(gid, CS_WCS1, wcs1);
    SetGrpCoordTrans(gid, CS_WCS2, wcs2);
    SetGrpCoordTrans(gid, CS_WCS3, wcs3);

    //  Refer to Workpiece Coordinate System 1 & Product Coordinate System
    SetGrpCoordSys(gid, CS_WCS1 | CS_PCS);
    ArrowMotion(gid);
```

```
    //  Refer to Workpiece Coordinate System 2 & Product Coordinate System
    SetGrpCoordSys(gid, CS_WCS2 | CS_PCS);
    ArrowMotion(gid);

    //  Refer to Workpiece Coordinate System 3 & Product Coordinate System
    SetGrpCoordSys(gid, CS_WCS3 | CS_PCS);
    ArrowMotion(gid);

    //  Refer to Machine Coordinate System
    SetGrpCoordSys(gid, CS_MCS);

    //  Set end position
    LineAbs2D(gid, 0, 0);
    Till(IsGrpInPos(gid));

    //  Clear coordinate transformation parameters
    double zeros[6] = {0, 0, 0, 0, 0, 0};
    SetGrpCoordTrans(gid, CS_PCS, zeros);
    SetGrpCoordTrans(gid, CS_WCS1, zeros);
    SetGrpCoordTrans(gid, CS_WCS2, zeros);
    SetGrpCoordTrans(gid, CS_WCS3, zeros);
}
```

Example 5-3: Coordinate transformation (MCS→OFFSET→WCSn→PCS)

This example is the extension of Example 5-2. With Coordinate Offset (OFFSET) provided by HIMC, the result of Example 5-2 can be offset, as Figure 8.1.6.6 shows.

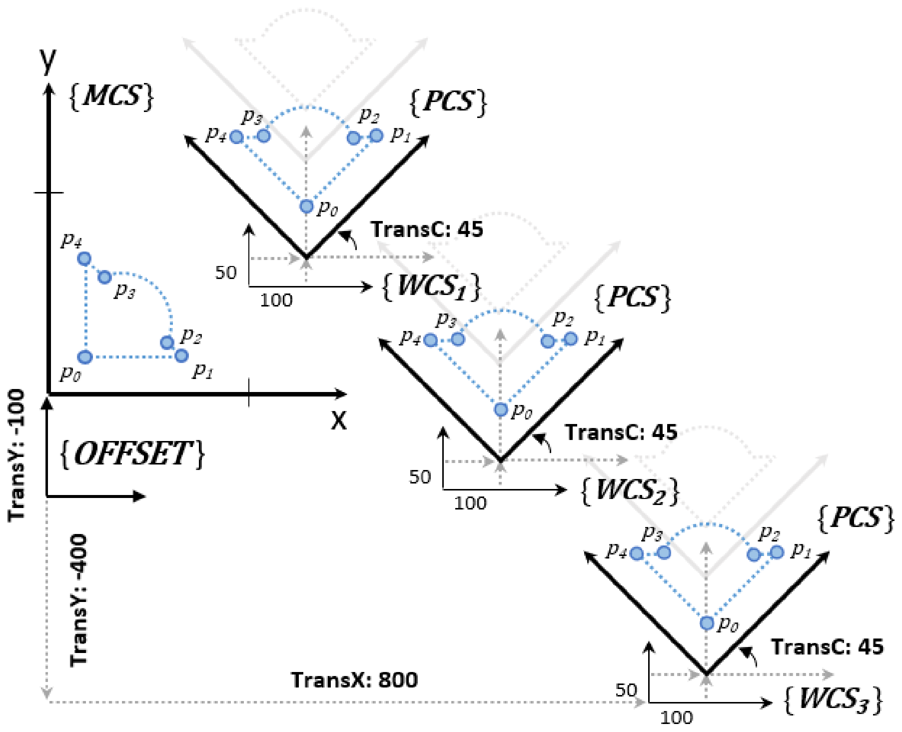


Figure 8.1.6.6

```
typedef struct {
    double x, y;
} Point2D;

//  Set space points
Point2D p0 = {.x = 10, .y = 10};
Point2D p1 = {.x = 110, .y = 10};
Point2D p2 = {.x = 100, .y = 40};
Point2D p3 = {.x = 40, .y = 100};
Point2D p4 = {.x = 10, .y = 110};
Point2D center = {.x = 60, .y = 60};

void ArrowMotion(int gid) {
    LineAbs2D(gid, p0.x, p0.y);
    LineAbs2D(gid, p1.x, p1.y);
    LineAbs2D(gid, p2.x, p2.y);
```

```
    Circle2D(gid, center.x, center.y, p3.x, p3.y, 0);
    LineAbs2D(gid, p4.x, p4.y);
    LineAbs2D(gid, p0.x, p0.y);
}

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

    //  Create and enable an axis group
    UngrpAllAxes(gid);
    Enable(axis[0]);       Enable(axis[1]);
    Till(IsEnabled(axis[0]) && IsEnabled(axis[1]));
    SetupGroup(gid, axis[0], axis[1]);   //  axis 0 is X axis, axis 1 is Y axis

    //  arrow motion profile
    ArrowMotion(gid);
    Till(IsGrpInPos(gid));

    //  Set Product coordinate transformation parameters:
    //  X translates 100 mm, Y translates 50 mm, Z rotates 45 deg
    double transfer[6] = {100, 50, 0, 0, 0, 45};

    //  Set transformation parameters of workpiece coordinate 1~3
    double wcs1[6] = {400, 400, 0, 0, 0, 0};
    double wcs2[6] = {600, 0, 0, 0, 0, 0};
    double wcs3[6] = {800, -400, 0, 0, 0, 0};

    //  Set transformation parameters of Coordinate Offset
    double offset[6] = {0, -100, 0, 0, 0, 0};

    SetGrpCoordTrans(gid, CS_PCS, transfer);
    SetGrpCoordTrans(gid, CS_WCS1, wcs1);
    SetGrpCoordTrans(gid, CS_WCS2, wcs2);
    SetGrpCoordTrans(gid, CS_WCS3, wcs3);
    SetGrpCoordTrans(gid, CS_OFFSET, offset);
```

```
    //  Refer to Coordinate Offest & Workpiece Coordinate System 1
    //  & Product Coordinate System
    SetGrpCoordSys(gid, CS_OFFSET | CS_WCS1 | CS_PCS);
    ArrowMotion(gid);

    //  Refer to Coordinate Offest & Workpiece Coordinate System 2
    //  & Product Coordinate System
    SetGrpCoordSys(gid, CS_OFFSET | CS_WCS2 | CS_PCS);
    ArrowMotion(gid);

    //  Refer to Coordinate Offest & Workpiece Coordinate System 3
    //  & Product Coordinate System
    SetGrpCoordSys(gid, CS_OFFSET | CS_WCS3 | CS_PCS);
    ArrowMotion(gid);

    //  Refer to Machine Coordinate System
    SetGrpCoordSys(gid, CS_MCS);

    //  Set end position
    LineAbs2D(gid, 0, 0);
    Till(IsGrpInPos(gid));

    //  Clear coordinate transformation parameters
    double zeros[6] = {0, 0, 0, 0, 0, 0};
    SetGrpCoordTrans(gid, CS_PCS, zeros);
    SetGrpCoordTrans(gid, CS_WCS1, zeros);
    SetGrpCoordTrans(gid, CS_WCS2, zeros);
    SetGrpCoordTrans(gid, CS_WCS3, zeros);
    SetGrpCoordTrans(gid, CS_OFFSET, zeros);
}
```

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

Requirement

Minimum supported version	iA Studio 0.23

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

8.2.3 ResetGroup

Purpose

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

Syntax

```
int ResetGroup(
    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 1.3

8.2.4 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

8.2.5 HaltGroup

Purpose

To halt the motion of an axis group; its velocity will be set as 0.

Syntax

```
int HaltGroup(
    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

If the axis group is not in-position, it will keep moving.

Requirement

Minimum supported version	iA Studio 1.3

8.2.6 ResumeGroup

Purpose

To resume the motion of an axis group from “halt” status.

Syntax

```
int ResumeGroup(
    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 1.3

8.2.7 JogGroup

Purpose

To make an axis group start a never-ending motion at a specific velocity in the designated direction of machine coordinate system.

Syntax

```
int JogGroup(
    int    group_id,
    int    carte_dir,
    double jog_vel
);
```

Parameter

group_id [in]	Axis group index.
carte_dir [in]	The direction of motion in the machine coordinate system. The number 0 ~ 5 orderly repersents 6-DOF {X, Y, Z, A, B, C} in the machine coordinate system.
jog_vel [in]	The value of a specific velocity. Its positive / negative value represents the same / reverse direction movement of the direction of motion. Parameter unit: mm/s or deg/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 1.4

8.2.8 JogGroupAxis

Purpose

To make the specific axis in an axis group start a never-ending motion at a specific velocity in the axis coordinate system.

Syntax

```
int JogGroupAxis(
    int    group_id,
    int    grp_axis,
    double jog_vel
);
```

Parameter

group_id [in]	Axis group index.
grp_axis [in]	Axis index in the axis group. The number 0 ~ 8 orderly repersents the sequence that each axis is added to the axis group.
jog_vel [in]	The value of a specific velocity. Its positive / negative value represents the same / reverse direction movement of the direction of motion. Parameter unit: mm/s or deg/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 1.4

8.2.9 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 = 100;
    double target_y = 100;
    LineAbs2D (0 /* group_id */, target_x, target_y);
    //  Move to (target_x, target_y)
    //  that is (100, 100)
}
```

Requirement

Minimum supported version	iA Studio 0.24

8.2.10 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

8.2.11 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 (100, 200)
    double distance_x = 100;
    double distance_y = 100;
    LineRel2D (0 /* group_id */, distance_x, distance_y);
    //  Move to (X starting position + distance_x, Y starting position + distance_y)
    //  that is (200, 300)
}
```

Requirement

Minimum supported version	iA Studio 0.24

8.2.12 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

8.2.13 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

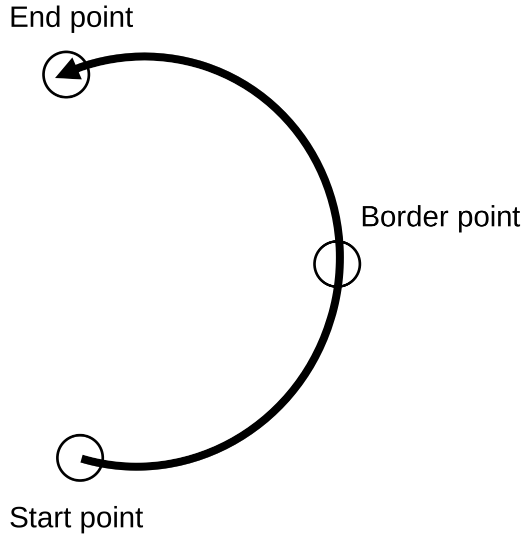
```
void main() {
    //  Assume that axis group 0 which includes two orthogonal axes is enabled
    //  and the starting positions of the two axes are (0, 0)
    double border_x = 100;
    double border_y = 100;
    double end_x = 200;
    double end_y = 0;
    Arc2D(0 /* group_id */, border_x, border_y, end_x, end_y);
    //  Circle to (end_x, end_y) through (border_x , border_y)
    //  that is, circle to (200, 0) through (100, 100)
}
```

Requirement

Minimum supported version	iA Studio 0.24

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.

8.2.14 ArcCW2D

Purpose

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

Syntax

```
int ArcCW2D(
    int    group_id,
    double center_x,
    double center_y,
    double end_x,
    double end_y,
);
```

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.

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 1.3

8.2.15 ArcCCW2D

Purpose

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

Syntax

```
int ArcCCW2D(
    int    group_id,
    double center_x,
    double center_y,
    double end_x,
    double end_y,
);
```

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.

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 1.3

8.2.16 ArcAngle2D

Purpose

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

Syntax

```
int ArcAngle2D(
    int    group_id,
    double center_x,
    double center_y,
    double angle
);
```

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.
angle [in]	The angle that start point and end point relative to the absolute center position. It determines the direction and the rotation angle of circular movement. Parameter unit: deg

Return value

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

Remark

Parameter “angle” represents the direction of circular trajectory’s rotation.

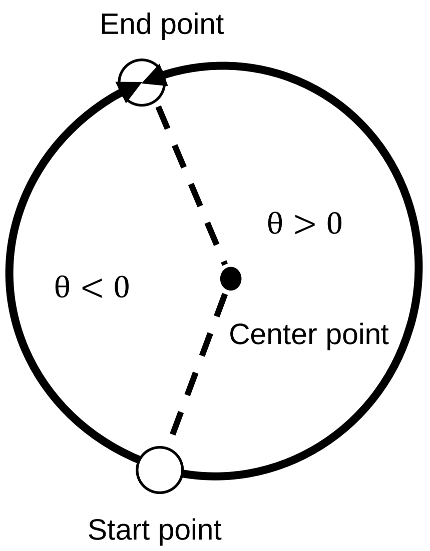
If “angle” > 0, move counterclockwise; if “angle” < 0, move clockwise.

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 = 10;
    double center_y = 10;
    double angle = 45;
    ArcAngle2D(0 /* group_id */, center_x, center_y, angle);
    //  Take (center_x, center_y) as the center and rotate 45 degrees
    //  that is, take (10, 10) as the center,
    //  and circle to (10, 10-10√2) from (0, 0)
}
```

Requirement

Minimum supported version	iA Studio 2.0



The value of θ determines the direction of circular movement.

θ > 0: move counterclockwise

θ < 0: move clockwise

8.2.17 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.

Remark

(1) Parameter “turns” represents the direction of circular trajectory’s rotation. If “turns” >= 0, move counterclockwise; if “turns” < 0, move clockwise.

(2) When ||turns|| <= 1, the total moving angle of circular trajectory is < 360°. If the total moving angle of circular trajectory is >= 360° (that is, one turn, or more than one turn), ||turns|| must be >= 2.

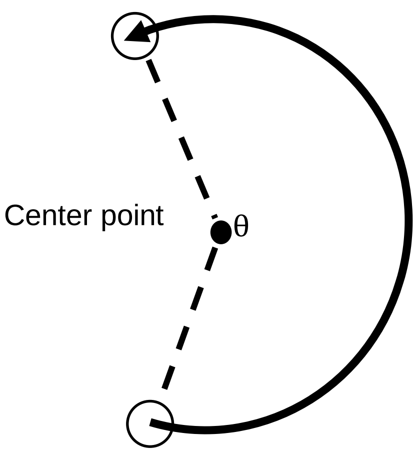
(3) The behavior of “turns = 0” and that of “turns =1” are the same when using this function.

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 = 100;
    double center_y = 0;
    double end_x = 200;
    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 (100, 0) as the center and circle to (200, 0)
}
```

Requirement

Minimum supported version	iA Studio 1.1



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	$210 + 1 \times 360°$	$570°$

8.3 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

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

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

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

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

8.3.6 SetGrpMotionProfile

Purpose

To set TCP linear motion parameters for an axis group.

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 maximum linear profile velocity for an axis group. Parameter unit: mm/s Input range: 0 ~ 5000
max_acceleration [in]	The maximum linear profile acceleration for an axis group. Parameter unit: mm/s^2 Input range: >0 ~ 50000 (acceleration cannot be 0)
max_deceleration [in]	The maximum linear profile deceleration for an axis group. Parameter unit: mm/s^2 Input range: >0 ~ 50000 (deceleration cannot be 0)
smooth_time [in]	The linear profile smooth time for an axis group. Parameter unit: ms Input range: 0 ~ 500

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 linear group motion profile is **[100, 500, 500, 50]** for velocity, acceleration, deceleration, and smooth time respectively.

Requirement

Minimum supported version	iA Studio 0.24

8.3.7 SetGrpAngMotionProfile

Purpose

To set TCP angular motion parameters for an axis group.

Syntax

```
int SetGrpAngMotionProfile(
    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 maximum angular profile velocity for an axis group. Parameter unit: deg/s Input range: 0 ~ 7200
max_acceleration [in]	The maximum angular profile acceleration for an axis group. Parameter unit: deg/s^2 Input range: >0 ~ 72000 (acceleration cannot be 0)
max_deceleration [in]	The maximum angular profile deceleration for an axis group. Parameter unit: deg/s^2 Input range: >0 ~ 72000 (deceleration cannot be 0)
smooth_time [in]	The angular profile smooth time for an axis group. Parameter unit: ms Input range: 0 ~ 500

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 angular group motion profile is **[360, 1800, 1800, 50]** for velocity, acceleration, deceleration, and smooth time respectively.

Requirement

Minimum supported version	iA Studio 2.0

8.3.8 GetGrpKin

Purpose

To get the kinematics type of an axis group.

Syntax

```
int GetGrpKin(
    int group_id
);
```

Parameter

group_id [in]   Axis group index.

Return value

The kinematics type of the axis group. **Refer to section 8.1.3 for details.**

Requirement

Minimum supported version	iA Studio 1.3

8.3.9 SetGrpKin

Purpose

To set the kinematics type of an axis group.

Syntax

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

Parameter

group_id [in]	Axis group index.
kin_type [in]	The new kinematics type of an axis group. **Refer to section 8.1.3 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

8.3.10 GetGrpMaxVel

Purpose

To get the maximum profile velocity of an axis group.

Syntax

```
double GetGrpMaxVel(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

The maximum profile velocity of the axis group.

Unit: mm/s or deg/s

Requirement

Minimum supported version	iA Studio 1.3

8.3.11 SetGrpVel

Purpose

To set the maximum profile velocity of an axis group.

Syntax

```
int SetGrpVel(
    int    group_id,
    double vel
);
```

Parameter

group_id [in]	Axis group index.
vel [in]	The new maximum profile velocity of an axis group. Parameter unit: mm/s or deg/s Input range: 0 ~ 5000

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 1.3

8.3.12 GetGrpMaxAcc

Purpose

To get the maximum profile acceleration of an axis group.

Syntax

```
double GetGrpMaxAcc(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

The maximum profile acceleration of the axis group.

Unit: mm/s^2 or deg/s^2

Requirement

Minimum supported version	iA Studio 1.3

8.3.13 SetGrpAcc

Purpose

To set the maximum profile acceleration of an axis group.

Syntax

```
int SetGrpAcc(
    int    group_id,
    double acc
);
```

Parameter

group_id [in]	Axis group index.
acc [in]	The new maximum profile acceleration of an axis group. Parameter unit: mm/s^2 or deg/s^2 Input range: >0 ~ 50000 (acceleration cannot be 0)

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 1.3

8.3.14 SetGrpAccTime

Purpose

To set the acceleration time of an axis group.

Syntax

```
int SetGrpAccTime(
    int    group_id,
    double acc_time
);
```

Parameter

group_id [in]	Axis group index.
acc_time [in]	The acceleration time of an axis group. Parameter unit: ms 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 1.3

8.3.15 GetGrpMaxDec

Purpose

To get the maximum profile deceleration of an axis group.

Syntax

```
double GetGrpMaxDec(
    int group_id
);
```

Parameter

group_id [in]   Axis group index.

Return value

The maximum profile deceleration of the axis group.

Unit: mm/s^2 or deg/s^2

Requirement

Minimum supported version	iA Studio 1.3

8.3.16 SetGrpDec

Purpose

To set the maximum profile deceleration of an axis group.

Syntax

```
int SetGrpDec(
    int    group_id,
    double dec
);
```

Parameter

group_id [in]	Axis group index.
dec [in]	The new maximum profile deceleration of an axis group. Parameter unit: mm/s^2 or deg/s^2 Input range: >0 ~ 50000 (deceleration cannot be 0)

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 1.3

8.3.17 SetGrpDecTime

Purpose

To set the deceleration time of an axis group.

Syntax

```
int SetGrpDecTime(
    int    group_id,
    double dec_time
);
```

Parameter

group_id [in]	Axis group index.
dec_time [in]	The deceleration time of an axis group. Parameter unit: ms 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 1.3

8.3.18 GetGrpSMTime

Purpose

To get the profile smooth time of an axis group.

Syntax

```
double GetGrpSMTime(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

The profile smooth time of the axis group.

Unit: ms

Requirement

Minimum supported version	iA Studio 1.3

8.3.19 SetGrpSMTime

Purpose

To set the profile smooth time of an axis group.

Syntax

```
int SetGrpSMTime(
    int    group_id,
    double smooth_time
);
```

Parameter

group_id [in]	Axis group index.
smooth_time [in]	The new profile smooth time of an axis group. Parameter unit: ms Input range: 0 ~ 500

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 1.3

8.3.20 GetGrpCoordSys

Purpose

To get the coordinate system of an axis group.

Syntax

```
double GetGrpCoordSys(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

The coordinate system of the axis group. **Refer to section 8.1.2 for details.**

Requirement

Minimum supported version	iA Studio 1.3

8.3.21 SetGrpCoordSys

Purpose

To set the coordinate system of an axis group.

Syntax

```
int SetGrpCoordSys(
    int group_id,
    int coord_sys
);
```

Parameter

group_id [in]	Axis group index.
coord_sys [in]	The new coordinate system of an axis group. **Refer to section 8.1.2 for details.** Example: 1. CS_MCS 2. CS_WCS1 | CS_PCS 3. CS_OFFSET | CS_WCS2 | CS_PCS

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 2.0

8.3.22 GetGrpBufferMode

Purpose

To get the buffer mode of an axis group.

Syntax

```
double GetGrpBufferMode(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

The buffer mode of the axis group. **Refer to section 8.1.4 for details.**

Requirement

Minimum supported version	iA Studio 1.3

8.3.23 SetGrpBufferMode

Purpose

To set the buffer mode of an axis group.

Syntax

```
int SetGrpBufferMode(
    int group_id,
    int buffer_mode
);
```

Parameter

group_id [in]	Axis group index.
buffer_mode [in]	The new buffer mode of an axis group. **Refer to section 8.1.4 for details.** Input range: 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 1.3

8.3.24 GetGrpTransMode

Purpose

To get the transition mode of an axis group.

Syntax

```
double GetGrpTransMode(
    int group_id
);
```

Parameter

group_id [in]   Axis group index.

Return value

The transition mode of the axis group. **Refer to section 8.1.5 for details.**

Requirement

Minimum supported version	iA Studio 1.3

8.3.25 SetGrpTransMode

Purpose

To set the transition mode of an axis group.

Syntax

```
int SetGrpTransMode(
    int group_id,
    int trans_mode
);
```

Parameter

group_id [in]	Axis group index.
trans_mode [in]	The new transition mode of an axis group. **Refer to section 8.1.5 for details.** Input range: 0 ~ 4

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 1.3

8.3.26 SetGrpTransPrm

Purpose

To set the transition mode's parameters of an axis group.

Syntax

```
int SetGrpTransPrm(
    int    group_id,
    double trans_vel,
    double trans_dis,
    double trans_dev,
    double trans_curv
);
```

Parameter

group_id [in]	Axis group index.
trans_vel [in]	The new transition mode's velocity parameter of an axis group. **Refer to section 8.1.5 for details.**
trans_dis [in]	The new transition mode's distance parameter of an axis group. **Refer to section 8.1.5 for details.**
trans_dev [in]	The new transition mode's maximum deviation parameter of an axis group. **Refer to section 8.1.5 for details.**
trans_curv [in]	The new transition mode's curvature parameter of an axis group. **Refer to section 8.1.5 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 1.3

8.3.27 GetGrpCmdNum

Purpose

To get the number of commands of an axis group in the command buffer.

Syntax

```
double GetGrpCmdNum(
    int group_id
);
```

Parameter

group_id [in] Axis group index.

Return value

The number of commands of an axis group in the command buffer.

Requirement

Minimum supported version	iA Studio 1.3

8.3.28 SetGrpVelScale

Purpose

To set the velocity scale of axis group motion.

Syntax

```
int SetGrpVelScale(
    int    group_id,
    double vel_scale
);
```

Parameter

group_id [in]	Axis group index.
vel_scale [in]	The new velocity scale of axis group motion. Input range: 0 ~ 100

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 1.4

8.3.29 GetGrpVelScale

Purpose

To get the velocity scale of axis group motion.

Syntax

```
double GetGrpVelScale(
    int group_id
);
```

Parameter

group_id [in]   Axis group index.

Return value

The velocity scale of axis group motion. Its range is from 0 to 100.

Requirement

Minimum supported version	iA Studio 1.4

8.3.30 GetGrpCoordTrans

Purpose

To get the transformation parameters of axis group's coordinate system.

Syntax

```
int GetGrpCoordTrans(
    int group_id,
    int coord_sys,
    double *trans_param
);
```

Parameter

group_id [in]	Axis group index.
coord_sys [in]	Coordinate system. **Refer to section 8.1.2 for details.**
trans_param [out]	A pointer to a six-element array which contains the transformation parameters in 6-DOF {X, Y, Z, A, B, C}. Parameter unit: mm for X, Y, Z; deg for A, B, C

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 2.0

8.3.31 SetGrpCoordTrans

Purpose

To set the transformation parameters of axis group's coordinate system.

Syntax

```
int SetGrpCoordTrans(
    int group_id,
    int coord_sys,
    double *trans_param
);
```

Parameter

group_id [in]	Axis group index.
coord_sys [in]	Coordinate system. **Refer to section 8.1.2 for details.**
trans_param [in]	A pointer to a six-element array which contains the transformation parameters in 6-DOF {X, Y, Z, A, B, C}. Parameter unit: mm for X, Y, Z; deg for A, B, C

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 2.0

8.3.32 GetGrpPoseCmd

Purpose

To get the pose command of axis group's coordinate system.

Syntax

```
int GetGrpPoseCmd(
    int group_id,
    int coord_sys,
    double *pose_cmd
);
```

Parameter

group_id [in]	Axis group index.
coord_sys [in]	Coordinate system. **Refer to section 8.1.2 for details.**
pose_cmd [out]	A pointer to a six-element array which contains the pose command in 6-DOF {X, Y, Z, A, B, C}. Parameter unit: mm for X, Y, Z; deg for A, B, C

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 2.0

8.3.33 GetGrpPoseFb

Purpose

To get the pose feedback of axis group's coordinate system.

Syntax

```
int GetGrpPoseFb(
    int group_id,
    int coord_sys,
    double *pose_fb
);
```

Parameter

group_id [in]	Axis group index.
coord_sys [in]	Coordinate system. **Refer to section 8.1.2 for details.**
pose_fb [out]	A pointer to a six-element array which contains the pose feedback in 6-DOF {X, Y, Z, A, B, C}. Parameter unit: mm for X, Y, Z; deg for A, B, C

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 2.0

8.4 Group status

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 at the “GrpEnabled” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23

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 at the “GrpMoving” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23

8.4.3 IsGrpInPos

Purpose

To query the “in-position” status of an axis group. If the axis group is in-position, all axes in the group are in-position.

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 at the “GrpInPos” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.24

8.4.4 IsGrpErrorStop

Purpose

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

Syntax

```
int IsGrpErrorStop(
    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 at the “GrpErrorStop” state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 1.3

8.5 Advanced group motion control

8.5.1 LineAbs

Purpose

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

Syntax

```
int LineAbs(
    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: mm for X, Y, Z; deg 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 8.1.2 for details.**
buff_mode [in]	Specify the path buffer mode. **Refer to section 8.1.4 for details.**
trans_mode [in]	Specify the path transition mode. **Refer to section 8.1.5 for details.**

trans_par [in]	Specify the pointer for specific transition mode. **Refer to section 8.1.5 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 3.0

8.5.2 LineRel

Purpose

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

Syntax

```
int LineRel(
    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: mm for X, Y, Z; deg 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 8.1.2 for details.**
buff_mode [in]	Specify the path buffer mode. **Refer to section 8.1.4 for details.**
trans_mode [in]	Specify the path transition mode. **Refer to section 8.1.5 for details.**
trans_par [in]	Specify the pointer for specific transition mode. **Refer to section 8.1.5 for details.** 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 3.0

8.5.3 CircleAbs

Purpose

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

Syntax

```
int CircleAbs(
    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: mm

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: mm

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: mm for X, Y, Z; deg 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 8.1.2 for details.**
buff_mode [in]	Specify the path buffer mode. **Refer to section 8.1.4 for details.**
trans_mode [in]	Specify the path transition mode. **Refer to section 8.1.5 for details.**
trans_par [in]	Specify the pointer for specific transition mode. **Refer to section 8.1.5 for details.** 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 3.0

8.5.4 CircleRel

Purpose

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

Syntax

```
int CircleRel(
    int group_id,
    double *center_pos,
    double *normal_vector,
    int turns,
    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.

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: mm

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: mm

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

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: mm for X, Y, Z; deg 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 8.1.2 for details.**
buff_mode [in]	Specify the path buffer mode. **Refer to section 8.1.4 for details.**
trans_mode [in]	Specify the path transition mode. **Refer to section 8.1.5 for details.**
trans_par [in]	Specify the pointer for specific transition mode. **Refer to section 8.1.5 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 3.0

9. GPIO functions

9.1 Overview

HIMC provides 8 sets of general purpose input / output (GPIO) pins; the hardware delay time is within 1ms and the power is 24V. The slave device can connect to the controller via CoE communication and update its IO status. The number of IO depends on the slave device. With the functions provided in this chapter, such as “SetGPO” and “SetSlvGPO”, users can set the signal of output pin respectively for HIMC and slave. Besides, users can query the signal status of input / output pin. With iA Studio’s function module “Digital IO” (refer to section 4.4 in “iA Studio User Guide”), users can observe and set HIMC’s and slave’s input / output status.

HIMC’s digital input “I8” is E-stop signal (refer to section 3.3 in “HIMC Installation Guide”), which will be triggered when receiving rising-edge signal. At this time, all axes will be disabled and all HMPL tasks will be stopped.

Note: After rising-edge signal is triggered, users can re-enable the axes or re-execute HMPL tasks.

9.1.1 Controller GPIO variables

Users can select the desired controller’s general purpose input/output system variables via Scope Manager in iA Studio (refer to section 4.8 in “iA Studio User Guide”). Detailed descriptions are shown in Table 9.1.1.1.

Table 9.1.1.1 Controller’s general purpose input / output variables

Name	Variable	Unit	Description
HIMC GPO	himc_gpo_bits	N/A	State of controller’s general purpose output pin.
HIMC GPI	himc_gpi_bits	N/A	State of controller’s general purpose input pin.

9.1.2 Example

Example 1

With the 4th general purpose input pin of controller, when the input rising-edge signal is detected, disable all axes and set the state of the 3rd general purpose output pin of controller as 1.

```
void main() {
    int last_state = 0;
    while (1) {
        int in = GetGPI(4);
        //*  Detect the signal of the 4th general purpose input pin of
    controller  */
        if ( (in ^ last_state) & in) {
            DisableAll();  //  Disable all axes
            SetGPO(3, 1);  /*  Set the state of the 3rd general purpose output pin
                               of controller as 1  */
        }
        last_state = in;   /*  Record the state of controller's general purpose
                               input  */
    }
}
```

Example 2

With the 3rd general purpose input pin of slave 1, when the input falling-edge signal is detected, toggle the state of 2nd general purpose output pin of slave 1 and set the state of all general purpose output pins of controller as 1.

```
void main() {
    int last_state = 0;
    while (1) {
        int in = GetSlvGPI(1, 3);
        /*  Detect the signal of the 3rd general purpose input pin of
    slave 1  */
        if ( (in ^ last_state) && !in) {
            ToggleSlvGPO(1, 2);  /*  Toggle the state of 2nd general purpose
                                     output pin of slave 1  */
            SetAllGPO(0xff);     /*  Set the state of all general purpose output
                                     pins of controller as 1  */
```

```
        }
        last_state = in;  //  Record the state of slave’s general purpose input
    }
}
```

Example 3

Set the 3rd general purpose output pin on slot 2 of slave 1 and toggle the state of slave’s output pin.

```
void main() {
    int last_state = 0;
    while (1) {
        SetSlvGPO(1 | HMPL_SLOT_2, 3);
        //  Set the 3rd general purpose output pin on slot 2 of slave 1
        Sleep(1000);
        //  Wait for 1 second
        ToggleSlvGPO(1 | HMPL_SLOT_2, 3);
        //  Toggle the 3rd general purpose output pin on slot 2 of slave 1
    }
}
```

9.2 Controller IO setting

9.2.1 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. "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

9.2.2 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

9.2.3 SetAllGPO

Purpose

To toggle the state of the controller's all general purpose outputs.

Syntax

```
int SetAllGPO(
    int all_gpo_state
);
```

Parameter

all_gpo_state [in]   State value of all general purpose outputs.

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 1.3

9.2.4 SetGPIInvert

Purpose

To set the invert state of the controller’s general purpose input.

Syntax

```
int SetGPIInvert(
    int gpi_idx,
    int invert
);
```

Parameter

gpi_idx [in]	General purpose input index.
invert [in]	The invert state to be set. “1” is for invert, and “0” is for not invert.

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 2.0

9.2.5 SetGPOInvert

Purpose

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

Syntax

```
int SetGPOInvert(
    int gpo_idx,
    int invert
);
```

Parameter

gpo_idx [in]	General purpose output index.
invert [in]	The invert state to be set. "1" is for invert, and "0" is for not invert.

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 2.0

9.2.6 BindEMO

Purpose

To set the general purpose input pin to be bound to E-Stop.

Syntax

```
int BindEMO(
    int gpi_idx
);
```

Parameter

gpi_idx [in]	General purpose input index. The default value is 8. If it is set as 0, all general purpose input pins are not bound to E-Stop.

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 2.0

9.3 Slave IO setting

9.3.1 SetSlvGPO

Purpose

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

Syntax

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

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored
gpo_idx [in]	General purpose output index.
on_off [in]	The state to be set. “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.

Remark

Users must configure Digital output object as PDO when using this function. For example, to set 0x60FE(Digital outputs) of the servo drive as PDO.

Requirement

Minimum supported version	iA Studio 1.1

9.3.2 ToggleSlvGPO

Purpose

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

Syntax

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

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
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.

Remark

Users must configure Digital output object as PDO when using this function. For example, to set 0x60FE(Digital outputs) of the servo drive as PDO.

Requirement

Minimum supported version	iA Studio 1.1

9.3.3 SetSlvAllGPO

Purpose

To toggle the state of the slave's all general purpose outputs.

Syntax

```
int SetSlvAllGPO(
    int slv_slot_id,
    int all_gpo_state
);
```

Parameter

slv_slot_id [in] Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.

all_gpo_state [in] State value of all general purpose outputs.

Return value

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

Remark

Users must configure Digital output object as PDO when using this function. For example, to set 0x60FE(Digital outputs) of the servo drive as PDO.

Requirement

Minimum supported version	iA Studio 2.0

9.4 Controller IO status

9.4.1 GetGPI

Purpose

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

Syntax

```
int GetGPI(
    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 "GPI_On" state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 3.0

9.4.2 GetGPO

Purpose

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

Syntax

```
int GetGPO(
    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 "GPO_On" state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 3.0

9.4.3 GetAllGPI

Purpose

To get the state of the controller's all general purpose inputs.

Syntax

```
int GetAllGPI();
```

Parameter

N/A

Return value

State value of the controller's all general purpose inputs.

If the 1st and the 4th GPI pin are TURE, it will return 9.

Requirement

Minimum supported version	iA Studio 1.3

9.4.4 GetAllGPO

Purpose

To get the state of the controller’s all general purpose outputs.

Syntax

```
int GetAllGPO();
```

Parameter

N/A

Return value

State value of the controller’s all general purpose outputs.

If the 2nd and the 3rd GPO pin are TURE, it will return 6.

Requirement

Minimum supported version	iA Studio 1.3

9.4.5 GetAllGPIInvertSt

Purpose

To get the invert state of the controller's all general purpose inputs.

Syntax

```
int GetAllGPIInvertSt();
```

Parameter

N/A

Return value

Invert state value of the controller's all general purpose inputs.

If the 1st and the 4th GPI invert pin are TURE, it will return 9.

Requirement

Minimum supported version	iA Studio 2.0

9.4.6 GetAllGPOInvertSt

Purpose

To get the invert state of the controller's all general purpose outputs.

Syntax

```
int GetAllGPOInvertSt();
```

Parameter

N/A

Return value

Invert state value of the controller's all general purpose outputs.

If the 2nd and the 3rd GPO invert pin are TURE, it will return 6.

Requirement

Minimum supported version	iA Studio 2.0

9.5 Slave IO status

9.5.1 GetSlvGPI

Purpose

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

Syntax

```
int GetSlvGPI(
    int slv_slot_id,
    int gpi_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
gpi_idx [in]	General purpose input index.

Return value

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

Remark

Users must configure Digital input object as PDO when using this function. For example, to set 0x60FE(Digital inputs) of the servo drive as PDO.

Requirement

Minimum supported version	iA Studio 3.0

9.5.2 GetSlvGPO

Purpose

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

Syntax

```
int GetSlvGPO(
    int slv_slot_id,
    int gpo_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
gpo_idx [in]	General purpose output index.

Return value

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

Remark

Users must configure Digital output object as PDO when using this function. For example, to set 0x60FE(Digital outputs) of the servo drive as PDO.

Requirement

Minimum supported version	iA Studio 3.0

(This page is intentionally left blank.)

10. AIO functions

10.1 Overview

With AIO functions, slaves with analog input (AI) or analog output (AO) function can read and set the related parameters. Among them, HMPL provides users with the setting of specifying conversion type between digital and analog. Detailed specifications are shown in Table 10.1.

Table 10.1.1. Definition of analog data conversion type

Conversion type	Conversion description	Specification	Remark
HMPL_DAC_NONE	N/A	Use only Hex function to operate this conversion.	-
HMPL_DAC_N10_P10	-10~10	Convert analog Hex value to -10~10.	Common in analog voltage module
HMPL_DAC_P0_P10	0~10	Convert analog Hex value to 0~10.	
HMPL_DAC_N5_P5	-5~5	Convert analog Hex value to -5~10.	
HMPL_DAC_P0_P5	0~5	Convert analog Hex value to -0~10.	
HMPL_DAC_P0_P20	0~20	Convert analog Hex value to 0~20.	Common in analog Current module
HMPL_DAC_P4_P20	4~20	Convert analog Hex value to 4~20.	
HMPL_DAC_N20_P20	-20~20	Convert analog Hex value to -20~20.	
HMPL_DAC_SIGNED	High byte defines whether the sign is positive or negative.	0 represents positive number in high byte; 1 represents negative number in high byte.	-

10.1.1 Example

Example 1

To set and read the analog output Hex value .

```
void main() {
    int slv_id = 0;
    int ao_index = 1;          //  analog output channel
    int ao_hex_val = 0xFFFF;  //  analog Hex output：0xFFFF = 65535
    SetSlvAOHex(slv_id, ao_index, ao_hex_val );
```

```
    Print("%l", GetSlvAOHex(slv_id, ao_index));
}
```

Example 2

To read analog input Hex value.

```
void main() {
    int slv_id = 1;
    int ai_index = 1;        //  analog input channel
    Print("%l", GetSlvAIHex(slv_id, ai_index));
}
```

Example 3

To set analog input/output conversion type.

```
void main() {
    int slv_id_ao = 1;
    int ao_index = 1;        //  analog output channel
    int slv_id_ai = 2;
    int ai_index = 1;        //  analog input channel

    SetSlvAOType(slv_id_ao, ao_index, HMPL_DAC_N10_P10 );
    //  Set analog output conversion to -10~10.
    SetSlvAO(slv_id_ao, ao_index, 10)
    //  Set analog output to 10
    SetSlvAIType(slv_id_ai, ai_index, HMPL_DAC_N10_P10 );
    //  Set analog input conversion to -10~10.
    Print("%f", GetSlvAI(slv_id, ai_index));
}
```

10.2 Slave AIO setting

10.2.1 SetSlvAIType

Purpose

To set the conversion type of the slave's analog input value.

Syntax

```
int SetSlvAIType(
    int slv_slot_id,
    int ai_idx,
    int range_type
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
Ai idx [in]	Analog input channel.
range_type [in]	Analog data conversion. **Refer to Table 10.1 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 3.0

10.2.2 SetSlvAOType

Purpose

To set the conversion type of the slave's analog output value.

Syntax

```
int SetSlvAOType(
    int slv_slot_id,
    int ao_idx,
    int range_type
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
Ai idx [in]	Analog output channel.
range_type [in]	Analog data conversion. **Refer to Table 10.1 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 3.0

10.2.3 SetSlvAOHex

Purpose

To set the analog output Hex value of the slave.

Syntax

```
int SetSlvAOHex(
    int slv_slot_id,
    int ao_idx,
    long long ao_hex_val
);
```

Parameter

slv_slot_id [in]	Slave ID and its Slot ID. Slot ID could be ignored if there is no slot on the slave.
ao idx [in]	Analog output channel.
ao_hex_val [in]	Analog 64bit output value.

Return value

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

Remark

Users must configure Analog output object as PDO when using this function.

Requirement

Minimum supported version	iA Studio 3.0

10.2.4 SetSlvAO

Purpose

To set the analog output value of the slave.

Syntax

```
int SetSlvAO(
    int slv_slot_id,
    int ao_idx,
    double ao_val
);
```

Parameter

slv_slot_id [in]	Slave ID and its Slot ID. Slot ID could be ignored if there is no slot on the slave.
ao_idx [in]	Analog output channel.
ao_val [in]	Analog output value.

Return value

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

Remark

Users must configure Analog output object as PDO when using this function.

Requirement

Minimum supported version	iA Studio 3.0

10.3 Slave AIO status

10.3.1 GetSlvAIType

Purpose

To get the analog input type of the slave.

Syntax

```
int GetSlvAIType(
    int slv_slot_id,
    int ai_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and its Slot ID. Slot ID could be ignored if there is no slot on the slave.
ai_idx [in]	Analog input channel.

Return value

The analog input type of the slave. **Refer to Table 10.1 for details.**

Requirement

Minimum supported version	iA Studio 3.0

10.3.2 GetSlvAOType

Purpose

To get the analog output type of the slave.

Syntax

```
int GetSlvAOType(
    int slv_slot_id,
    int ao_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
ao_idx [in]	Analog output channel.

Return value

The analog output type of the slave. **Refer to Table 10.1 for details.**

Requirement

Minimum supported version	iA Studio 3.0

10.3.3 GetSlvAIHex

Purpose

To get the analog input Hex value of the slave.

Syntax

```
int GetSlvAIHex(
    int slv_slot_id,
    int ai_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
ao_idx [in]	Analog input channel.

Return value

The analog input Hex value.

Remark

Users must configure Analog input object as PDO when using this function.

Requirement

Minimum supported version	iA Studio 3.0

10.3.4 GetSlvAI

Purpose

To get the analog input of the slave.

Syntax

```
double GetSlvAI(
    int slv_slot_id,
    int ai_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
ai_idx [in]	Analog input channel.

Return value

The analog input value.

Remark

Users must configure Analog input object as PDO when using this function.

Requirement

Minimum supported version	iA Studio 2.0

10.3.5 GetSlvAOHex

Purpose

To get the analog output Hex value of the slave.

Syntax

```
int GetSlvAOHex(
    int slv_slot_id,
    int ao_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
ao_idx [in]	Analog output channel.

Return value

The analog output Hex value.

Remark

Users must configure Analog output object as PDO when using this function.

Requirement

Minimum supported version	iA Studio 3.0

10.3.6 GetSlvAO

Purpose

To get the analog output value of the slave.

Syntax

```
double GetSlvAO(
    int slv_slot_id,
    int ao_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
ao_idx [in]	Analog output channel.

Return value

The analog output value.

Remark

Users must configure Analog output object as PDO when using this function.

Requirement

Minimum supported version	iA Studio 2.0

10.4 Slave AO bound to HIMC internal buffer variable

10.4.1 SetSlvAOMonitor

Purpose

To set the controller variable to be bound to analog output.

Syntax

```
int SetSlvAOMonitor(
    int slv_slot_id,
    int ao_idx,
    int var_id
);
```

Parameter

slv_slot_id [in] Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.

ao_idx [in] Analog output channel.

var_id [in] Controller variable and axis ID. The following tables show the definition of controller variable and axis ID.

Refer to section 17.1.2 for more controller variables.

Example: HMPL_AXIS_0 | HMPL_REF_VEL

Controller variable	Definition	Controller variable	Definition
HMPL_REF_POS	Reference position of axis	HMPL_POS_FB	Position feedback of axis
HMPL_REF_VEL	Reference velocity of axis	HMPL_VEL_FB	Velocity feedback of axis
HMPL_REF_ACC	Reference acceleration of axis	HMPL_ACC_FB	Acceleration feedback of axis
		HMPL_CUR_FB	Current feedback of axis

Axis ID	Definition
HMPL_AXIS_0	Axis 0
HMPL_AXIS_1	Axis 1
…	…
HMPL_AXIS_15	Axis 15

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 2.0

10.4.2 SetSlvAOParam

Purpose

To set the slave's analog output bound to controller variable.

Syntax

```
int SetSlvAOParam(
    int slv_slot_id,
    int ao_idx,
    int ao_en_bind,
    double ao_scale,
    double ao_offset
);
```

Parameter

slv_slot_id [in] Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.

ao_idx [in] Analog output channel.

ao_en_bind [in] 0: The function of analog output bound to controller variable is OFF (default)
1: The function of analog output bound to controller variable is ON

ao_scale [in] Scale of analog output and controller variable.

ao_offset [in] Offset of analog output and controller 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 2.0

10.4.3 GetSlvAOScale

Purpose

To get the slave's scale of analog output and controller variable.

Syntax

```
double SetSlvAOScale(
  int slave_id,
  int slot_idx,
  int ao_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
ao_idx [in]	Analog output channel.

Return value

The slave's scale of analog output and controller variable.

Requirement

Minimum supported version	iA Studio 2.0

10.4.4 GetSlvAOOffset

Purpose

To get the slave’s offset of analog output and controller variable.

Syntax

```
double GetSlvAOOffset(
    int slv_slot_id,
    int ao_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
ao_idx [in]	Analog output channel.

Return value

The slave’s offset of analog output and controller variable.

Requirement

Minimum supported version	iA Studio 2.0

10.4.5 IsSlvAOBound

Purpose

To query whether the slave's analog output is bound to controller variable.

Syntax

```
int IsSlvAOBound(
    int slv_slot_id,
    int ao_idx
);
```

Parameter

slv_slot_id [in]	Slave ID and Slot ID. If there is no slot on the slave, Slot ID can be ignored.
ao_idx [in]	Analog output channel.

Return value

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

Requirement

Minimum supported version	iA Studio 2.0

(This page is intentionally left blank.)

11. User Table functions

11.1 Overview

HIMC provides users with free memory space, which can store up to 512,000 double-type variable data (500K Bytes). With the functions provided in this chapter, users can access memory space. The written values will be stored to controller's random access memory (RAM). With function "SaveUserTable", User Table's data in memory space will be saved to HIMC's hard disk space. After HIMC is power cycled, with function "LoadUserTable", the stored data will be copied to User Table's memory space again.

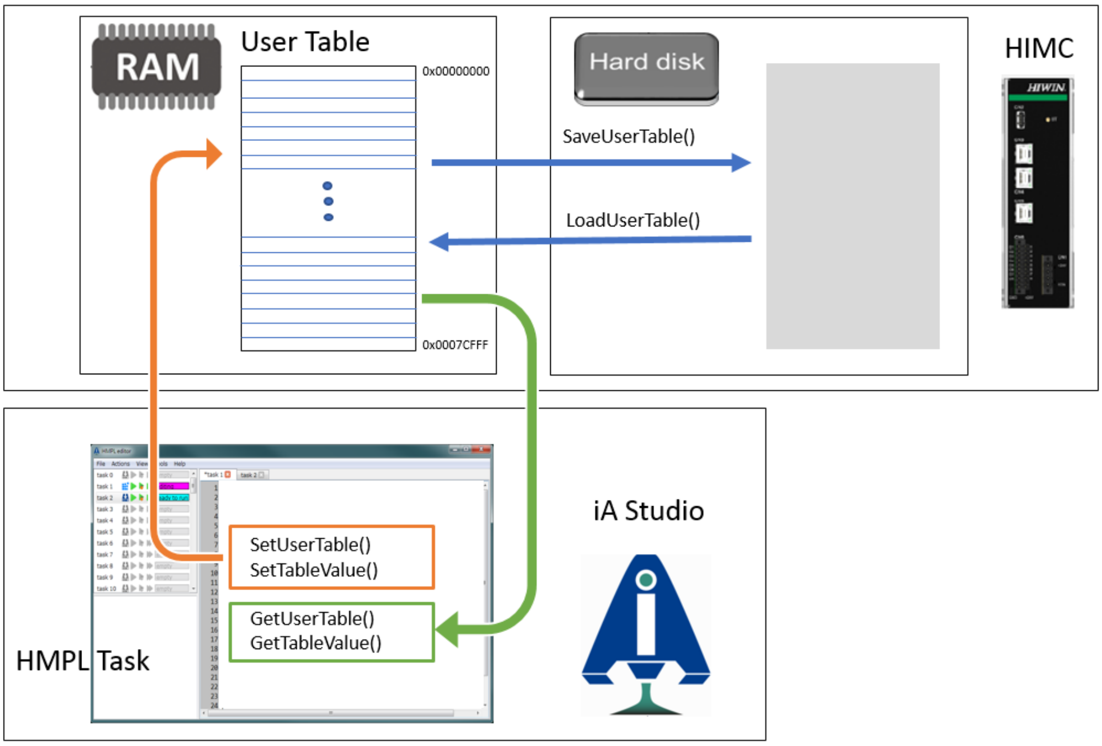


Figure 11.1.1

Note: Users can access User Table's variable values via Table Viewer in iA Studio (refer to section 4.11 in "iA Studio User Guide"), including loading and saving to HIMC's memory and hard disk.

Attention:
The error map used by dynamic error compensation functions is stored in User Table's memory space. When enabling dynamic error compensation, users should ensure the access to other User Table values will not affect the built error compensation values.

11.2 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

11.3 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

11.4 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

11.5 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

11.6 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 0.23

11.7 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 0.23

12. Position Trigger functions

12.1 Overview

HIMC position trigger function can only be used with HIWIN servo drive. With HMPL commands, users can operate PT (position trigger) related functions. Before operating PT related functions, please consult HIWIN or local distributors for compatible drives.

Note: The requirements for HIWIN servo drives to use PT related functions are (1) digital encoder (2) complete homing procedure first.

12.1.1 PT variables

PT related functions are operated based on the variables listed in Table 12.1.1.1.

Table 12.1.1.1

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	mm or deg	Start position of PT function. PT output signal train starts at this point.	SetPT_StartPos
end position	double	mm or deg	End position of PT function. No more PT output signal is sent out after this point.	SetPT_EndPos
interval	double	mm or deg	Position interval between consecutive PT outputs.	SetPT_Interval
pulse width	int	ns	The width of each PT output signal. 1. For E1 series servo drive, the range is from 20 ns to 80,000 ns, with the minimum increment of 20 ns. For example, 20, 40, ... 80,000 ns.	SetPT_PulseWidth

In Figure 12.1.1.1, the polarity is set to be “active high”.

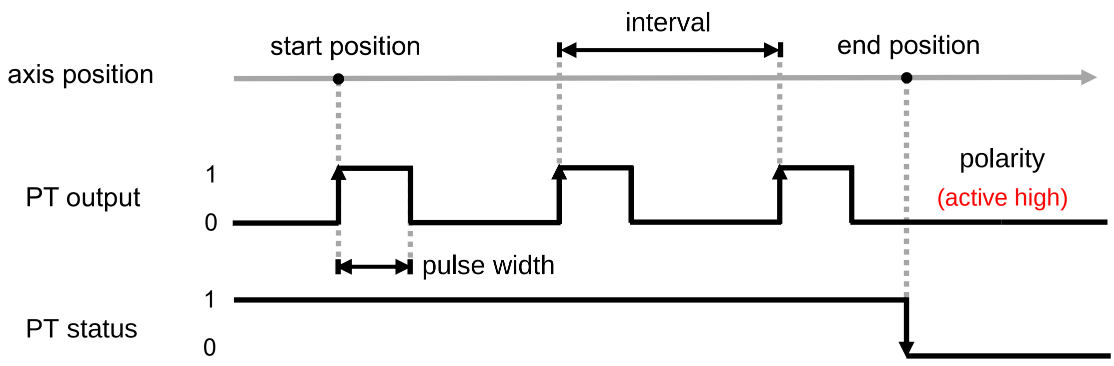


Figure 12.1.1.1

In Figure 12.1.1.2, the polarity is set to be “active low”.

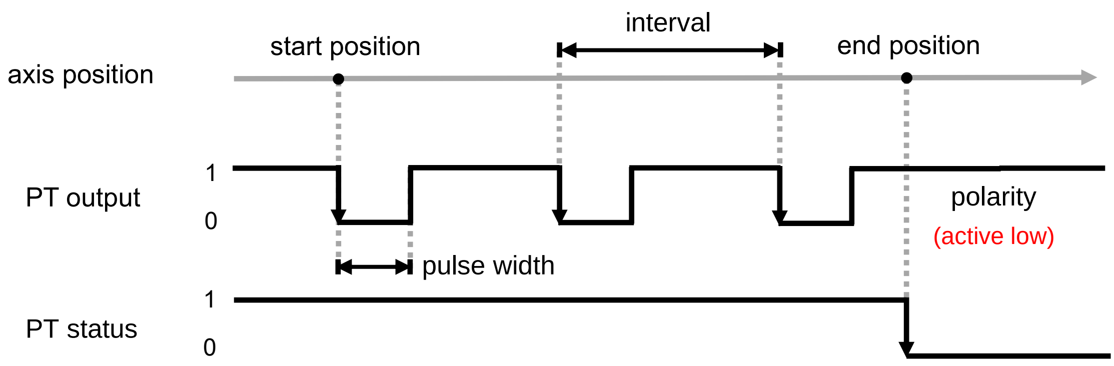


Figure 12.1.1.2

Limitation:

The interval of PT function and the velocity of axis must fit in the formula “Velocity < Interval x Position sampling rate”. If the interval is set as 100 um and the position sampling rate is 16 K, the velocity must be less than 1600 mm/s.

Note: To adjust PT function’s output polarity (active high / low), go to servo drive’s HMI for setting. After saving the setting, power cycle servo drive to make the output polarity be effective.

12.1.2 Flow of using PT function

◆ Fixed interval PT function

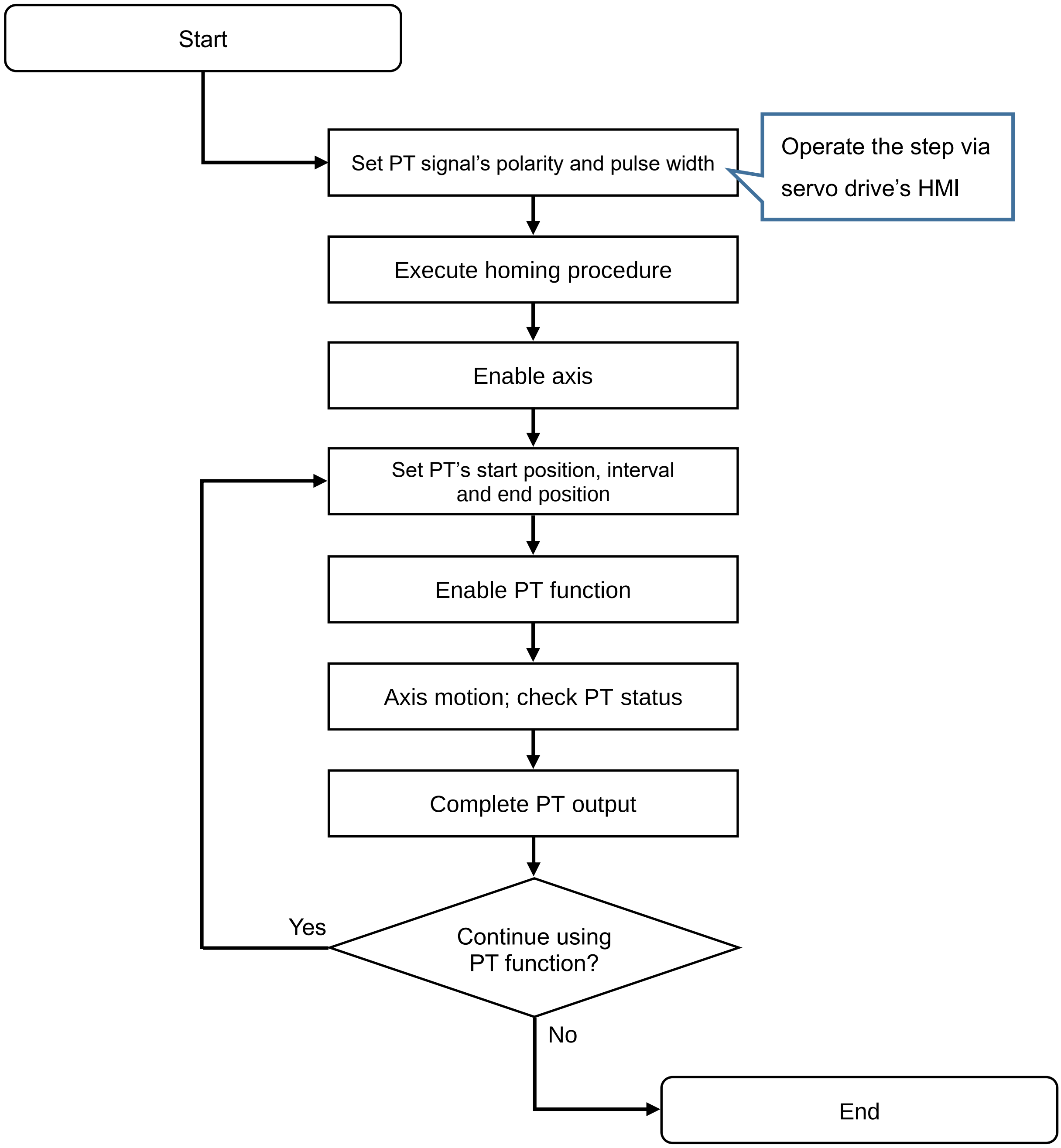


Figure 12.1.2.1

12.1.3 Example

Example: Fixed interval PT function

```
void main()
{
    int axis_id = 1;
    double start_pos = 10;    //  start position of PT function: 10 mm
    double end_pos = 20;      //  end position of PT function: 20 mm
    double interval = 1;      //  interval of PT function: 1 mm

    //  pulse width: 1000 ns
    SetPT_PulseWidth(axis_id, 1000);
    //  homing method 33 - homing with index signal from negative direction
    Home(HOME_METHOD_33, axis_id, 0, 20, 0, 10000);
    //  Enable axis
    Enable(axis_id);
    Till(IsEnabled(axis_id));
    //  Move to 0 mm
    MoveAbs(axis_id, 0);
    Till(IsInPos(axis_id));

    // ------ Move toward positive direction ------
    //  Set PT's start position, interval and end position
    SetPT_StartPos(axis_id, start_pos);
    SetPT_Interval(axis_id, interval);
    SetPT_EndPos(axis_id, end_pos);

    //  Enable PT function
    EnablePT(axis_id);

    //  Axis moves from  0 mm to 30 mm
    MoveAbs(axis_id, 30);
    Till(IsInPos(axis_id));

    // ------ Move toward negative direction ------
    //  Set PT's start position and end position
    SetPT_StartPos(axis_id, end_pos);
```

```
    SetPT_EndPos(axis_id, start_pos);

    //  Enable PT function
    EnablePT(axis_id);

    //  Axis moves from 30 mm to 0 mm
    MoveAbs(axis_id, 0.0);
    Till(IsInPos(axis_id));
}
```

12.2 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.

Remark

iA Studio 1.2 (included) and above support the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23

12.3 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.

Remark

iA Studio 1.2 (included) and above support the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23

12.4 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 “PTEnabled” state. Otherwise, it will return **FALSE** (0).

Remark

iA Studio 1.2 (included) and above support the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23

12.5 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: mm or deg

Return value

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

Remark

iA Studio 1.2 (included) and above support the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23

12.6 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: mm or deg

Return value

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

Remark

iA Studio 1.2 (included) and above support the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23

12.7 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: mm or deg

Return value

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

Remark

iA Studio 1.2 (included) and above support the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23

12.8 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: ns

Return value

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

Remark

iA Studio 1.2 (included) and above support the setting of E1 series servo drive.

Requirement

Minimum supported version	iA Studio 0.23

(This page is intentionally left blank.)

13. Touch Probe functions

13.1 Overview

Touch probe function is a latch function, used in homing procedure (applicable to AC motor, direct drive motor and linear motor). It captures encoder's position feedback value with the edge triggering of encoder input signal. As Figure 13.1.1 shows, when the servo drive passes by its encoder index signal, touch probe function will be triggered and the position of the index signal will be recorded. With touch probe function, users can query the controller whether touch probe function is triggered, and the controller can get the position of the index signal recorded by latch.

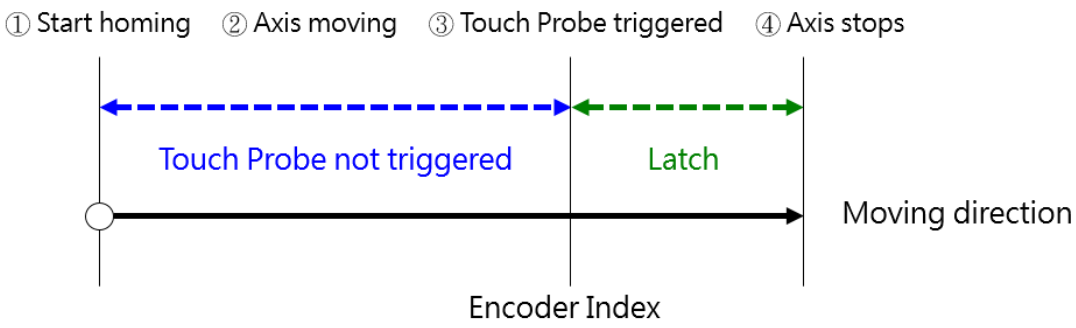


Figure 13.1.1

13.2 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.

Remark

Users must configure object 0x60B8(Touch probe function) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

13.3 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.

Remark

Users must configure object 0x60B8(Touch probe function) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

13.4 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 “TouchProbeEnabled” state. Otherwise, it will return **FALSE** (0).

Remark

Users must configure object 0x60B9(Touch probe status) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

13.5 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 "TouchProbeTriggered" state. Otherwise, it will return **FALSE** (0).

Remark

Users must configure object 0x60B9(Touch probe status) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 0.23

13.6 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: mm or deg

Return value

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

Remark

Users must configure the corresponding Touch probe object as PDO when using this function such as 0x60BA(Touch probe 1 positive edge).

Requirement

Minimum supported version	iA Studio 0.23

13.7 SetTouchProbeFunc

Purpose

To set touch probe function.

Syntax

```
int SetTouchProbeFunc(
    int    axis_id,
    int    tp_source,
    int    cont_tigger,
    int    detect_edge
);
```

Parameter

axis_id [in]	Axis index.
tp_source [in]	Touch probe source. Input range: 0 (touch probe1, default), 1 (touch probe 2).
cont_trigger [in]	Trigger mode. Input range: 0 (single trigger, default), 1 (continuous trigger)
detect_edge [in]	Edge detecting mode. Input range: 0 (rising-edge detection, default), 1 (falling-edge detection)

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 3.0

14. Dynamic Error Compensation functions

14.1 Overview

HIMC provides dynamic 1D / 2D / 3D error compensation function. Based on related error mearsurement and calculation results, users can establish an error map and do the setting on HIMC. The setting parameters include the compensated axis, reference axis, interval of map points, base of map points, number of map points and compensation value of each map point. As for the setting of compensation value, the memory space of HIMC User Table will be used and the first ID position of error map will be provided to User Table.

Note 1: Refer to chapter 11 for the usage and the description of User Table.

Note 2: Before enabling dynamic error compensation, axis must complete homing to fix the coordinate positions of compensated axis and reference axis.

Users can choose the same axis to be both compensated axis and reference axis, or choose multiple different axes to be the reference axes of the compensated axis. For example, the compensated axis is Z axis, and the reference axes are X axis and Y axis. The compensation value of the compensated axis will change according to the moving position of the reference axis. During axis motion, the established error map will calculate the compensation command value between map points with linear interpolation, so that the compensation value can keep continuous. When the position of the axis exceeds the range established by error map, HIMC will take the nearest map point's compensation value to be its compensation command, as Figure 14.1.1 shows.

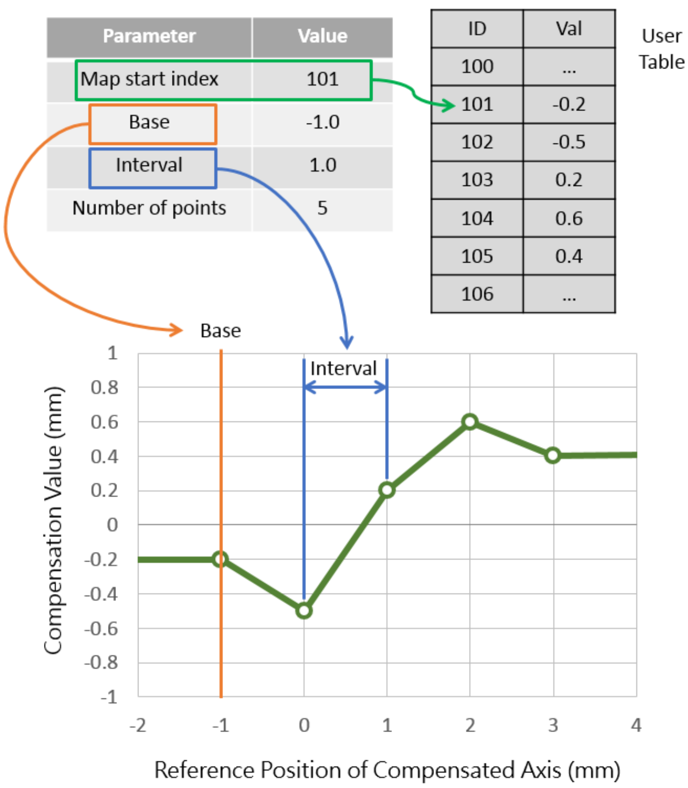


Figure 14.1.1

After enabling dynamic error compensation, in controller's axis control command, the output reference position will add the displacement to be compensated to eliminate the known measured error. As Figure 5.1.1 shows, the relationship can be described as:

Reference position + Position compensation = Position output

After enablilng dynamic error compensation, users can observe variables via Scope Manager in iA Studio.

- Compensation value: Axis → Motion Variable → Position Compensation
- Reference position (without compensation): Axis → Motion Variable → Reference Position
- Reference position (with compensation): Axis → Motion Variable → Position Output

Limit:
In HIMC, compensation command does not go through the profile generator, and the maximum compensation value preset by the controller is 1 mm. If the compensation value is larger than 1 mm, the system will display an error message to remind users.

When enabling dynamic error compensation, the reference coordinate to be compensated must be fixed. Therefore, users cannot change the home offset of the axis.

14.1.1 Example

Example 1: One-dimensional dynamic error compensation

In this example, axis 0 is the compensated axis, and axis 1 is the reference axis. The compensation value of the compensated axis will change according to the position of the reference axis. Their relationship is shown in Figure 14.1.1.1.

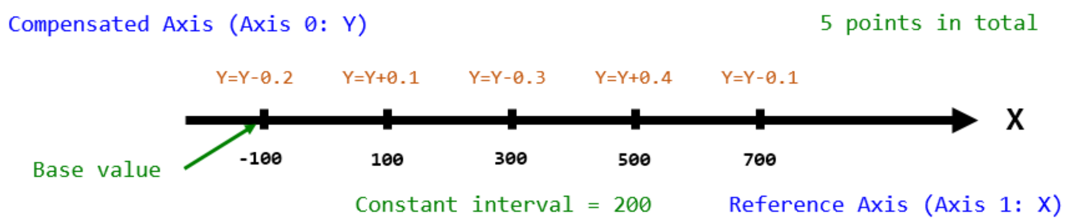


Figure 14.1.1.1

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] = {-0.2, 0.1, -0.3, 0.4, -0.1};
    SetUserTable(1, 5, data);

    SetupComp(
      0,    //  axis to be compensated
      1,    //  start index in user table
      -100, //  base value
      200,  //  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() {
    Disable(0);
    Till(!IsEnabled(0));
```

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

Example 2: Two-dimensional dynamic error compensation

This example is the application that the compensated axis and the reference axis are different, mostly used for Z axis on XYZ stage, since Z axis will have a height error of several microns in different XY positions. In this example, axis 2 is the compensated axis, and axis 0 and axis 1 are the reference axes. The compensation value of axis 2 will change according to the position of axis 0 and axis 1. Their relationship is shown in Figure 14.1.1.2.

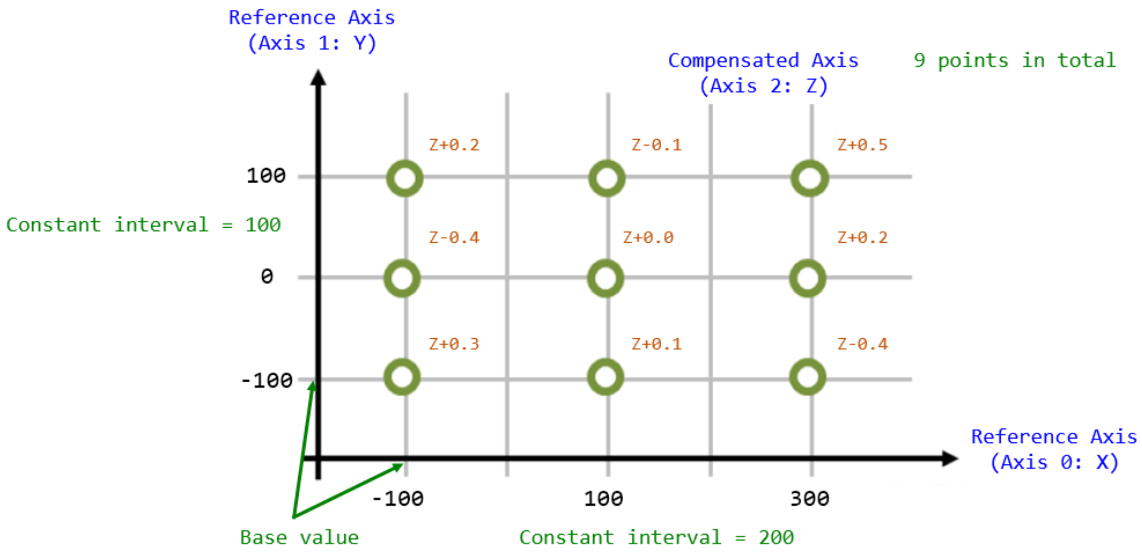


Figure 14.1.1.2

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] = {0.3, 0.1, -0.4, -0.4, 0.0, 0.2, 0.2, -0.1, 0.5};
    SetUserTable(3, 9, data);

    double base[2] = {-100, -100};
    double interval[2] = {200, 100};
    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() {
    Disable(2);
    Till(!IsEnabled(2));
    DisableComp(2);  //  Disable compensation on axis 2
}
```

Example 3: Three-dimensional dynamic error compensation

This example is a three-dimensional dynamic error compensation applied to precision XYZ stage. In this example, axis 2 is the compensated axis, and axis 0, axis 1 and axis 2 are the reference axes. The compensation value of axis 2 will change according to the position of axis 0, axis 1 and axis 2. Figure 14.1.1.3 describes the sequence of parameter input and its compensation value, and the detailed descriptions of user map tables are shown in Figure 14.1.1.4 to Figure 14.1.1.6.

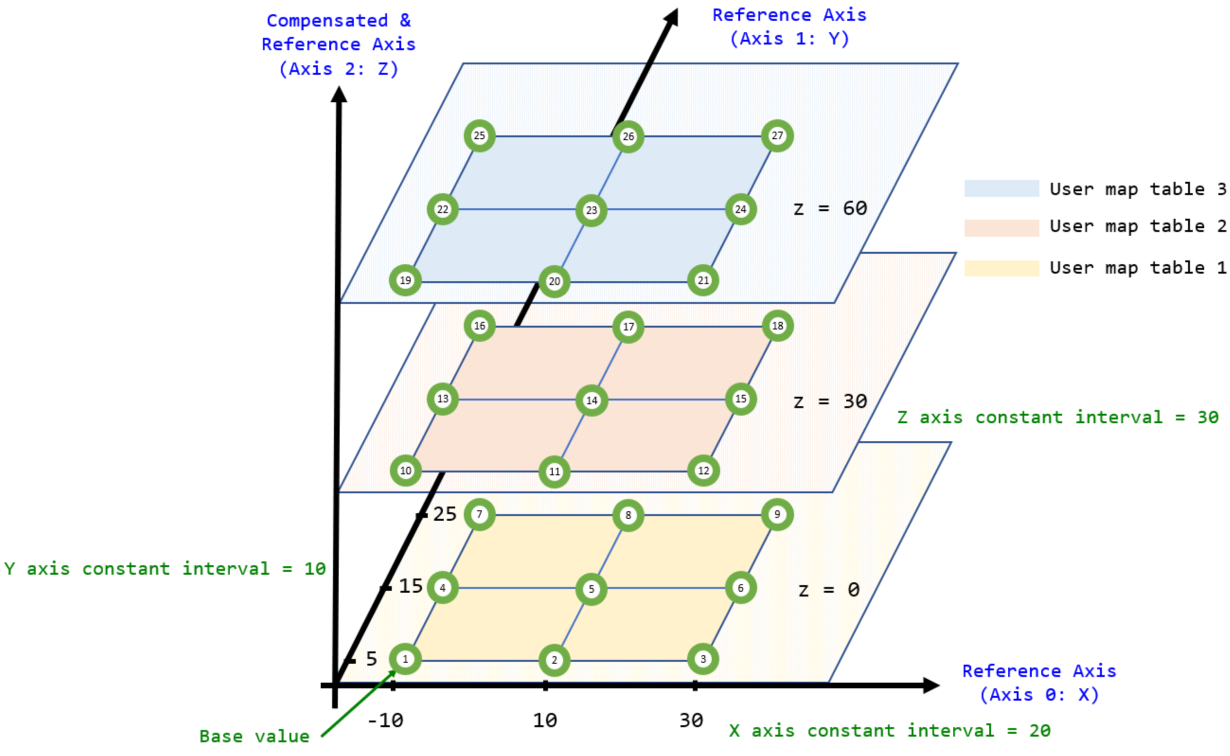


Figure 14.1.1.3

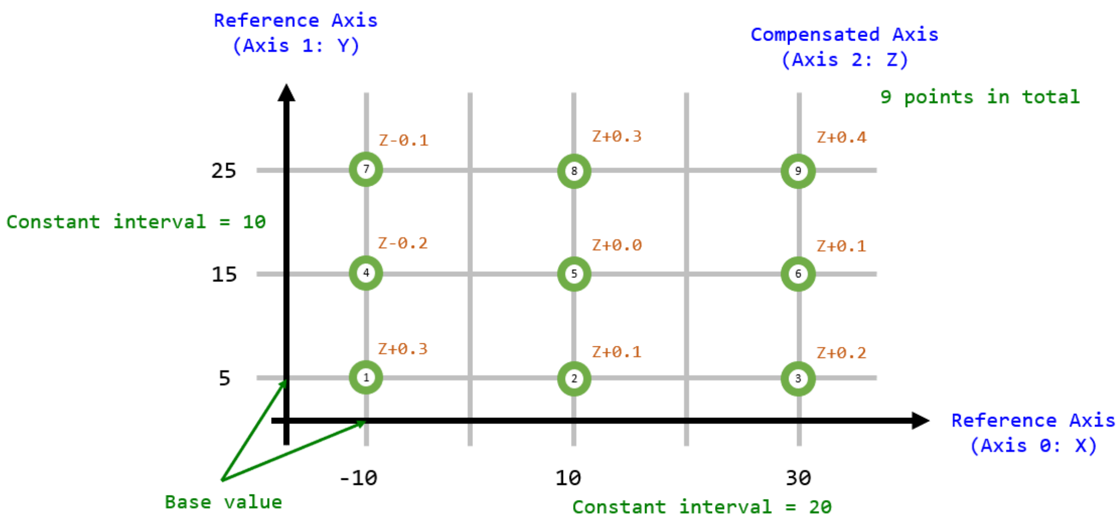


Figure 14.1.1.4 User map table 1

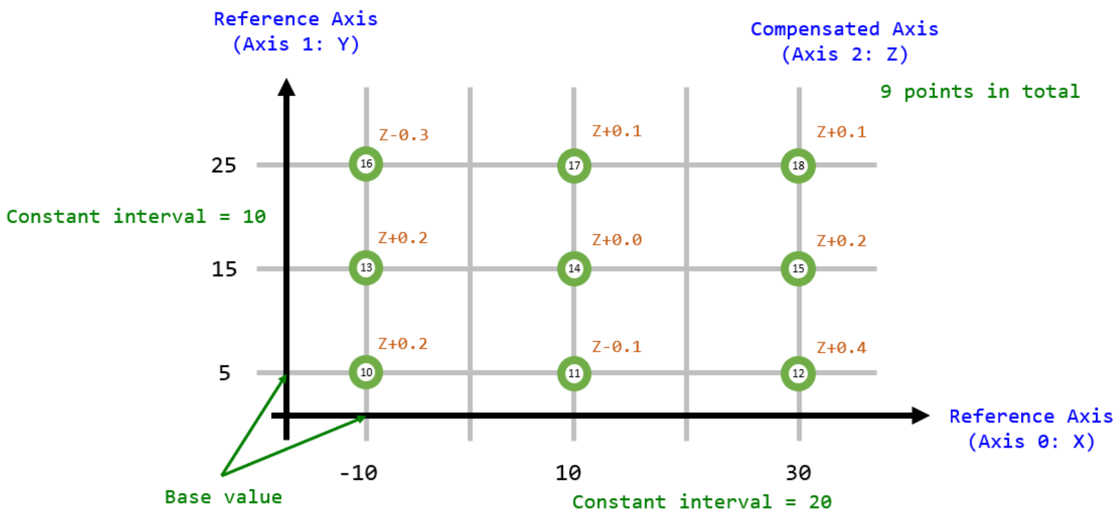


Figure 14.1.1.5 User map table 2

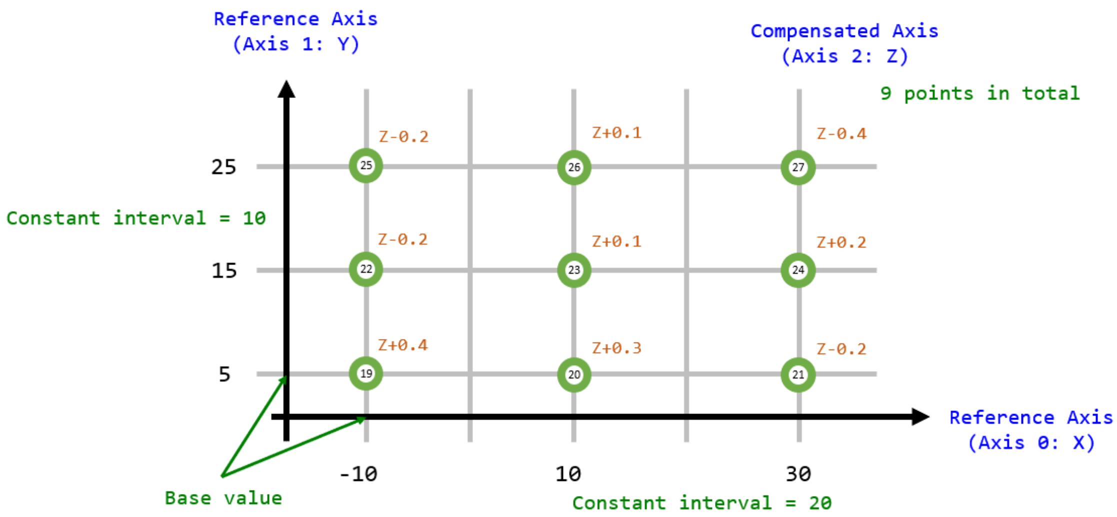


Figure 14.1.1.6 User map table 3

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, axis 1 and axis 2) to observe the compensation result.

```
void main() {
    //  Set up user map table
    double data[27] = {0.3, 0.1, 0.2, -0.2, 0.0, 0.1, -0.1, 0.3, 0.4,  //  1
                       0.2, -0.1, 0.4, 0.2, 0.0, 0.2, 0.3, 0.1, 0.1,  //  2
                       0.4, 0.3, -0.2, -0.2, 0.1, 0.2, -0.2, 0.1, -0.4};  //  3
    SetUserTable(3, 27, data);

    double base[3] = {-10, 5, 0};
```

```
    double interval[3] = {20, 10, 30};
    int num_pt[3] = {3, 3, 3};
    int ref_axis[3] = {0, 1, 2};

    SetupComp3D(
      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() {
    Disable(2);
    Till(!IsEnabled(2));
    DisableComp(2);  //  Disable compensation on axis 2
}
```

14.2 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

14.3 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.
(3) The setting of the dynamic error compensation will be cleared.
To enable the dynamic error compensation again, users need to reset the setting.

Requirement

Minimum supported version	iA Studio 0.23

14.4 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: mm or deg
interval [in]	Constant interval between adjacent map points. Parameter unit: mm or deg
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.

Requirement

Minimum supported version	iA Studio 0.23

14.5 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: mm or deg
interval [in]	A pointer to a two-element array which contains each dimension's constant interval between adjacent map points. Parameter unit: mm or deg
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.

Requirement

Minimum supported version	iA Studio 0.23

14.6 SetupComp3D

Purpose

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

Syntax

```
int SetupComp3D(
    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 three-element array which contains each dimension's base value (the smallest value of map input). Parameter unit: mm or deg
interval [in]	A pointer to a three-element array which contains each dimension's constant interval between adjacent map points. Parameter unit: mm or deg
num_pt [in]	A pointer to a three-element array which contains each dimension's number of map points.
ref_axis_id [in]	A pointer to a three-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.

Requirement

Minimum supported version	iA Studio 1.4

14.7 GetCompPos

Purpose

To get the error compensation value of an axis sent from the controller to the servo drive.

Syntax

```
double GetCompPos(
    int axis_id
);
```

Parameter

axis_id [in]   Axis index.

Return value

The error compensation value of the axis.

Unit: mm or deg

Requirement

Minimum supported version	iA Studio 1.3

14.8 SetCompAlgType

Purpose

To set the interpolation method of dynamic error compensation of an axis.

Syntax

```
int SetCompAlgType(
    int axis_id,
    int alg_type
);
```

Parameter

axis_id [in]	Axis index.
alg_type [in]	The interpolation method of dynamic error compensation. 0: first-order linear interpolation (defualt) 1: three-order spline interpolation

Return value

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

Remark

Three-dimensional dynamic error compensation does not support three-order spline interpolation.

Requirement

Minimum supported version	iA Studio 2.0

15. Filter functions

15.1 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.

15.1.1 Example

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

Example 1: Single axis

```
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
}
```

Example 2: Axis group

```
void main()
{
    int gid = 0;          //  Set gid as group id 0
    int axis1 = 0;        //  Set axis1 as axis 0
    int axis2 = 1;        //  Set axis2 as axis 1

    Enable(axis1);        //  Enable axis 0
    Enable(axis2);        //  Enable axis 1
    Till(IsEnabled(axis1) && IsEnabled(axis2));

    AddAxisToGrp(gid, axis1);        //  Add axis1 to axis group gid
    AddAxisToGrp(gid, axis2);        //  Add axis2 to axis group gid
    EnableGroup(gid);                //  Enable axis group gid

    //  Set InShape filter's parameters of axis group gid
    SetGrpInShape(gid, 10, 0.15, 1);
    //  Enable InShape filter of axis group gid
    EnableGrpInShape(gid);
}
```

15.2 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

15.3 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

15.4 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

15.5 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

15.6 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

15.7 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

15.8 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

15.9 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

15.10 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

16. HMPL Task functions

16.1 Overview

HIMC has 64 built-in HMPL tasks for users to practice motion profile commands based on application. In any HMPL task, users can start or stop other HMPL tasks with HMPL task function. When a HMPL task is being executed, users cannot ask it to be re-executed; instead, users should wait until the execution of the task is done and the task enters “stop” status. However, users can query whether the HMPL task is currently being executed, and control the order of multiple HMPL tasks for the application accordingly.

16.2 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

16.3 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

16.4 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

16.5 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

16.6 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 HMPL task is at the "TaskStop" state. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 0.23

(This page is intentionally left blank.)

17. Variable and Function Operation functions

17.1 Overview

HIMC provides users with variable operation functions for servo drive and controller. For servo drive's variable operation, data exchange can be performed according to Object Dictionary via CoE communication. As for controller's variable operation, specific variable's addressing ID must be given based on controller's variable ID list for the access. The definitions of parameters of the controller are given in section 17.1.1.

Attention:
If there is no requirement for specific purposes, it is recommended for users to access relevant system variables with relevant HMI and functions. When using variable operation functions, users should ensure the safety of accessing variables and entering values.

17.1.1 Controller variables list

HIMC takes 32 bits as controller variable's addressing ID. Its type is 0x□□□□□□□□, where "0x" indicates the value is in hexadecimal system. With variable operation functions, users can access system variables, axis variables and axis group variables provided by HIMC. The rule of addressing ID is explained as follows:

1. The 1st and 2nd value of addressing ID indicates "category of controller variable". 0x00□□□□□□ belongs to system variable; 0x83□□□□□□ belongs to axis variable; 0x82□□□□□□ belongs to axis group variable.

2. The 3rd and 4th value of addressing ID indicates "axis ID or axis group ID". For example, axis variable 0x8302□□□□ is a variable storing axis index 02, while axis group variable 0x8201□□□□ is a variable storing axis group index 01.

3. The 5th to 8th value of addressing ID indicates "addressing location of controller's system, axis or axis group variable". Refer to Table 17.1.1.1 to Table 17.1.1.3 for parameters list and description.

Table 17.1.1.1

System Variables		
Addressing ID	Variable Name	Description
0x0000012c	HCV_ID_fclk	System execution clock (increase 1 count per 250 us)
0x0000012e	HCV_ID_timeInMs	System execution time (ms)
0x000007d0	HCV_ID_user_table	double[512000] array variable that users can freely use
0x00002328	HCV_ID_ltest0	int variable that users can freely use
0x00002329	HCV_ID_ltest1	int variable that users can freely use
0x0000232a	HCV_ID_ltest2	int variable that users can freely use
0x0000232b	HCV_ID_ltest3	int variable that users can freely use
0x0000232c	HCV_ID_ltest4	int variable that users can freely use
0x0000232d	HCV_ID_ltest5	int variable that users can freely use
0x0000232e	HCV_ID_ltest6	int variable that users can freely use
0x0000232f	HCV_ID_ltest7	int variable that users can freely use
0x00002330	HCV_ID_ltest8	int variable that users can freely use
0x00002331	HCV_ID_ltest9	int variable that users can freely use
0x0000235a	HCV_ID_dtest0	double variable that users can freely use
0x0000235b	HCV_ID_dtest1	double variable that users can freely use
0x0000235c	HCV_ID_dtest2	double variable that users can freely use
0x0000235d	HCV_ID_dtest3	double variable that users can freely use
0x0000235e	HCV_ID_dtest4	double variable that users can freely use
0x0000235f	HCV_ID_dtest5	double variable that users can freely use
0x00002360	HCV_ID_dtest6	double variable that users can freely use

System Variables		
Addressing ID	Variable Name	Description
0x00002361	HCV_ID_dtest7	double variable that users can freely use
0x00002362	HCV_ID_dtest8	double variable that users can freely use
0x00002363	HCV_ID_dtest9	double variable that users can freely use
0x0000238c	HCV_ID_mtest	double[10] array variable that users can freely use

Table 17.1.1.2

Axis Variables		
Addressing ID	Variable Name	Description
0x83□□0015	HCV_ID_motion_type	Motion type
0x83□□0033	HCV_ID_pos_tr	In-position convergence radius
0x83□□0034	HCV_ID_pos_tr_t	In-position settling time
0x83□□0065	HCV_ID_sw_RL	Software right limit
0x83□□0066	HCV_ID_sw_LL	Software left limit
0x83□□0067	HCV_ID_vel_lim	Maximum velocity limit
0x83□□0068	HCV_ID_acc_lim	Maximum acceleration limit
0x83□□0069	HCV_ID_dec_lim	Maximum deceleration limit
0x83□□0079	HCV_ID_max_pos_err	Position error limit
0x83□□007a	HCV_ID_max_comp_lim	Position compensation limit
0x83□□00a0	HCV_ID_home_status	Homing state
0x83□□00a1	HCV_ID_home_method	Homing method
0x83□□00a2	HCV_ID_home_fast_vel	Fast homing velocity
0x83□□00a3	HCV_ID_home_slow_vel	Slow homing velocity
0x83□□00a4	HCV_ID_home_timeout	Homing delay time
0x83□□00a5	HCV_ID_home_acc	Homing acceleration
0x83□□00a6	HCV_ID_home_offset	Homing position offset
0x83□□00d3	HCV_ID_max_vel	Target velocity
0x83□□00d4	HCV_ID_max_acc	Target acceleration
0x83□□00d5	HCV_ID_max_dec	Target deceleration
0x83□□00d7	HCV_ID_sm_factor	Smooth time
0x83□□00db	HCV_ID_vel_scale	Velocity scale (0~100)
0x83□□00dd	HCV_ID_p2p_del	P2P motion waiting time
0x83□□00de	HCV_ID_p2p_pos1	P2P position 1
0x83□□00df	HCV_ID_p2p_pos2	P2P position 2
0x83□□00e0	HCV_ID_p2p_repeat	Repeat P2P motion
0x83□□00e1	HCV_ID_rlt_dist	Relative move distance
0x83□□00e2	HCV_ID_en_motionManager	Motion axis selection of Motion Manager
0x83□□00e3	HCV_ID_acc_time	Acceleration time
0x83□□00e4	HCV_ID_dec_time	Deceleration time

Axis Variables		
Addressing ID	Variable Name	Description
0x83□□00e9	HCV_ID_map_io_type	Error compensation type
0x83□□0117	HCV_ID_rollover_turns	Rollover turns
0x83□□0119	HCV_ID_rollover_val	Rollover value
0x83□□0193	HCV_ID_gant_pair	Gantry pair ID of gantry configuration
0x83□□01f7	HCV_ID_en_delay	Time out for axis enabling
0x83□□01ff	HCV_ID_fb_ratio_pos	Servo drive position resolution; length unit (denominator)
0x83□□0200	HCV_ID_fb_ratio_cnt	Servo drive position resolution; unit: count (numerator)
0x83□□0209	HCV_ID_fb_curr_ratio_curr	Servo drive current resolution; current unit (denominator)
0x83□□020a	HCV_ID_fb_curr_ratio_cnt	Servo drive current resolution; unit: count (numerator)
0x83□□0213	HCV_ID_rotor_inertia	Rotor intertia ratio of the motor
0x83□□0214	HCV_ID_force_constant	Torque constant of the motor
0x83□□0263	HCV_ID_last_err	Axis error code
0x83□□03c2	HCV_ID_gear_ratio	Gear ratio

Note: Symbols □□ will be the axis ID in hexadecimal format. For example, 01 stands for axis index 01; 0f stands for axis index 15.

Table 17.1.1.3

Axis Group Variables		
Addressing ID	Variable Name	Description
0x82□□0002	HCV_ID_grp_num_axis	Number of axes for axis group
0x82□□00ca	HCV_ID_grp_lin_vel_lim	Velocity limit for axis group’s linear motion
0x82□□00cb	HCV_ID_grp_lin_acc_lim	Acceleration limit for axis group’s linear motion
0x82□□00cc	HCV_ID_grp_lin_dec_lim	Deceleration limit for axis group’s linear motion
0x82□□00d4	HCV_ID_grp_ang_vel_lim	Velocity limit for axis group’s rotary motion
0x82□□00d5	HCV_ID_grp_ang_acc_lim	Acceleration limit for axis group’s rotary motion
0x82□□00d6	HCV_ID_grp_ang_dec_lim	Deceleration limit for axis group’s rotary motion
0x82□□00dd	HCV_ID_grp_lin_vel	Target velocity for axis group’s linear motion
0x82□□00de	HCV_ID_grp_lin_acc	Target acceleration for axis group’s linear motion
0x82□□00df	HCV_ID_grp_lin_dec	Target deceleration for axis group’s linear motion
0x82□□00e0	HCV_ID_grp_lin_sf	Smooth time for axis group’s linear motion
0x82□□00e1	HCV_ID_grp_lin_acc_time	Target acceleration time for axis group’s linear motion
0x82□□00e2	HCV_ID_grp_lin_dec_time	Target deceleration time for axis group’s linear motion
0x82□□00e7	HCV_ID_grp_ang_vel	Target velocity for axis group’s rotary motion
0x82□□00e8	HCV_ID_grp_ang_acc	Target acceleration for axis group’s rotary motion
0x82□□00e9	HCV_ID_grp_ang_dec	Target deceleration for axis group’s rotary motion
0x82□□00ea	HCV_ID_grp_ang_sf	Smooth time for axis group’s rotary motion

Axis Group Variables		
Addressing ID	Variable Name	Description
0x82□□00eb	HCV_ID_grp_ang_acc_time	Target acceleration time for axis group’s rotary motion
0x82□□00ec	HCV_ID_grp_ang_dec_time	Target deceleration time for axis group’s rotary motion
0x82□□00f1	HCV_ID_grp_coord_sys	Coordinate system for axis group
0x82□□00f2	HCV_ID_grp_buffer_mode	Buffer mode for axis group
0x82□□00f3	HCV_ID_grp_trans_mode	Transition mode for axis group
0x82□□00f4	HCV_ID_grp_trans_vel	Transition velocity for axis group
0x82□□00f5	HCV_ID_grp_trans_dis	Tramsition distance for axis group
0x82□□00f6	HCV_ID_grp_trans_dev	Tramsition deviation for axis group
0x82□□00f7	HCV_ID_grp_trans_curvature	Tramsition curvature for axis group
0x82□□0104	HCV_ID_grp_vel_scale	Velocity scale (0~100) for axis group
0x82□□0119	HCV_ID_grp_shaper_fr	InShape filter’s frequency for axis group
0x82□□011a	HCV_ID_grp_shaper_xi	InShape filter’s damping ratio for axis group
0x82□□038f	HCV_ID_grp_last_err	Axis group error code

Note: Symbols □□ will be the axis group ID in hexadecimal format. For example, 01 stands for axis group index 01; 0f stands for axis group index 15.

17.2 Servo drive variable operation

17.2.1 ReadSDO

Purpose

To read the object value of the slave through SDO.

Syntax

```
double ReadSDO(
    int slv_id,
    int obj_index,
    int obj_subindex,
    int obj_length
);
```

Parameter

slv_id [in]	Slave index.
obj_index [in]	Index of the slave object.
obj_subindex [in]	Subindex of the slave object.
obj_length [in]	Byte length of the slave object.

Return value

It will return a **SDO object** value of **double** type if the function succeeds, a **-1** value of **double** type if the function fails.

Example

```
void main()
{
    double value= ReadSDO(0, 0x6041, 0 , 2);
    Print("value = %f", value);
}
```

Requirement

Minimum supported version	iA Studio 3.0

17.2.2 ReadSDOEx

Purpose

To read the object value of the slave through SDO and store the data in a pointer to the buffer.

Syntax

```
int ReadSDOEx(
    int slv_id,
    int obj_index,
    int obj_subindex,
    int obj_length,
    double* obj_value
);
```

Parameter

slv_id [in]	Slave index.
obj_index [in]	Index of the slave object.
obj_subindex [in]	Subindex of the slave object.
obj_length [in]	Byte length of the slave object.
obj_value [out]	A pointer to the buffer to store the returned SDO object value.

Return value

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

Example

```
void main()
{
    double value;
    int rtn;
    rtn = ReadSDOEx(0, 0x6041, 0 , 2, &value);
    Print("return = %d, value = %f", rtn, value);
}
```

Requirement

Minimum supported version	iA Studio 3.0

17.2.3 WriteSDO

Purpose

To write the object value of the slave through SDO.

Syntax

```
int WriteSDO(
    int slv_id,
    int obj_index,
    int obj_subindex,
    int obj_length,
    double obj_value
);
```

Parameter

slv_id [in]	Slave index.
obj_index [in]	Index of the slave object.
obj_subindex [in]	Subindex of the slave object.
obj_length [in]	Byte length of the slave object.
obj_value [in]	The SDO object value to be written.

Return value

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

Requirement

Minimum supported version	iA Studio 3.0

17.2.4 ReadPDO

Purpose

To read PDO object value of the slave through PDO.

Syntax

```
double ReadPDO(
    int slv_id,
    int obj_index,
    int obj_subindex,
);
```

Parameter

slv_id [in]	Slave index.
obj_index [in]	Index of the slave object.
obj_subindex [in]	Subindex of the slave object.
obj_value [out]	A pointer to the buffer to store the returned PDO object value.

Return value

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

Example

```
void main()
{
    double value;
    int rtn;
    rtn = ReadPDO(0, 0x6041, 0, &value);
    Print("value = %f", value);
}
```

Requirement

Minimum supported version	iA Studio 3.0

17.2.5 ReadPDOEx

Purpose

To read PDO object value of the slave through PDO.

Syntax

```
int ReadPDOEx(
    int slv_id,
    int obj_index,
    int obj_subindex,
    double* obj_value
);
```

Parameter

slv_id [in]	Slave index.
obj_index [in]	Index of the slave object.
obj_subindex [in]	Subindex of the slave object.
obj_value [out]	A pointer to the buffer to store the returned PDO object value.

Return value

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

Example

```
void main()
{
    double value;
    value = ReadPDOEx(0, 0x6041, 0, &value);
    Print("value = %f", value);
}
```

Requirement

Minimum supported version	iA Studio 3.0

17.2.6 WritePDO

Purpose

To write PDO object of the slave through PDO.

Syntax

```
int WritePDO(
    int slv_id,
    int obj_index,
    int obj_subindex,
    double obj_value
);
```

Parameter

slv_id [in]	Slave index.
obj_index [in]	Index of the slave object.
obj_subindex [in]	Subindex of the slave object.
obj_value [in]	The PDO object value to be written.

Return value

If other sources write to the object at the same time, there is a risk of being overwritten.

Requirement

Minimum supported version	iA Studio 3.0

17.2.7 ForceWritePDO

Purpose

To force write PDO object of the slave through PDO.

Syntax

```
int ForceWritePDO(
    int slv_id,
    int obj_index,
    int obj_subindex,
    double obj_value
);
```

Parameter

slv_id [in]	Slave index.
obj_index [in]	Index of the slave object.
obj_subindex [in]	Subindex of the slave object.
obj_value [in]	The PDO object value to be written.

Return value

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

Requirement

Minimum supported version	iA Studio 3.0

17.2.8 ReleasePDO

Purpose

To release force written PDO object and use it with ForceWritePDO.

Syntax

```
int ReleasePDO(
    int slv_id,
    int obj_index,
    int obj_subindex
);
```

Parameter

slv_id [in]	Slave index.
obj_index [in]	Index of the slave object.
obj_subindex [in]	Subindex of the slave object.

Return value

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

Requirement

Minimum supported version	iA Studio 3.0

17.3 Controller variable operation

17.3.1 GetConfigVar

Purpose

To get the variable value of the controller.

Syntax

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

Parameter

hcv_id [in]	HIMC controller variable ID. **Refer to section 17.1.2 for the definition.**
result [out]	It will return an **int** value **0** if the function succeeds, a **-1** value if the function fails.

Return value

The value of variable.

Example

```
void main()
{
    int result = 0;
    //  0x83020065 is axis 2's software right limit.
    double SW_RL = GetConfigVar(0x83020065, &result);
    Print("SW_RL = %f", SW_RL);
}
```

Requirement

Minimum supported version	iA Studio 1.1

17.3.2 SetConfigVar

Purpose

To set the variable value of the controller.

Syntax

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

Parameter

hcv_id [in]	HIMC controller variable ID. **Refer to section 17.1.2 for the definition.**
value [in]	New variable value.

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's software right
limit.
    Print("SW_RL = %f", GetConfigVar(0x83000065, &result));
}
```

Requirement

Minimum supported version	iA Studio 1.1

18. Error functions

18.1 Overview

HIMC offers 32-bit error codes to represent the related error messages. With the functions provided in this chapter, users can get or clear the error code of system, axis and axis group. (The error codes, names and descriptions of each category are listed in section 18.1.1 to 18.1.3.) The type of error code is 0x□□□□□□□□, where “0x” indicates the value is in hexadecimal system. Its rule is the same as that of controller variable addressing ID, which is explained as follows:

4. The 1st and 2nd value of error code indicates “category of controller variable”. 0x00□□□□□□ belongs to system variable; 0x83□□□□□□ belongs to axis variable; 0x82□□□□□□ belongs to axis group variable.
5. The 3rd and 4th value of error code indicates “axis ID or axis group ID”. For example, axis variable 0x8302□□□□ is a variable storing axis index 02, while axis group variable 0x8201□□□□ is a variable storing axis group index 01.
6. The 5th to 8th value of error code indicates “variable ID”. Refer to section 18.1.1 to 18.1.3 for details.

Note: Since the return value of function is in decimal system, users must convert it to hexadecimal system by themselves to get the correct error code.

18.1.1 System error messages

Table 18.1.1.1

System Error Codes		
Error Code	Error Name	Description
0x00000001	eERR_HCV_ID_NOT_FOUND	The variable ID was not found.
0x00000002	eERR_DATA_EXCEEDED	The requested data is out of range.
0x00000003	eERR_HCV_IS_READ_ONLY	Read-only parameter.
0x00000004	eERR_HCV_VALUE_OUT_OF_RANGE	The input value is out of range.
0x00000064	eERR_EMERGENCY_STOP	Emergency stop activated. Disable all axes and stop all tasks.
0x00000100	eERR_MAIL_BOX_BUSY	The mailbox between controller and slave is busy.
0x00000101	eERR_VAR_NOT_IN_SLV_DB	The slave variable was not found.
0x00000102	eERR_VAR_NOT_REGYET	The slave variable cannot be read.
0x00000103	eERR_READ_VAR_NO_RECV	There was no response from slave.
0x00000104	eERR_PREV_SLV_CMD_NOT_FIN	The previous command to slave is not finished.
0x00000105	eERR_SLV_ID_INVALID	The slave ID is invalid.
0x00000106	eERR_PDO_NUM_EXCEED	The number of PDO is out of range.
0x00000107	eERR_NOT_VALID_TASKID	The task ID is invalid.
0x00000108	eERR_TASK_IS_RUNNING	The task is already running.
0x00000109	eERR_FUNC_NOT_IN_TASK	The function was not found in task.
0x0000010a	eERR_TASK_EMPTY	The task is empty.
0x0000010b	eERR_TASK_NOT_RUNNING	The task is not running.
0x0000012c	eERR_NIC_INIT_TOUT	The network port of mega-ulink is not ready.
0x0000012d	eERR_HARDWARE_MISMATCH	The hardware is unrecognized.
0x0000012e	eERR_SLAVE_NUM_MISMATCH	The number of slaves is different from configuration.
0x0000012f	eERR_INVALID_PDO	The PDO is invalid.
0x00000130	eERR_INVALID_MCK_CNFG	The configuration of motion kernel is invalid.
0x00000138	eERR_HIMC_LOAD_CONFIG_FAIL	Load configuration from SSD failed. Please save it again.
0x00000139	eERR_HIMC_SAVE_CONFIG_FAIL	Store configuration to HIMC failed. Please save it again.
0x0000013a	eERR_HIMC_SAVE_CONFIG_COPY_FAIL	Store configuration to HIMC failed. Cannot save file into SAVE folder.
0x0000013b	eERR_HIMC_SAVE_UPDATE_PRM_TIMEOUT	Store configuration to HIMC failed. Update Prm values timeout.
0x0000013c	eERR_ETHERCAT_LICENSE_MISMATCH	The authorization of EtherCAT is invalid.
0x000001f4	eERR_ISR_NOT_STABLE	The period of interrupt is not stable.
0x000001f5	eERR_MCK_OVERLOAD	The motion kernel is overloaded.
0x000001f6	eERR_ISR_OVERLOAD	The CPU is overloaded.
0x00001388	eERR_HMPL_INVALID_ARG	The arguments are invalid in HMPL.
0x00001389	eERR_HMPL_INVALID_PTR	The pointer is invalid in HMPL.
0x0000138a	eERR_HMPL_STACK_OVERFLOW	Stack overflow in HMPL.
0x0000138b	eERR_HMPL_ILLEGAL_MEM_OP	The operation of memory is illegal in HMPL.
0x0000138c	eERR_HMPL_MOTION_NOT_READY	Motion function should be called in synchronized state.

System Error Codes		
Error Code	Error Name	Description
0x0000138d	eERR_HMPL_STR_TOO_LONG	String length is out of range.
0x0000138e	eERR_HMPL_INVALID_STR_FORMAT	String format is invalid.
0x0000138f	eERR_HMPL_ARG_OUT_OF_RANGE	The argument is out of range.
0x00001392	eERR_HMPL_ASCII_AGENT_RUNNING	ASCII agent is already running.
0x0000139c	eERR_HMPL_CANNOT_RUN_IN_DEBUG	The function cannot run in debug mode.
0x000013a6	eERR_HMPL_TOO_MANY_BRK_POINT	There are too many break points in the task.
0x000013ec	eERR_HMPL_MUTEX_LOCK_TWICE	Cannot lock the same mutex twice in the same task.
0x00001450	eERR_HMPL_INVALID_SYS_TIME_MEMORY	Buffer too small, minimum size must be 30 Byte.
0x00001451	eERR_HMPL_NOT_SUPPORTED	This HMPL function not supported for this platform.
0x00001452	eERR_HMPL_CLIENT_NOT_CONNECTED	Cannot send as client disconnected.
0x0000176f	eERR_HMPL_INTERNAL_ERROR	HMPL internal error.
0x00001770	eERR_HMPL_EXEC_FAILED	HMPL function execution failed.
0x00001771	eERR_HMPL_ASM_LOAD_FAILED	HMPL compilation failed, assembly file empty or not generated.
0x00001772	eERR_HMPL_STARTTASK_TIMEOUT	HMPL StartTask function timeout.
0x00001773	eERR_HMPL_STOPTASK_TIMEOUT	HMPL StopTask function timeout.
0x000017d4	eERR_ASCII_CONNECT_TIMEOUT	ASCII client connection timeout.
0x000017d5	eERR_ASCII_CONNECT_FAILED	ASCII client connection failed. Please check ip and port.
0x000017d6	eERR_ASCII_MULTI_CONNECTING	Multiple ASCII clients connecting at the same time.
0x000017d7	eERR_ASCII_MULTI_DISCONNECTING	Multiple ASCII clients disconnecting at the same time.
0x000017d8	eERR_ASCII_DISCONNECT_TIMEOUT	ASCII client disconnection timeout.
0x000017de	eERR_ASCII_RECV_TIMEOUT	ASCII client receive timeout. Please try again later.
0x000017df	eERR_ASCII_RECV_FAIL	ASCII client receive failed. Please check if the connection is still alive.
0x000017e0	eERR_ASCII_MULTI_RECVING	Multiple ASCII clients receiving at the same time.
0x000017e8	eERR_ASCII_SEND_TIMEOUT	ASCII client send timeout. Please try again later.
0x000017e9	eERR_ASCII_SEND_FAIL	ASCII client send failed. Please check if the connection is still alive.
0x000017ea	eERR_ASCII_MULTI_SENDING	Multiple ASCII clients sending at the same time.
0x00001838	eERR_MODBUS_CONNECT_TIMEOUT	Modbus client connection timeout.
0x00001839	eERR_MODBUS_CONNECT_FAILED	Modbus client connection failed. Please check ip.
0x0000183a	eERR_MODBUS_MULTI_CONNECTING	Multiple Modbus clients connecting at the same time.
0x0000183b	eERR_MODBUS_MULTI_DISCONNECTING	Multiple Modbus clients disconnecting at the same time.
0x0000183c	eERR_MODBUS_DISCONNECT_TIMEOUT	Modbus client disconnection timeout.
0x0000183d	eERR_MODBUS_DATALENGTH_ERR	Modbus client’s read/write data number exceeds the limitation.
0x0000183e	eERR_MODBUS_SOCKET_BUSY	Modbus client deals with two or more commands at the same time.
0x0000183f	eERR_MODBUS_JOB_TIMEOUT	Modbus client job execution timeout. Please try again later.

System Error Codes		
Error Code	Error Name	Description
0x00001840	eERR_MODBUS_JOB_FAIL	Modbus client job execution failed. Please check if the connection is still alive.

18.1.2 Axis error messages

The following error codes appear due to an error or invalid operation in an axis. Symbols □□ will be the axis ID in hexadecimal format. For example, 01 stands for axis index 01; 0f stands for axis index 15.

Table 18.1.2.1

Axis Error Codes		
Error Code	Error Name	Description
0x83□□000a	eERR_AXIS_CMD_UNKOWN	The command name is unknown.
0x83□□001e	eERR_AXIS_CMD_QUEUE_FULL	Axis command queue is full.
0x83□□0064	eERR_AXIS_CMD_INVALID_STATE	The axis is unable to execute the command in current motion state.
0x83□□006e	eERR_AXIS_CMD_INVALID_ENABLED	The command is not allowed while enabled.
0x83□□0078	eERR_AXIS_CMD_INVALID_DISABLED	The command is not allowed while disabled.
0x83□□0082	eERR_AXIS_CMD_INVALID_MOVING	The axis is unable to execute the command while moving.
0x83□□008c	eERR_AXIS_CMD_INVALID_STOPPING	The command is invalid when axis stops moving.
0x83□□0096	eERR_AXIS_CMD_INVALID_ERROR_STATE	The command is invalid when axis is in ErrorStop state.
0x83□□00a0	eERR_AXIS_CMD_INVALID_IN_SYNC	The command is invalid when axis is in synchronized motion state.
0x83□□00aa	eERR_AXIS_CMD_INVALID_GEAR_MASTER	The command is invalid when axis is the gear master axis.
0x83□□00b4	eERR_AXIS_CMD_INVALID_PP_MODE	The command is invalid when axis is in PP mode.
0x83□□00be	eERR_AXIS_CMD_INVALID_MAP_SWITCHIN G	The command is invalid when axis is switching the compensation map.
0x83□□00c8	eERR_AXIS_CMD_INVALID_INPUTSHAPING _ENABLED	The axis is unable to execute the command when position command shaping function is activated.
0x83□□00d2	eERR_AXIS_CMD_INVALID_COMP_ENABLE D	The axis is unable to execute the command when dynamic compensation is enabled.
0x83□□00dc	eERR_AXIS_CMD_INVALID_GANTRY_MODE	The axis is unable to execute the command in gantry mode.
0x83□□00e6	eERR_AXIS_CMD_INVALID_GROUPED	The command is not allowed when axis is in an axis group.
0x83□□00f0	eERR_AXIS_CMD_INVALID_CONTROL_MOD E	The command is invalid in current control mode.
0x83□□00fa	eERR_AXIS_CMD_INVALID_OP_MODE	The operational mode is invalid.
0x83□□0104	eERR_AXIS_CMD_INVALID_BUFFER_MODE	The axis buffer mode is invalid.
0x83□□010e	eERR_AXIS_CMD_INVALID_TP_ENABLED	The command is not allowed when touch probe is enabled.
0x83□□012c	eERR_AXIS_CMD_INVALID_PARAMETER	The parameter of axis command is invalid.
0x83□□0136	eERR_AXIS_CMD_INVALID_POS	Axis target position is out of allowable range.
0x83□□0140	eERR_AXIS_CMD_INVALID_VEL	Axis velocity setting is out of allowable range.
0x83□□014a	eERR_AXIS_CMD_INVALID_ACC	Axis acceleration setting is out of allowable range.
0x83□□0154	eERR_AXIS_CMD_INVALID_DEC	Axis deceleration setting is out of allowable range.
0x83□□015e	eERR_AXIS_CMD_INVALID_JERK	Axis jerk setting is out of allowable range.

Axis Error Codes		
Error Code	Error Name	Description
0x83□□0168	eERR_AXIS_CMD_INVALID_SM_TIME	Axis smooth time setting is out of allowable range.
0x83□□0172	eERR_AXIS_CMD_INVALID_KILL_DEC	Axis kill deceleration setting is out of allowable range.
0x83□□017c	eERR_AXIS_CMD_INVALID_VEL_SCALE	Axis velocity scale setting is out of allowable range.
0x83□□0190	eERR_AXIS_COMP_NOT_CNFG	Axis dynamic compensation settings have not been configured properly.
0x83□□01c2	eERR_AXIS_CMD_INVALID_MASTER_SLAVE_CONNECTION	Master-slave relationship setting is invalid.
0x83□□01cc	eERR_AXIS_CMD_INVALID_SLAVE_ID	Slave ID setting is invalid.
0x83□□01d6	eERR_AXIS_CMD_INVALID_GEAR_RATIO	The gear ratio setting of slave axis is out of allowable range.
0x83□□01f4	eERR_AXIS_CMD_INVALID_ROLLOVER_POS	Invalid axis rollover position, should be a positive value.
0x83□□03f2	eERR_AXIS_DRIVE_FAULT	The drive has reported a fault. Please check the corresponding error message in the drive.
0x83□□03fc	eERR_AXIS_DRIVE_ABNORMAL_DISABLE	The drive is abnormally disabled.
0x83□□0406	eERR_AXIS_DRIVE_ENABLE_TOUT	It took too long to enable the drive.
0x83□□0410	eERR_AXIS_DRIVE_CLEAR_ERROR_TOUT	It took too long to clear drive error.
0x83□□041a	eERR_AXIS_DRIVE_DISABLE_TOUT	It took too long to disable the drive.
0x83□□0424	eERR_AXIS_DRIVE_HOME_TOUT	It took too long to home the axis.
0x83□□042e	eERR_AXIS_DRIVE_HOME_FAILED	Axis homing error. Please check error code from drive.
0x83□□0456	eERR_AXIS_VEL_LIMIT	The reference velocity has exceeded the velocity limit.
0x83□□0460	eERR_AXIS_ACC_LIMIT	The reference acceleration has exceeded the acceleration limit.
0x83□□046a	eERR_AXIS_CURR_LIMIT	The current command has exceeded the current limit.
0x83□□0474	eERR_AXIS_DAMPINGRATIO_LIMIT	The damping ratio setting of axis is out of allowable range.
0x83□□047e	eERR_AXIS_FREQUENCY_LIMIT	The frequency setting of axis is out of allowable range.
0x83□□07da	eERR_AXIS_SWRL	Axis reference position reached right software limit.
0x83□□07e4	eERR_AXIS_SWLL	Axis reference position reached left software limit.
0x83□□07ee	eERR_AXIS_HWRL	Axis right hardware limit signal triggered.
0x83□□07f8	eERR_AXIS_HWLL	Axis left hardware limit signal triggered.
0x83□□0802	eERR_AXIS_COMP_LIMIT	Axis compensation position has exceeded maximum compensation limit.
0x83□□083e	eERR_AXIS_PERR	Axis position error has exceeded the protection limit. Please first check if there is any mechanical interference for motor motion.
0x83□□0848	eERR_AXIS_VERR	Axis velocity error has exceeded the protection limit. Please first check if there is any mechanical interference for motor motion.
0x83□□08a2	eERR_AXIS_PVT_MOTION_VEL_LIMIT	Velocity of axis PVT motion has exceeded the protection limit. Please first check if the given parameters are valid.
0x83□□08ac	eERR_AXIS_PVT_MOTION_ACC_LIMIT	Acceleration of axis PVT motion has exceeded the protection limit. Please first check if the given parameters are valid.

Axis Error Codes		
Error Code	Error Name	Description
0x83□□08b6	eERR_AXIS_PVT_MOTION_INVALID_TIME	Time sequence of axis PVT motion is invalid. Please first check if the given parameters are valid.
0x83□□0bb8	eERR_AXIS_CTRL_ERR	Axis internal control error.
0x83□□0fa0	eERR_AXIS_CMD_GEAR_DISABLED	Gear command is not allowed while gear is disabled.
0x83□□0fa1	eERR_AXIS_CMD_INVALID_AXIS_IN_CAM	Gear command is invalid when axis is in cam.
0x83□□1388	eERR_CAM_CMD_INVALID_ENGAGE_WINDOW	Cam engage window is out of allowable range.
0x83□□1389	eERR_CAM_CMD_INVALID_ENGAGE_POSITION	Cam engage position is out of cam table domain.
0x83□□138a	eERR_CAM_CMD_INVALID_MASTER_SCALE_FACTOR	Cam master scale factor is out of allowable range.
0x83□□138b	eERR_CAM_CMD_INVALID_CAM_SCALE_FACTOR	Cam scale factor is out of allowable range.
0x83□□138c	eERR_CAM_CMD_INVALID_CAMTABLE_ID	Cam table ID is out of allowable range.
0x83□□138d	eERR_CAM_CMD_INVALID_DISENGAGE_WINDOW	Cam disengage window is out of allowable range.
0x83□□138e	eERR_CAM_CMD_INVALID_DISENGAGE_POSITION	Cam disengage position is out of allowable range.
0x83□□138f	eERR_CAM_CMD_INVALID_OPERATION_IN_ENGAGED_STATE	Cam command is invalid in engage state.
0x83□□1390	eERR_CAM_CMD_INVALID_START_MODE	Cam start mode does not correspond to a valid enumeration value.
0x83□□1391	eERR_CAM_CMD_INVALID_MOVE_MODE	Cam move mode does not correspond to a valid enumeration value.
0x83□□1392	eERR_CAM_ENGAGED_FAILED	CamMaster may pass through the engage window because engage window is too small.
0x83□□1393	eERR_CAM_CMD_NOTINUSE	End of profile mode is not in use right now
0x83□□1394	eERR_CAM_CMD_CAM_DISABLED	Cam command is not allowed while cam is disabled
0x83□□1395	eERR_CAM_CMD_INVALID_AXIS_NOT_INCAM	Cam command is invalid when axis is not in cam
0x83□□1396	eERR_CAM_CMD_INVALID_AXIS_NOT_IN_DISENGAGED	Cam command is invalid when axis is not in disengaged
0x83□□1397	eERR_CAM_CMD_INVALID_AXIS_IN_GEAR	Cam command is invalid when axis is in gear

18.1.3 Group error messages

The following error codes appear due to an error or invalid operation in an axis group. Symbols □□ will be the axis group ID in hexadecimal format. For example, 01 stands for axis group index 01; 0f stands for axis group index 15.

Table 18.1.3.1

Axis Group Error Codes		
Error Code	Error Name	Description
0x82□□000a	eERR_CRD_CMD_UNKNOWN	The axis group command is unknown.
0x82□□0028	eERR_CRD_CMD_AXIS_DUPLICATED	Could not add the axis since it is already in the group.
0x82□□0032	eERR_CRD_CMD_GRP_SIZE_EMPTY	The axis group is empty.
0x82□□003c	eERR_CRD_CMD_GRP_SIZE_FULL	The axis group is full and cannot hold any more axis.
0x82□□0046	eERR_CRD_CMD_INVALID_MOVING	The command is invalid while the axis group is moving.
0x82□□0050	eERR_CRD_CMD_INVALID_DISABLED	The command is invalid while the axis group is disabled.
0x82□□005a	eERR_CRD_CMD_INVALID_INPUTSHAPIN G_PARAMETER_INCOMPLETE	The parameters of axis group inshape function is incomplete.
0x82□□001e	eERR_CRD_CMD_INVALID_KIN_SETTING	The kinematics type setting is invalid.
0x82□□001f	eERR_CRD_CMD_INVALID_SPECIFIC_KIN	The command is invalid when axis group is in specific kinematics type.
0x82□□006e	eERR_CRD_CMD_INVALID_STATE	The axis group is unable to execute the command in current motion state.
0x82□□0078	eERR_CRD_CMD_QUEUE_FULL	Please wait till the last command is done.
0x82□□0082	eERR_CRD_CMD_GRP_AXIS_INVALID	The group axis is invalid.
0x82□□00d2	eERR_CRD_CMD_INVALID_POS	The axis group target position or orientation is out of allowable range.
0x82□□00dc	eERR_CRD_CMD_INVALID_LIN_VEL	The linear velocity setting of axis group is out of allowable range.
0x82□□00e6	eERR_CRD_CMD_INVALID_LIN_ACC	The linear acceleration setting of axis group is out of allowable range.
0x82□□00f0	eERR_CRD_CMD_INVALID_LIN_DEC	The linear deceleration setting of axis group is out of allowable range.
0x82□□00fa	eERR_CRD_CMD_INVALID_LIN_JERK	The linear jerk setting of axis group is out of allowable range.
0x82□□0104	eERR_CRD_CMD_INVALID_LIN_SM_TIME	The linear smooth time setting of axis group is out of allowable range.
0x82□□010e	eERR_CRD_CMD_INVALID_DAMPINGRATIO	The damping ratio setting of axis group is out of allowable range.
0x82□□0118	eERR_CRD_CMD_INVALID_FREQUENCY	The frequency setting of axis group is out of allowable range.
0x82□□0140	eERR_CRD_CMD_INVALID_ANG_VEL	The angular velocity setting of axis group is out of allowable range.
0x82□□014a	eERR_CRD_CMD_INVALID_ANG_ACC	The angular acceleration setting of axis group is out of allowable range.
0x82□□0154	eERR_CRD_CMD_INVALID_ANG_DEC	The angular deceleration setting of axis group is out of allowable range.
0x82□□015e	eERR_CRD_CMD_INVALID_ANG_JERK	The angular jerk setting of axis group is out of allowable range.

Axis Group Error Codes		
Error Code	Error Name	Description
0x82□□0168	eERR_CRD_CMD_INVALID_ANG_SM_TIME	The angular smooth time setting of axis group is out of allowable range.
0x82□□0190	eERR_CRD_CMD_INVALID_VEL_SCALE	The velocity scale of axis group is out of allowable range.
0x82□□019a	eERR_CRD_CMD_INVALID_TRANS_VEL	The transition velocity of axis group is invalid.
0x82□□01a4	eERR_CRD_CMD_INVALID_TRANS_DIS	The transition distance of axis group is invalid.
0x82□□01a5	eERR_CRD_CMD_INVALID_TRANS_DEV	The transition deviation of axis group is invalid.
0x82□□01a6	eERR_CRD_CMD_INVALID_TRANS_CURVE	The transition curvature of axis group is invalid.
0x82□□01b8	eERR_CRD_CMD_TRANS_MODE_UNKNOWN	The path transition mode name is unknown.
0x82□□01c2	eERR_CRD_CMD_COORD_SYS_UNKNOWN	The coordinate system is unknown.
0x82□□01cc	eERR_CRD_CMD_BLEND_MODE_UNKNOWN	The path blending mode name is unknown.
0x82□□01fe	eERR_CRD_CMD_LIN_INVALID_PARAM	The parameters are invalid for linear path planning.
0x82□□0262	eERR_CRD_CMD_CIRC_INVALID_PARAM	The parameters are invalid for circular path planning.
0x82□□026c	eERR_CRD_CMD_CIRC_INVALID_CENTER	The center position of circular path is too close to start / end point.
0x82□□0276	eERR_CRD_CMD_CIRC_ANGLE_SMALL	The central angle of circular path is too small.
0x82□□0280	eERR_CRD_CMD_CIRC_INVALID_RADIUS	The radius of circular path is invalid.
0x82□□028a	eERR_CRD_CMD_CIRC_INVALID_COORD	The coordinate system of circular path is invalid.
0x82□□02c6	eERR_CRD_CMD_BEZIER_INVALID_PARAM	The parameters are invalid for Bezier curve path planning.
0x82□□02d0	eERR_CRD_CMD_BSPLINE_INVALID_PARAM	The parameters are invalid for BSpline curve path planning.
0x82□□02da	eERR_CRD_CMD_COORD_INVALID_PARAM	The start position is invalid for curve path planning.
0x82□□02e4	eERR_CRD_CMD_COORD_INVALID_PARAM	The parameters are invalid for coordinate transformation.
0x82□□02ee	eERR_CRD_CMD_NURBS_INVALID_PARAM	The parameters are invalid for NURBS curve path planning.
0x82□□03f2	eERR_CRD_AXIS_ABNORMALLY_DISABLED	One or more axes in the axis group are abnormally disabled.
0x82□□03fc	eERR_CRD_AXIS_SWL	One of the axes in axis group touches software limit.

18.2 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.

Refer to section 18.1.1 for the definition.

Requirement

Minimum supported version	iA Studio 1.1

18.3 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.

Refer to section 18.1.2 for the definition.

Requirement

Minimum supported version	iA Studio 1.1

18.4 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

18.5 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.

Refer to section 18.1.3 for the definition.

Requirement

Minimum supported version	iA Studio 1.1

18.6 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

18.7 GetDriveErr

Purpose

To get the error code of a drive.

Syntax

```
int GetDriveErr(
    int    axis_id
);
```

Parameter

axis_id [in] Axis Index.

Return value

Return the error code of a drive.

Remark

Users must configure object 0x603F(Error code) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 3.0

19. Marco definition and functions

19.1 _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(1)) {
        StopTask( _TASKID_ );  //  Stop current task
      }
    }
}

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

Requirement

Minimum supported version	iA Studio 0.23

19.2 _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

19.3 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)

Remark

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

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

    //  Do something
}
```

Requirement

Minimum supported version	iA Studio 0.23

19.4 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

```
void 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

19.5 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

```
void 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

20. Homing functions

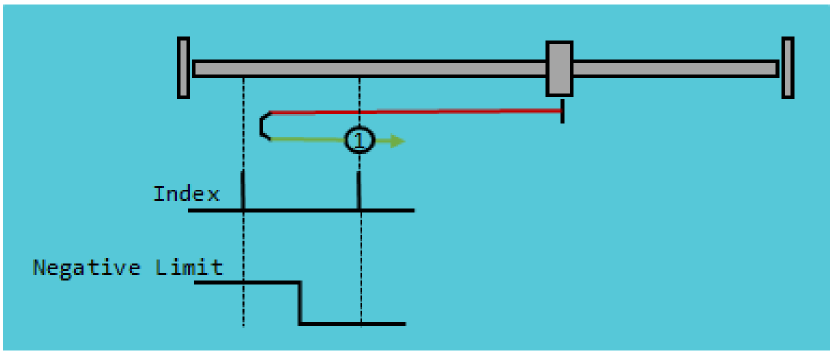
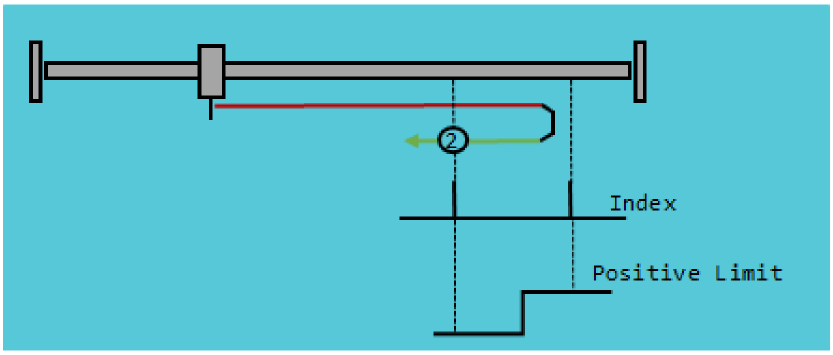
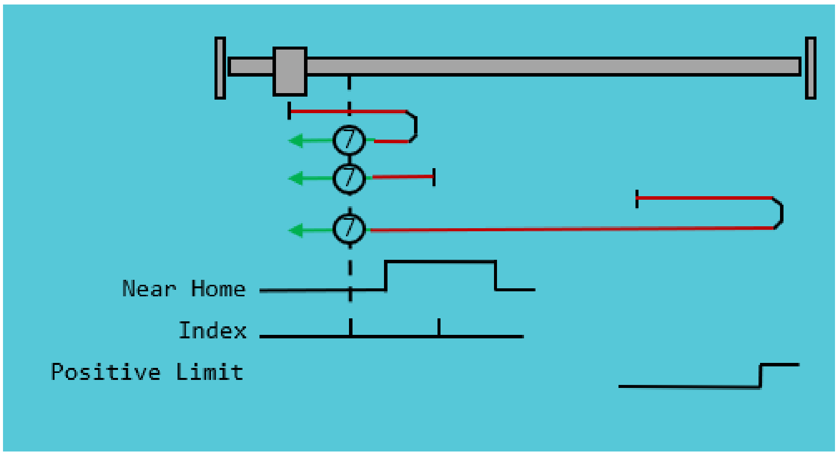
20.1 Overview

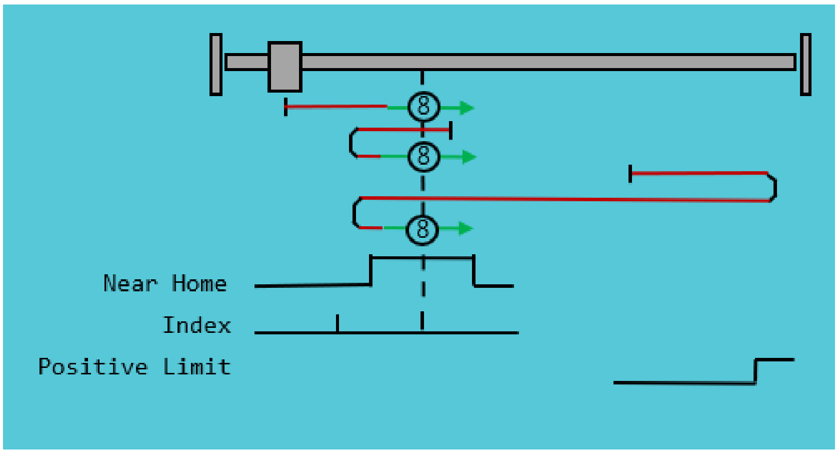
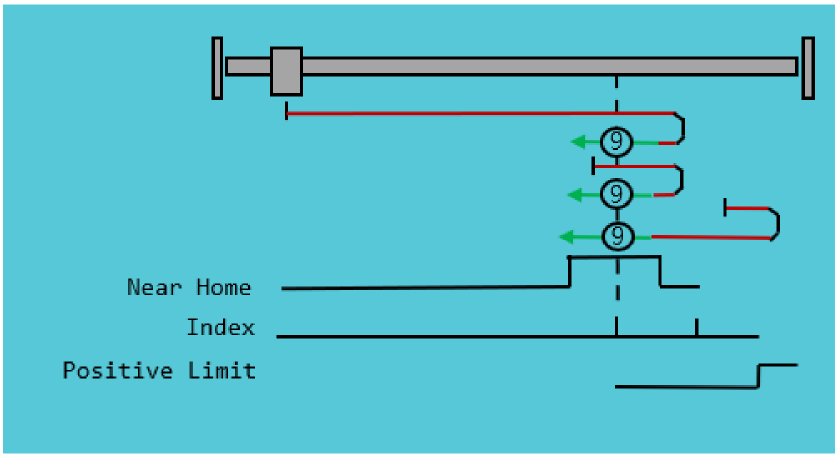
HIMC supports homing method CiA 402, which allows users to set the homing method of each axis based on stage configuration. All homing methods are listed in Table 20.1.1, and the detailed diagrams and descriptions are shown in Table 20.1.2.

Table 20.1.1

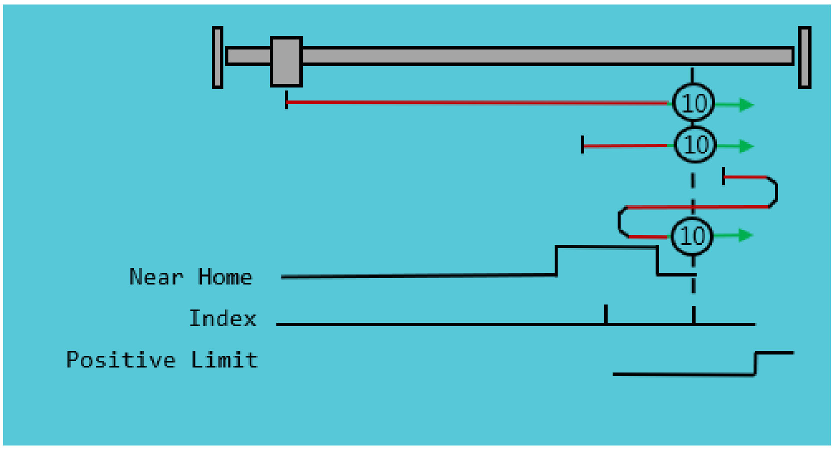
HMPL definition	Description
HOME_METHOD_1	Search for negative limit in negative direction first, and search for index in positive direction.
HOME_METHOD_2	Search for positive limit in positive direction first, and search for index in negative direction.
HOME_METHOD_7	Search for index on the left of near home rising-edge in positive direction.
HOME_METHOD_8	Search for index on the right of near home rising-edge in positive direction.
HOME_METHOD_9	Search for index on the left of near home falling-edge in positive direction.
HOME_METHOD_10	Search for index on the right of near home falling-edge in positive direction.
HOME_METHOD_11	Search for index on the right of near home rising-edge in negative direction.
HOME_METHOD_12	Search for index on the left of near home rising-edge in negative direction.
HOME_METHOD_13	Search for index on the right of near home falling-edge in negative direction.
HOME_METHOD_14	Search for index on the left of near home falling-edge in negative direction.
HOME_METHOD_17	Search for negative limit in negative direction first, and move to the position of home offset.
HOME_METHOD_18	Search for positive limit in positive direction first, and move to the position of home offset.
HOME_METHOD_19	Search for negative limit in negative direction first, and search for positive limit in positive direction. Finally, move to the middle of positive and negative limit.
HOME_METHOD_33	Search for index in negative direction first, and move to the position of home offset.
HOME_METHOD_34	Search for index in positive direction first, and move to the position of home offset.
HOME_METHOD_37	Set the current position of the motor as home position.

Table 20.1.2

HMPL definition	Homing procedure
HOME_METHOD_1	
	Search for negative limit in negative direction at fast velocity first, and search for index in positive direction at slow velocity. Take the position of index as home position, and move to home offset at slow velocity.
HOME_METHOD_2	
	Search for positive limit in positive direction at fast velocity first, and search for index in negative direction at slow velocity. Take the position of index as home position, and move to home offset at slow velocity.
HOME_METHOD_7	
	1. (not on near home) Search for near home rising-edge in positive direction first, and search for index in negative direction. 2. (on near home) Search for near home falling-edge in negative direction first, and search for index in negative direction. 3. (not on near home) Search for positive limit in positive direction first, and search for near home falling-edge in negative direction. Finally, search for index in negative direction.

HMPL definition	Homing procedure
HOME_METHOD_8	
	1. (not on near home) Search for near home rising-edge in positive direction first, and search for index in positive direction. 2. (on near home) Search for near home falling-edge in negative direction first, and search for index in positive direction. 3. (not on near home) Search for positive limit in positive direction first, and search for near home falling-edge in negative direction. Finally, search for index in positive direction.
HOME_METHOD_9	
	1. (not on near home) Search for near home falling-edge in positive direction first, and search for index in negative direction. 2. (on near home) Search for near home falling-edge in positive direction first, and search for index in negative direction. 3. (not on near home) Search for positive limit in positive direction first, and search for near home rising-edge in negative direction. Finally, search for index in negative direction.

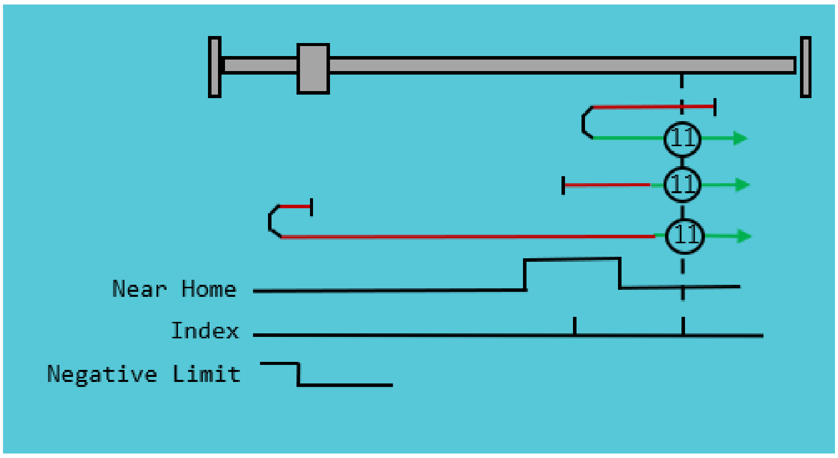
<table>
<tr><th>HMPL definition</th><th>Homing procedure</th></tr>
<tr><td rowspan="2">HOME_METHOD_10</td><td>



</td></tr>
<tr><td>

1. (not on near home) Search for near home falling-edge in positive direction first, and search for index in positive direction.
2. (on near home) Search for near home falling-edge in positive direction first, and search for index in positive direction.
3. (not on near home) Search for positive limit in positive direction first, and search for near home rising-edge in negative direction. Finally, search for index in positive direction.

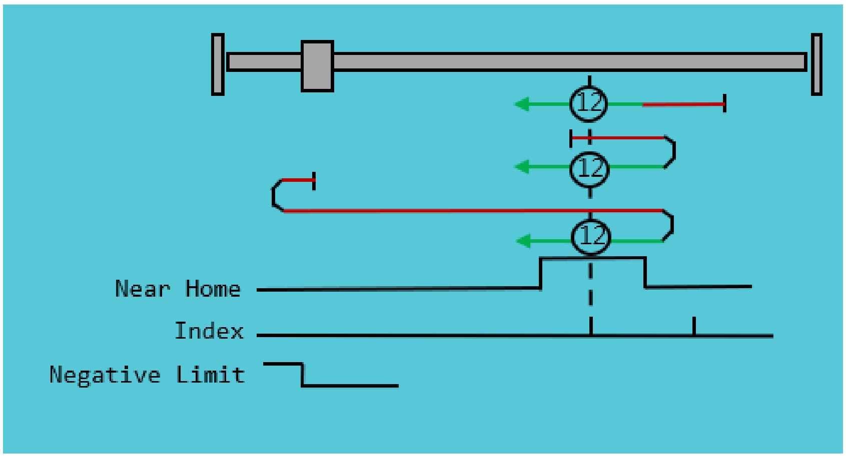
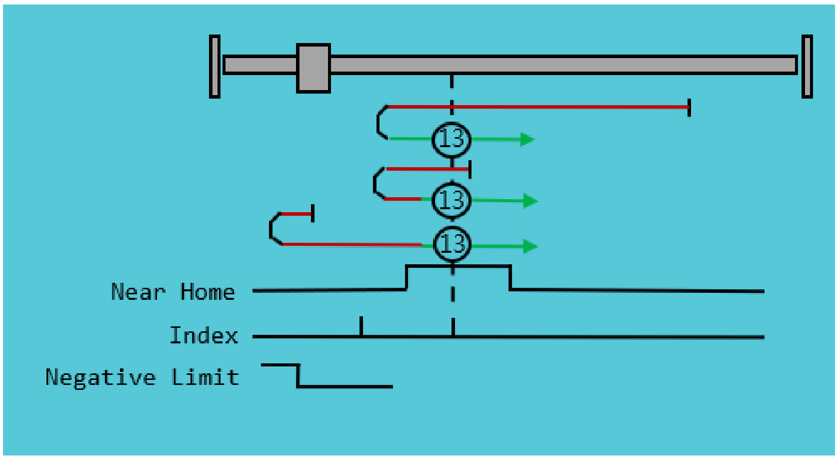
</td></tr>
<tr><td rowspan="2">HOME_METHOD_11</td><td>

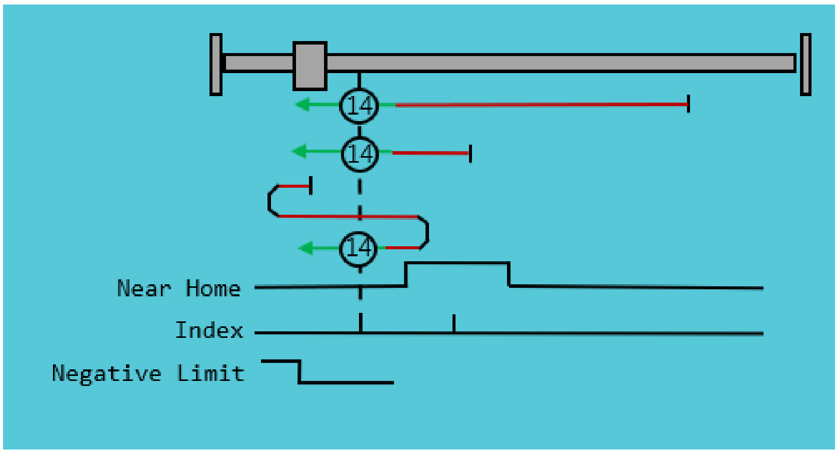
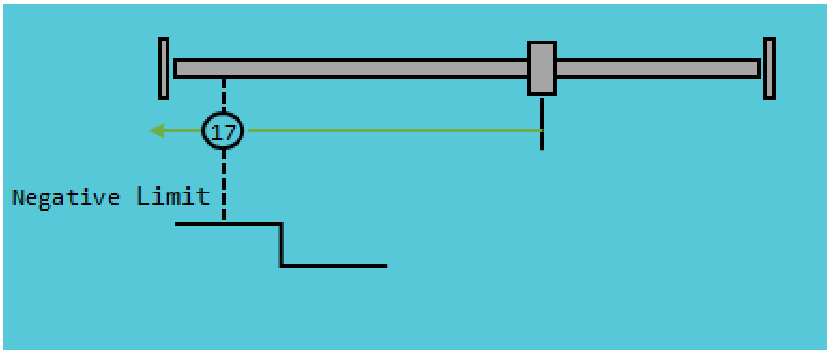
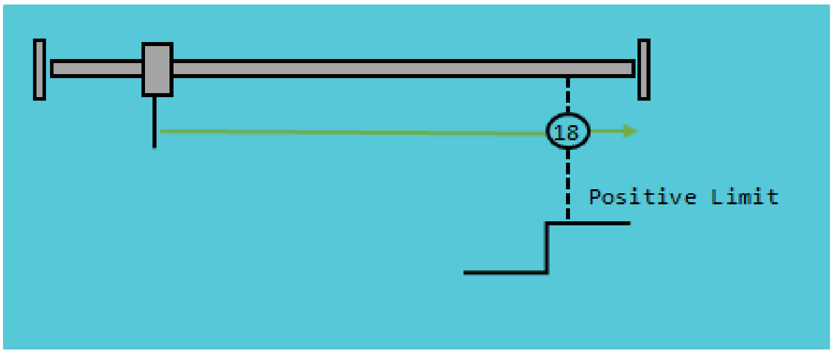


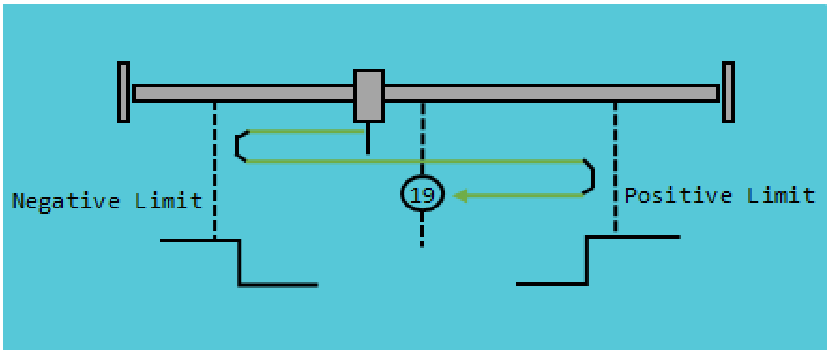
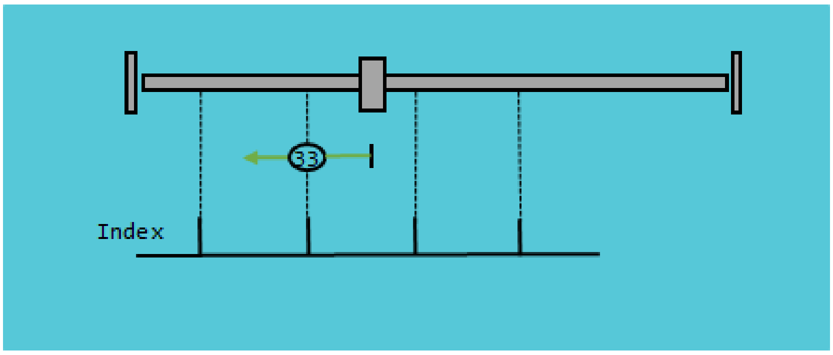
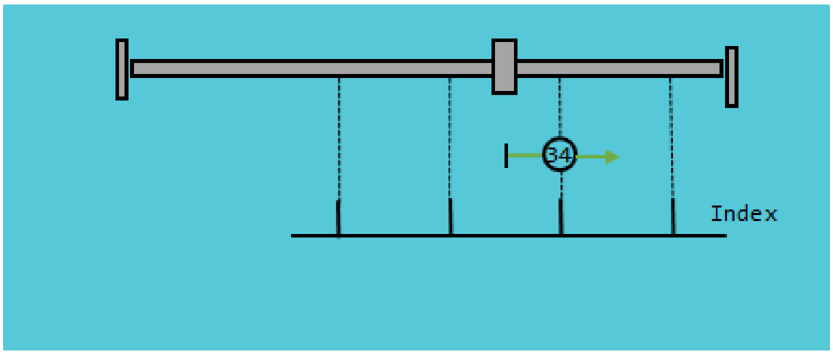
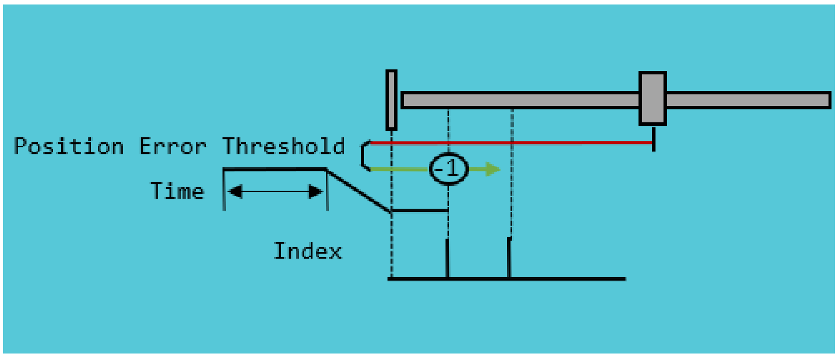
</td></tr>
<tr><td>

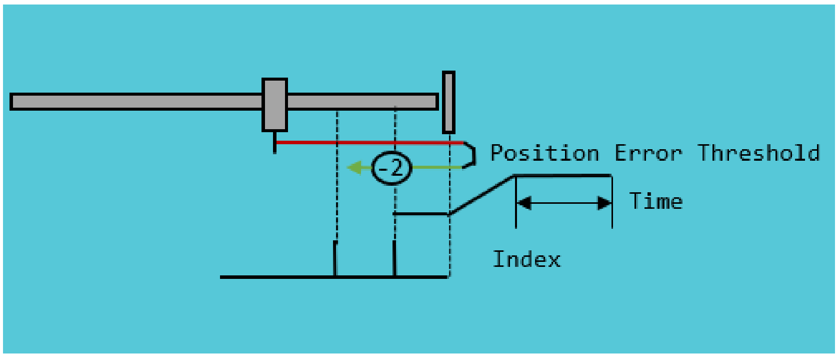
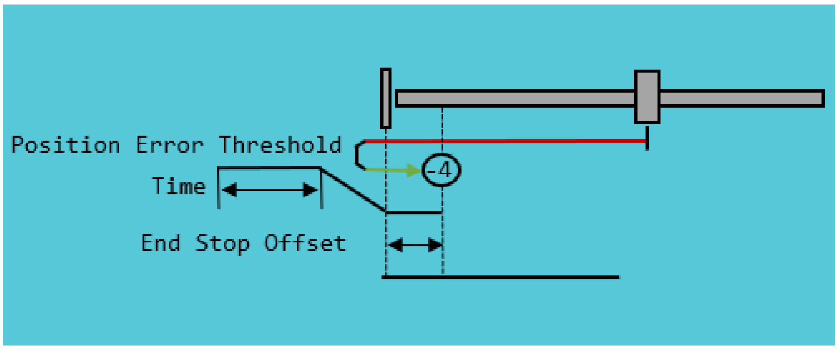
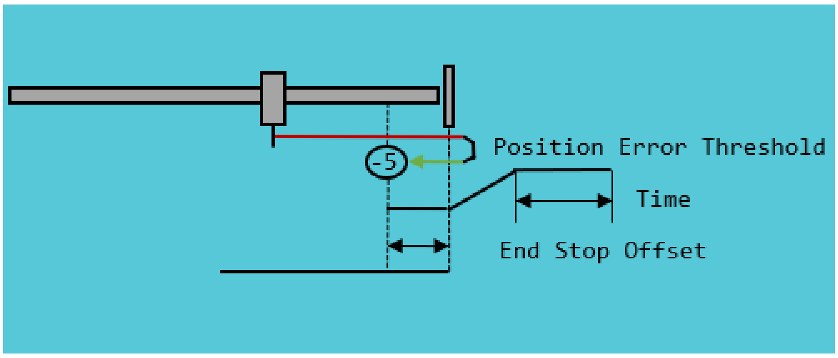
1. (not on near home) Search for near home rising-edge in negative direction first, and search for index in positive direction.
2. (on near home) Search for near home falling-edge in positive direction first, and search for index in positive direction.
3. (not on near home) Search for negative limit in negative direction first, and search for near home falling-edge in positive direction. Finally, search for index in positive direction.

</td></tr>
</table>

<table>
<tr><th>HMPL definition</th><th>Homing procedure</th></tr>
<tr><td rowspan="2">HOME_METHOD_12</td><td>
</td></tr>
<tr><td>1. (not on near home) Search for near home rising-edge in negative direction first, and search for index in negative direction.
2. (on near home) Search for near home falling-edge in positive direction first, and search for index in negative direction.
3. (not on near home) Search for negative limit in negative direction first, and search for near home falling-edge in positive direction. Finally, search for index in negative direction.</td></tr>
<tr><td rowspan="2">HOME_METHOD_13</td><td>
</td></tr>
<tr><td>1. (not on near home) Search for near home falling-edge in negative direction first, and search for index in positive direction.
2. (on near home) Search for near home falling-edge in negative direction first, and search for index in positive direction.
3. (not on near home) Search for negative limit in negative direction first, and search for near home rising-edge in positive direction. Finally, search for index in positive direction.</td></tr>
</table>

<table>
<tr><th>HMPL definition</th><th>Homing procedure</th></tr>
<tr><td rowspan="2">HOME_METHOD_14</td><td></td></tr>
<tr><td>1. (not on near home) Search for near home falling-edge in negative direction first, and search for index in negative direction.
2. (on near home) Search for near home falling-edge in negative direction first, and search for index in negative direction.
3. (not on near home) Search for negative limit in negative direction first, and search for near home rising-edge in positive direction. Finally, search for index in negative direction.</td></tr>
<tr><td rowspan="2">HOME_METHOD_17</td><td></td></tr>
<tr><td>Search for negative limit in negative direction at slow velocity. Take the position of negative limit as home position, and move to home offset at slow velocity.</td></tr>
<tr><td rowspan="2">HOME_METHOD_18</td><td></td></tr>
<tr><td>Search for positive limit in positive direction at slow velocity. Take the position of positive limit as home position, and move to home offset at slow velocity.</td></tr>
</table>

HMPL definition	Homing procedure
HOME_METHOD_19	
	Search for negative limit in negative direction at slow velocity first, and search for positive limit in positive direction at slow velocity. Move to the middle at slow velocity, and take the middle position as home position.
HOME_METHOD_33	
	Search for index in negative direction at slow velocity. Take index as home position, and move to home offset at slow velocity.
HOME_METHOD_34	
	Search for index in positive direction at slow velocity. Take index as home position, and move to home offset at slow velocity.
HOME_METHOD_N1	

HMPL definition	Homing procedure
	Search for end stop in negative direction at fast velocity first, and search for index in positive direction at slow velocity. Take index as home position.
HOME_METHOD_N2	
	Search for end stop in positive direction at fast velocity first, and search for index in negative direction at slow velocity. Take index as home position.
HOME_METHOD_N4	
	Search for end stop in negative direction at fast velocity. Take end stop as home position, and move to end stop offset in positive direction at slow velocity.
HOME_METHOD_N5	
	Search for end stop in positive direction at fast velocity. Take end stop as home position, and move to end stop offset in negative direction at slow velocity.

Note 1: Homing procedure does not support simulator.

20.1.1 Example

Example: Single axis homing with HOME_METHOD_33

```
void main()
{
    int axis_id = 0;  //  axis index (Axis Mode)
    int home_method = HOME_METHOD_33;  //  homing method
    double fast_vel = 20;  //  homing velocity (Search for Limit Switch)
    double slow_vel = 2;   //  homing velocity (Search for Index)
    double acc = 2000;  // acceleration time
    double home_offsets = 0;  //  home offset
    int time_out = 10000;  //  time out

    SetHomeMethod(axis_id, home_method);
    SetHomeSwitchVel(axis_id, fast_vel);
    SetHomeZeroVel(axis_id, slow_vel);
    SetHomeAcc(axis_id, acc);
    SetHomeOffset(axis_id, homeoffsets);
    SetHomeTimeout(axis_id, time_out);
    int result = MoveHome(axis_id);
}
```

20.1.2 User-defined homing procedure

Example 1: Search for index signal in positive / negative direction.

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

/* Set parameters */
int axis = 0;
double Home_vel = -20;  //  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);
}
```

Example 2: Search for limit in positive / negative direction first, and search for index signal.

```
//  standard procedure for axis homing --- advanced version
//  (touch probe and limit switch)

/* Set parameters */
int Homing_Type=0;  //  homing method
int axis = 1;  //  Configure the axis
int axis_1 = 2;  //  Configure the axes to a HIMC gantry pair
double Home_vel = -20;  //  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 HIMC 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 HIMC 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;
}
```

20.2 MoveHome

Purpose

To execute homing procedure of an axis.

Syntax

```
int MoveHome(
    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

Users must configure object 0x6060(Mode of operation) and object 0x6061(Mode of operation display) as PDO when using this function.

Requirement

Minimum supported version	iA Studio 3.0

20.3 SetHomeMethod

Purpose

To set the homing method of homing procedure.

Syntax

```
int SetHomeMethod(
    int axis_id,
    int method
);
```

Parameter

axis_id [in] Axis index.

method [in] Description of HMPL definition for homing method index.

Refer to Table 20.1.1 and Table 20.1.2 for details.

The default value is HOME_METHOD_33.

Homing method index	Description of HMPL definition	Homing method index	Description of HMPL definition
1	HOME_METHOD_1	13	HOME_METHOD_13
2	HOME_METHOD_2	14	HOME_METHOD_14
7	HOME_METHOD_7	17	HOME_METHOD_17
8	HOME_METHOD_8	18	HOME_METHOD_18
9	HOME_METHOD_9	19	HOME_METHOD_19
10	HOME_METHOD_10	33	HOME_METHOD_33
11	HOME_METHOD_11	34	HOME_METHOD_34
12	HOME_METHOD_12	37	HOME_METHOD_37

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 2.0

20.4 SetHomeSwitchVel

Purpose

To set fast homing velocity of homing procedure.

Syntax

```
int SetHomeSwitchVel (
    int axis_id,
    double fast_vel
);
```

Parameter

axis_id [in]	Axis index.
fast_vel [in]	Fast homing velocity, the default value is 20. Parameter unit: mm/s or deg/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 3.0

20.5 SetHomeZeroVel

Purpose

To set slow homing velocity of homing procedure.

Syntax

```
int SetHomeZeroVel(
    int axis_id,
    double slow_vel
);
```

Parameter

axis_id [in]	Axis index.
slow_vel [in]	Slow homing velocity, the default value is 5. Parameter unit: mm/s or deg/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 3.0

20.6 SetHomeAcc

Purpose
To set homing acceleration of homing procedure.

Syntax
```
int SetHomeAcc(
    int axis_id,
    double acc
);
```

Parameter
axis_id [in]	Axis index.
acc [in]	Homing acceleration, the default value is 2000. Parameter unit: mm/s or deg/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 3.0

20.7 SetHomeOffset

Purpose

To set the home offset of homing procedure.

Syntax

```
int SetHomeOffset(
    int axis_id,
    double offset
);
```

Parameter

axis_id [in]	Axis index.
offset [in]	Home offset. The default value is 0. Parameter unit: mm or deg

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 2.0

20.8 SetHomeTimeout

Purpose

To set the timeout of homing procedure.

Syntax

```
int SetHomeTimeout(
    int axis_id,
    int timeout
);
```

Parameter

axis_id [in]	Axis index.
timeout [in]	Timeout. The default value is 120,000. Parameter unit: ms

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 2.0

20.9 IsHomed

Purpose

To query whether the axis has completed homing procedure.

Syntax

```
int IsHomed(
    int axis_id
);
```

Parameter

axis_id [in]   Axis index.

Return value

It will return an **int** value **TRUE** (1) if the axis has completed homing procedure. Otherwise, it will return **FALSE** (0).

Requirement

Minimum supported version	iA Studio 3.0

20.10 IsHoming

Purpose

To query whether the axis is operating homing procedure.

Syntax

```
int IsHoming(
    int axis_id
);
```

Parameter

axis_id [in] Axis index.

Return value

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

Requirement

Minimum supported version	iA Studio 3.0

(This page is intentionally left blank.)

21. Communication functions

21.1 Overview

With TCP/IP protocol and network interface, HIMC provides three communication methods, API, Modbus and ASCII. With these methods, HIMC can connect to the related devices or the applications developed by users and perform communication. Under the structure of network socket communication, HIMC can be used as server or client to accept the related devices or the applications to perform communication via HIMC API (Note 1), Modbus (Note 2) and ACSII.

When HIMC is used as server, HMPL provides the functions related to ASCII communication, including Native ASCII and User ASCII. The default connection port of Native ASCII is 3999, and the default connection port of User ASCII is 4000 (Note 3). Native ASCII can support most of the HMPL functions, marked as >_ on the top of each function description in this manual; while User ASCII can use user-defined string interface via user-defined parser (refer to section 21.2.1 START_ASCII_AGENT) to make it more flexible.

When HIMC is used as client, it can connect to other devices. HMPL provides ASCII and Modbus communication functions, ASCII client functions can send ASCII code data to and receive ASCII code data from communication devices; while Modbus client functions can perform read/write operation in the memory blocks of Modbus, including Holding Registers, Input Registers, Coils and Discrete Inputs.

Note 1: Refer to “HIMC API Reference Guide”.

Note 2: Refer to “Modbus TCP User Guide”.

Note 3: Users can change connection port via IP Setting in iA Studio. Refer to section 4.12 in “iA Studio User Guide”.

21.2 ASCII communication

21.2.1 START_ASCII_AGENT

Purpose

Take the controller as server 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”.
}
```

Example 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

21.2.2 ASCII_ServerBroadcast

Purpose

Take the controller as server to send a message to all connected clients.

Syntax

```
void ASCII_ServerBroadcast(
    char *message,
    int   length
);
```

Parameter

message [in] The string of broadcast message.

length [in] The string length of broadcast message. Its maximam value is 128.

Return value

N/A

Remark

Complete the connection of User ASCII before executing this function.

Example

```
void main() {
    char buf[128] = {0};
    //  Write the string to be sent “test” by function StringCopy
    //  \n stands for newline, and \r stands for carriage return.
    StringCopy(buf, "test\n\r");
    int len = StringLen(buf);
    ASCII_ServerBroadcast(buf, len);  //  Send the string command
}
```

Requirement

Minimum supported version	iA Studio 1.4

21.2.3 ASCII_ClientConnect

Purpose

Take the controller as client to build ASCII communication with server.

Syntax

```
int ASCII_ClientConnect(
    char *ip,
    char *port,
    int  *socket_id
);
```

Parameter

ip [in]	The IP address to connect to server.
port [in]	The communication port to connect to server.
socket_id [out]	A pointer to the buffer to receive the socket ID that is successfully connected to server.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "192.168.0.2";
    char port[5] = "1234";
    int  socket_id;
    int  err = ASCII_ClientConnect(ip, port, &socket_id);
    switch (err) {
        case 0:
            Print("Connect Sucess: Sock id %d", socket_id);
            break;
        case 0x17D5:
            Print("Connect Fail: Please check ip & port or already reach limit");
            break;
        case 0x17D4:
            Print("Connect Fail: Connect timeout");
            break;
```

```
        default:
            Print("Connect Fail");
            break;
    }
}
```

Requirement

Minimum supported version	iA Studio 1.4

21.2.4 ASCII_ClientRecv

Purpose

Take the controller as client to receive ASCII data sent from server.

Syntax

```
int ASCII_ClientRecv(
    int  socket_id,
    int  length,
    char *buffer
);
```

Parameter

socket_id [in]	The socket ID to receive ASCII data.
length [in]	The string length to receive ASCII data. Its maximum value is 512.
buffer [out]	A pointer to the buffer to receive the ASCII data to be received.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char buff[200] = {0};
    int  len = 100;
    int  socket_id = 10;
    int  err = ASCII_ClientRecv(socket_id, len, buff);
    switch (err) {
        case 0:
            Print("Recv = %s", buff);
            break;
        case 0x17DF:
            Print("Recv Fail: Can't Recv from this client");
            break;
        case 0x17DE:
            Print("Recv Fail: Timeout");
            break;
```

```
        default:
            Print("Recv Fail");
            break;
    }
}
```

Requirement

Minimum supported version	iA Studio 1.4

21.2.5 ASCII_ClientSend

Purpose

Take the controller as client to send ASCII data to server.

Syntax

```
int ASCII_ClientSend(
    int  socket_id,
    int  length,
    char *buffer
);
```

Parameter

socket_id [in]	The socket ID to send ASCII data.
length [in]	The string length to send ASCII data. Its maximum value is 512.
buffer [in]	A pointer to the buffer to store the ASCII data to be sent.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char msg[20] = "Test String";
    int  len = 100;
    int  socket_id = 10;
    int  err = ASCII_ClientSend(socket_id, len, msg);
    switch (err) {
        case 0:
            Print("Send OK = %s", msg);
            break;
        case 0x17E9:
            Print("Send Fail: Can't Send from this client");
            break;
        case 0x17E8:
            Print("Send Fail: Timeout");
            break;
```

```
        default:
            Print("Send Fail");
            break;
    }
}
```

Requirement

Minimum supported version	iA Studio 1.4

21.2.6 ASCII_ClientDisconnect

Purpose

Take the controller as client to end ASCII communication with server.

Syntax

```
int ASCII_ClientDisconnect(
    int socket_id
);
```

Parameter

socket_id [in] The socket ID to end the communication.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "192.168.0.2";
    char port[5] = "1234";
    int socket_id;
    int err = ASCII_ClientConnect(ip, port, &socket_id);
    if (err) {
        Print("Connect Fail");
        return;
    }
    //  Do something: Read/Write by function ASCII_ClientRecv and ASCII_ClientSend
    err = ASCII_ClientDisconnect(socket_id);
    if (err == 0x17D8)
        Print("Disconnect Timeout");
}
```

Requirement

Minimum supported version	iA Studio 1.4

21.3 Modbus communication

21.3.1 Modbus_ClientConnect

Purpose

Take the controller as client to build Modbus communication with server.

Syntax

```
int Modbus_ClientConnect(
    char *ip,
    int  *socket_id
);
```

Parameter

ip [in]	The IP address to connect to server.
socket_id [out]	A pointer to the buffer to receive the socket ID that is successfully connected to server.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "169.254.188.17";
    int  socket_id;
    int  err = Modbus_ClientConnect(ip, &socket_id);
    switch (err) {
        case 0:
            Print("Connect Sucess: Sock id %d", socket_id);
            break;
        case 0x1839:
            Print("Connect Fail: Please check ip or already reach limit");
            break;
        case 0x1838:
            Print("Connect Fail: Connect timeout");
            break;
```

```
        default:
            Print("Connect Fail");
            break;
    }
}
```

Requirement

Minimum supported version	iA Studio 2.0

21.3.2 Modbus_ClientDisconnect

Purpose

Take the controller as client to end Modbus communication with server.

Syntax

```
int Modbus_ClientDisconnect(
    int socket_id
);
```

Parameter

socket_id [in] The socket ID to end the communication.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "169.254.188.17";
    int socket_id;
    int err = Modbus_ClientConnect(ip, &socket_id);
    if (err) {
        Print("Connect Fail");
        return;
    }
    //  Do something: Perform Read/Write operation with Modbus related functions
    err = Modbus_ClientDisconnect(socket_id);
    if (err == 0x183C)
        Print("Disconnect Timeout");
}
```

Requirement

Minimum supported version	iA Studio 2.0

21.3.3 Modbus_ClientRead_HoldReg

Purpose

Take the controller as client to read Modbus Holding Registers data of server.

Syntax

```
int Modbus_ClientRead_HoldReg(
    int socket_id,
    uint16_t start_addr,
    uint16_t num_regs,
    uint8_t *output_buf,
    int *use_length
);
```

Parameter

socket_id [in]	The socket ID to read Holding Registers data.
start_addr [in]	The start address of the Holding Registers data to be read.
num_regs [in]	The number of the Holding Registers data to be read. Its maximum value is 125.
output_buf [out]	A pointer to the buffer to store the Holding Registers data to be read.
use_length [out]	A pointer to the buffer to store the used length of "output_buf".

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "169.254.188.17";
    int  socket_id;
    int  err = Modbus_ClientConnect(ip, &socket_id);
    //  Wait for client to connect to server
    Sleep(5000);
    if (!err) {
        uint8_t bufs[512];
        int len = 0;
        int addr = 30;
        int regs = 1;
```

```
        err = Modbus_ClientRead_HoldReg(socket_id, addr, regs, bufs, &len);
        if(!err){
            for(int i = 0; i < len; i++){
                Print("%x", bufs[i]);
            }
        }
    }
}
```

Requirement

Minimum supported version	iA Studio 2.0

21.3.4 Modbus_ClientRead_InputReg

Purpose

Take the controller as client to read Modbus Input Registers data of server.

Syntax

```
int Modbus_ClientRead_InputReg(
    int socket_id,
    uint16_t start_addr,
    uint16_t num_regs,
    uint8_t *output_buf,
    int *use_length
);
```

Parameter

socket_id [in]	The socket ID to read Input Registers data.
start_addr [in]	The start address of the Input Registers data to be read.
num_regs [in]	The number of the Input Registers data to be read. Its maximum value is 125.
output_buf [out]	A pointer to the buffer to store the Input Registers data to be read.
use_length [out]	A pointer to the buffer to store the used length of "output_buf".

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "169.254.188.17";
    int  socket_id;
    int  err = Modbus_ClientConnect(ip, &socket_id);
    //  Wait for client to connect to server
    Sleep(5000);
    if (!err) {
        uint8_t bufs[512];
        int len = 0;
        int addr = 30;
        int regs = 1;
```

```
        err = Modbus_ClientRead_InputReg(socket_id, addr, regs, bufs, &len);
        if(!err){
            for(int i = 0; i < len; i++){
                Print("%x", bufs[i]);
            }
        }
    }
}
```

Requirement

Minimum supported version	iA Studio 2.0

21.3.5 Modbus_ClientRead_Coils

Purpose

Take the controller as client to read Modbus Coils data of server.

Syntax

```
int Modbus_ClientRead_Coils(
    int socket_id,
    uint16_t start_addr,
    uint16_t num_coils,
    uint8_t *output_buf,
    int *use_length
);
```

Parameter

socket_id [in]	The socket ID to read Coils data.
start_addr [in]	The start address of the Coils data to be read.
num_coils [in]	The number of the Coils data to be read. Its maximum value is 2000.
output_buf [out]	A pointer to the buffer to store the Coils data to be read.
use_length [out]	A pointer to the buffer to store the used length of “output_buf”.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "169.254.188.17";
    int  socket_id;
    int  err = Modbus_ClientConnect(ip, &socket_id);
    //  Wait for client to connect to server
    Sleep(5000);
    if (!err) {
        uint8_t bufs[512];
        int len = 0;
        int addr = 30;
        int coils = 40;
```

```
        err = Modbus_ClientRead_Coils(socket_id, addr, coils, bufs, &len);
        if(!err){
            for(int i = 0; i < len; i++){
                Print("%x", bufs[i]);
            }
        }
    }
}
```

Requirement

Minimum supported version	iA Studio 2.0

21.3.6 Modbus_ClientRead_Inputs

Purpose

Take the controller as client to read Modbus Discrete Inputs data of server.

Syntax

```
int Modbus_ClientRead_Inputs(
    int socket_id,
    uint16_t start_addr,
    uint16_t num_inputs,
    uint8_t *output_buf,
    int *use_length
);
```

Parameter

socket_id [in]	The socket ID to read Discrete Inputs data.
start_addr [in]	The start address of the Discrete Inputs data to be read.
num_inputs [in]	The number of the Discrete Inputs data to be read. Its maximum value is 2000.
output_buf [out]	A pointer to the buffer to store the Discrete Inputs data to be read.
use_length [out]	A pointer to the buffer to store the used length of “output_buf”.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "169.254.188.17";
    int  socket_id;
    int  err = Modbus_ClientConnect(ip, &socket_id);
    //  Wait for client to connect to server
    Sleep(5000);
    if (!err) {
        uint8_t bufs[512];
        int len = 0;
        int addr = 30;
        int inputs = 40;
```

```
        err = Modbus_ClientRead_Inputs(socket_id, addr, inputs, bufs, &len);
        if(!err){
            for(int i = 0; i < len; i++){
                Print("%x", bufs[i]);
            }
        }
    }
}
```

Requirement

Minimum supported version	iA Studio 2.0

21.3.7 Modbus_ClientWrite_HoldReg

Purpose

Take the controller as client to write Modbus Holding Registers data to server.

Syntax

```
int Modbus_ClientWrite_HoldReg(
    int socket_id,
    uint16_t start_addr,
    uint16_t num_regs,
    uint16_t *write_data,
    uint8_t *output_buf,
    int *use_length
);
```

Parameter

socket_id [in]	The socket ID to write Holding Registers data.
start_addr [in]	The start address of the Holding Registers data to be written.
num_regs [in]	The number of the Holding Registers data to be written. Its maximum value is 123.
write_data [in]	A pointer to the buffer to store the Holding Registers data to be written.
output_buf [out]	A pointer to the buffer to store the Holding Registers data replied by server.
use_length [out]	A pointer to the buffer to store the used length of "output_buf".

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "169.254.188.17";
    int  socket_id;
    int  err = Modbus_ClientConnect(ip, &socket_id);
    //  Wait for client to connect to server
    Sleep(5000);
    if (!err) {
        uint8_t bufs[512];
        uint16_t data[1];
```

```
        //  Enable axis 0 in HIMC
        data[0] = 1;
        int len = 0;
        uint16_t addr = 30;
        uint16_t regs = 1;
        err = Modbus_ClientWrite_HoldReg(socket_id, addr, regs, data, bufs, &len);
        if(!err){
            for(int i = 0; i < len; i++){
                Print("%x", bufs[i]);
            }
        }
    }
}
```

Requirement

Minimum supported version	iA Studio 2.0

21.3.8 Modbus_ClientWrite_Coils

Purpose

Take the controller as client to write Modbus Coils data to server.

Syntax

```
int Modbus_ClientWrite_Coils(
    int socket_id,
    uint16_t start_addr,
    uint16_t num_coils,
    uint16_t *write_data,
    uint8_t *output_buf,
    int *use_length
);
```

Parameter

socket_id [in]	The socket ID to write Coils data.
start_addr [in]	The start address of the Coils data to be written.
num_coils [in]	The number of the Coils data to be written. Its maximum value is 1968.
write_data [in]	A pointer to the buffer to store the Coils data to be written.
output_buf [out]	A pointer to the buffer to store the Coils data replied by server.
use_length [out]	A pointer to the buffer to store the used length of “output_buf”.

Return value

It will return an **int** value **0** if the function succeeds, an error code of the controller (refer to section 18.1.1 for the definition) if the function fails.

Example

```
void main() {
    char ip[15] = "169.254.188.17";
    int  socket_id;
    int  err = Modbus_ClientConnect(ip, &socket_id);
    //  Wait for client to connect to server
    Sleep(5000);
    if (!err) {
        uint8_t bufs[512];
        uint16_t data[1];
```

```
        data[0] = 15;
        int len = 0;
        uint16_t addr = 30;
        uint16_t coils = 4;
        err = Modbus_ClientWrite_Coils(socket_id, addr, coils, data, bufs, &len);
        if(!err){
            for(int i = 0; i < len; i++){
                Print("%x", bufs[i]);
            }
        }
    }
}
```

Requirement

Minimum supported version	iA Studio 2.0

22. Appendix

22.1 Math constants

Table 22.1.1

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.	0.707106781186547524401

22.2 System variables

Table 22.2.1

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 11.6 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.

22.3 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.)