

**PCI-8254/58 Function Library V1.2
AMP-204/8C Function Library V1.2
Build Date 2020.04.27**

Contents

PCI-8254/58 Function Library V1.2	1
AMP-204/8C Function Library V1.2	1
Build Date 2020.04.27	1
Introduction	4
1. Programming Library	7
2. List of all functions	8
3. System and Initialization	17
4. Motion IO and motion status	41
5. Single axis motion	67
6. Jog move	77
7. Interpolation	79
8. Advanced single move & interpolation	89
9. Multi-axes move trigger & stop	141
10. Interrupt	147
11. Sampling	163
12. DIO & AIO	186
13. Point table motion	195
14. Advanced point table	211
15. Compare trigger	255
16. Program download	281
17. Gear / Gantry functions	290
18. Watch dog timer	292
19. VAO/PWM functions (Laser function)	298
20. Circular limit functions	322
21. Manual pulser operation functions	325
22. Pitch error compensation functions	328
23. Position latch functions	335
24. Backlash functions	344
25. Board Parameter Table	347
26. Axis Parameter Table	349
27. Sampling parameters table	356
1. Sampling parameter table	356
2. Sampling source table	356
28. Motion IO status and motion status definitions	359

1.	Motion IO status table	359
2.	Motion status table.....	359
29.	Interrupt factor table	361
30.	Trigger parameter table.....	363
31.	Latch parameter table	368
32.	Device information table	369
33.	VAO parameter table	370
	APS Functions Return Code	371

Introduction

APS means “Automation Product Software”. APS library provides users a uniform interface to access all of ADLINK products which support it. It can cover many automation fields especially in machine automation. The most important component in machine automation is motion control. APS library was first born with motion control which co-working components such as system platform management, field bus communication function, general digital input/output, general analog input/output and various counter/timer supports are all built-in components in APS. The APS library will be an all-in-one solution in automation field of ADLINK products.

The benefits of using this library are

- a) Hardware independent
- b) OS independent
- c) Programming style consistent

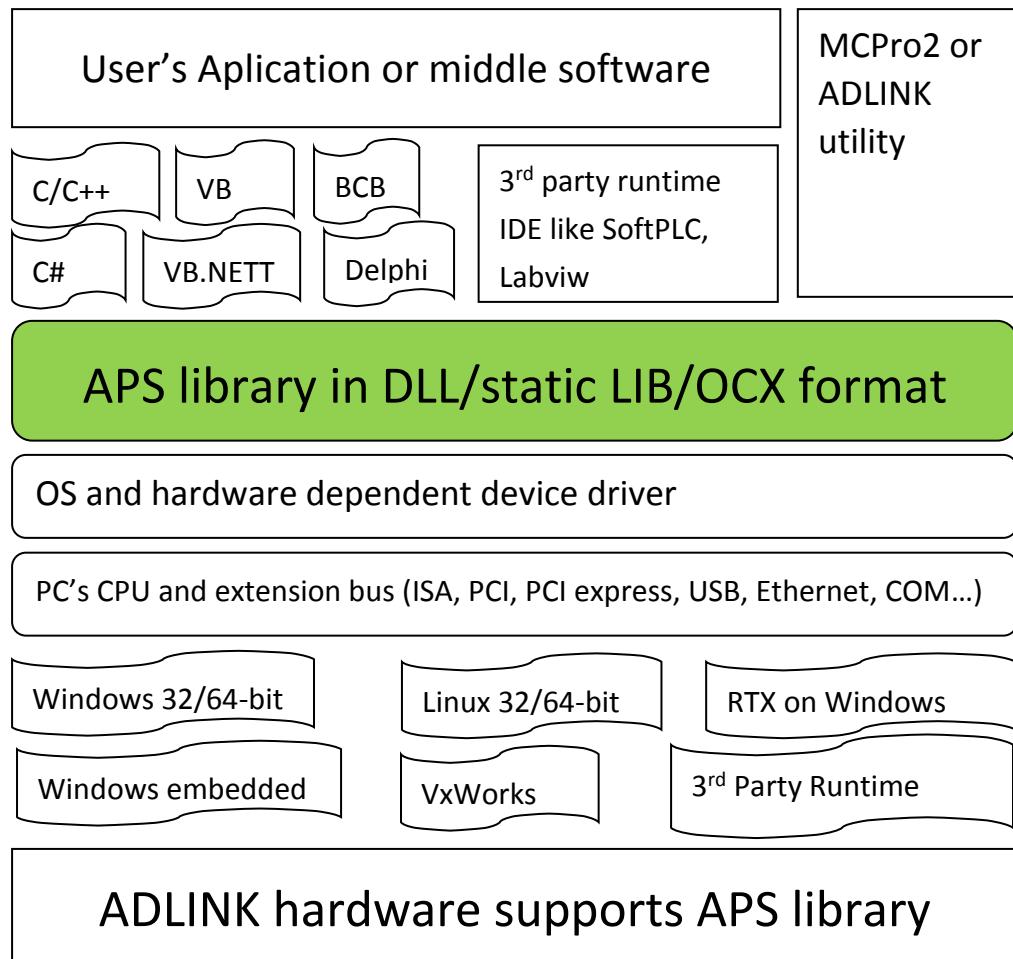
The first benefit is hardware independent. In the past, each product has its own software function set. Every time users want to add or remove different kinds of product even for the same purpose, they must re-program their software to fit it. Most of time, they must re-study new function usage. That's a big effort to users in development and maintenance. It's also not easy to achieve on time development. Now, if users use APS library, they can take APS library as their middle layer of software. It is easy to re-use their own software component which is interfacing with APS without taking care different kinds of same purpose product. That's the meaning of hardware independent.

The second benefit is OS independent. We will continuously research and develop new operating system supports. The standard package of APS supports Microsoft Windows series like Windows XP/2000/Vista and coming new Windows OS. No matter it is 32-bit or 64-bit and no matter platform is single core or multi-cores (SMP), it guarantees all functions running in every OS identically so users don't need to worry about it. It saves much time for users to focus on their machine design. For non-Windows OS, APS also has plan to support it. It will support not only general OS like Linux, and DOS but also real-time OS like RTX, VxWorks and so on. This benefit can help users on product positioning from low-end to high-end machine.

The third benefit is programming style consistent. APS library makes different type of applications like motion control, I/O control and communication to have the same programming style. No matter the motor is stepper or servo, no matter it is distributed or centralized topology, APS library has the same style in programming and also in parameters definitions. APS library also provides various programming language interface and examples for users like ANSI C/C++, Microsoft Visual C/C++, Visual Basic, C#, Visual Basic.NET and Borland Delphi, C/C++ builder and so on. It satisfies different users and purposes on machine development. APS library also provides a visual user interface under Windows system to test all functions of product. This software is based on APS library. In other words, any product supports APS library, the utility also supports them. The utility is called "MotionCreatorPro2" or newer version. It is good to software programmer and system setup people because users don't even need to write any code before verifying the control results and hardware function. It is a good way from product testing to system development and debug.

APS library is not only a library. It is a total package ADLINK wants to provide. It includes various kinds of OS device drivers, dynamic or static link library, many kinds of programming language interface, visualization utility, version control information, rich document, long time support and one-step installation software. It supports most of ADLINK automation products especially in machine control field. By using this library, users can reduce development time and no worry about PC's CPU and operating system changes.

The following diagram is about APS library's position.



1. Programming Library

APS supports many kinds of programming language. The header file of APS library contains function declarations, type definitions and constant variable definitions. The following is the example of C/C++ library. Others please refer to installed header file of corresponding languages.

The function prototype and some common data type are declared in **APS168.h**. We suggest you to use these data types in your application programs for compatibility. The following table shows the data type's name and the numeric range.

Type Name	C/C++ Data types	Description	Range
U8	unsigned char	8-bit ASCII character	0 to 255
I16	Short	16-bit signed integer	-32768 to 32767
U16	unsigned short	16-bit unsigned integer	0 to 65535
I32	long	32-bit signed long integer	-2147483648 to 2147483647
U32	unsigned long	32-bit unsigned long integer	0 to 4294967295
F32	Float	32-bit single-precision floating-point	-3.402823E38 to 3.402823E38
F64	double	64-bit double-precision floating-point	-1.797683134862315E308 to 1.797683134862315E309
Boolean	Char	Boolean logic value	TRUE, FALSE

The naming rule of APS library is full-name of purpose.

In a 'C' programming environment:

APS_{purpose_name}.

e.g. **APS_initial()**, **APS_get_position()**, **APS_relative_move()**

2. List of all functions

All functions List

Sec.	Function name	Descriptions	Page
<u>System & Initialization</u>			
3	APS_initial	Device initialization	
	APS_close	Device close	
	APS_version	Get the version of the library	
	APS_device_driver_version	Get the driver's version of devices	
	APS_get_axis_info	Get the information of the specified axis	
	APS_get_system_timer	Get system timer counter	
	APS_set_board_param	Set board parameter	
	APS_get_board_param	Get board parameter	
	APS_set_axis_param	Set axis parameter	
	APS_get_axis_param	Get axis parameter	
	APS_set_axis_param_f	Set axis parameter by double	
	APS_get_axis_param_f	Set axis parameter by double	
	APS_get_device_info	Get device information	
	APS_get_card_name	Get card name	
	APS_disable_device	Disable cards	
	APS_get_first_axisId	Get first axis id of the card	
	APS_save_parameter_to_flash	Save system & axes parameters to flash	
	APS_load_parameter_from_flash	Load system & axes parameters from flash	
	APS_load_parameter_from_default	Load system & axes parameters by default value	
	APS_load_param_from_file	Load parameters from file	
	APS_get_curr_sys_ctrl_mode	Get current system control mode	
<u>Motion IO and motion status</u>			
4	APS_motion_status	Return motion status	
	APS_motion_io_status	Return motion IO status	
	APS_set_servo_on	Set servo ON/OFF	
	APS_get_position	Get feedback position	
	APS_set_position	Set feedback position	
	APS_get_command	Get command position	
	APS_set_command	Set command position	
	APS_get_command_velocity	Get command velocity	

	APS_get_feedback_velocity	Get feedback velocity	
	APS_get_error_position	Get error position	
	APS_get_target_position	Get target position	
	APS_get_position_f	Get feedback position by double	
	APS_set_position_f	Set feedback position by double	
	APS_get_command_f	Get command position by double	
	APS_set_command_f	Set command position by double	
	APS_get_command_velocity_f	Get command velocity by double	
	APS_get_feedback_velocity_f	Get feedback velocity by double	
	APS_get_error_position_f	Get error position by double	
	APS_get_target_position_f	Get target position by double	
	APS_get_mq_free_space	Get free space of motion queue	
	APS_get_mq_usage	Get usage of motion queue	
	APS_get_stop_code	Get stop code	
	APS_get_encoder	Get raw feedback counter	
	APS_get_command_counter	Get raw command counter	
	APS_get_axis_latch_data	Get ORG/EZ latch data	
5	<u>Single axis motion</u>		
	APS_relative_move	Begin a relative distance move	
	APS_absolute_move	Begin a absolute position move	
	APS_velocity_move	Begin a velocity move	
	APS_home_move	Begin a home move	
	APS_stop_move	Stop move	
6	<u>Jog move</u>		
	APS_jog_start	Start / stop jog move	
7	<u>Interpolation</u>		
	APS_absolute_linear_move	Begin an absolute position linear interpolation	
	APS_relative_linear_move	Begin a relative distance linear interpolation	
	APS_absolute_arc_move	Begin an absolute position circular interpolation	
	APS_relative_arc_move	Begin a relative distance circular interpolation	
8	<u>Advanced single move & interpolation</u>		
	APS_ptp	Begin a single move	
	APS_ptp_v	Begin a single move with Vm profile	
	APS_ptp_all	Begin a single move with all profile	

	APS_vel	Begin a velocity move	
	APS_vel_all	Begin a velocity move with all profile	
	APS_line	Begin a line move	
	APS_line_v	Begin a line move with Vm profile	
	APS_line_all	Begin a line move with all profile	
	APS_arc2_ca	Begin an Arc2 move of angle type	
	APS_arc2_ca_v	Begin an Arc2 move of angle type with Vm profile	
	APS_arc2_ca_all	Begin an Arc2 move of angle type with all profile	
	APS_arc2_ce	Begin an Arc2 move of end position	
	APS_arc2_ce_v	Begin an Arc2 move of end position with Vm profile	
	APS_arc2_ce_all	Begin an Arc2 move of end position with all profile	
	APS_arc3_ca	Begin an Arc3 move of angle type	
	APS_arc3_ca_v	Begin an Arc3 move of angle type with Vm profile	
	APS_arc3_ca_all	Begin an Arc3 move of angle type with all profile	
	APS_arc3_ce	Begin an Arc3 move of end position	
	APS_arc3_ce_v	Begin an Arc3 move of end position with Vm profile	
	APS_arc3_ce_all	Begin an Arc3 move of end position with all profile	
	APS_spiral_ca	Begin a 3D spiral-helix move of angle type	
	APS_spiral_ca_v	Begin a 3D spiral-helix move of angle type with Vm profile	
	APS_spiral_ca_all	Begin a 3D spiral-helix move of angle type with all profile	
	APS_spiral_ce	Begin a 3D spiral-helix move of end position	
	APS_spiral_ce_v	Begin a 3D spiral-helix move of end position with Vm profile	
	APS_spiral_ce_all	Begin a 3D spiral-helix move of end position with all profile	
Multi-axes move trigger & stop			
9	APS_move_trigger	Send a trigger to sync all waiting moves	
	APS_stop_move_multi	Multi-axes stop move	
	APS_emg_stop_multi	Multi-axes emg stop move	
10	Interrupt		

	APS_int_enable	Interrupt main switch	
	APS_set_int_factor	Enable/Disable interrupt factor and get interrupt handle.	
	APS_get_int_factor	Get interrupt factor enable or disable	
	APS_wait_single_int	Wait single interrupt event	
	APS_wait_multiple_int	Wait multiple interrupt events	
	APS_reset_int	Reset interrupt event to non-signaled state.	
	APS_set_int	Set interrupt event to signaled state.	
	APS_set_int_factorH	Enable/Disable interrupt factor and get interrupt handle.(Win32)	
	APS_int_no_to_handle	Convert interrupt event number to interrupt handle.(Win32)	
11	<u>Sampling</u>		
	APS_set_sampling_param	Set sampling parameter.	
	APS_get_sampling_param	Get sampling parameter.	
	APS_wait_trigger_sampling	Waiting for sample data.	
	APS_wait_trigger_sampling_async	Waiting for sample data asynchronously	
	APS_get_sampling_count	Get sampled data count.	
	APS_stop_wait_sampling	Force stop wait sampling	
	APS_auto_sampling	Start/Stop auto sampling	
	APS_get_sampling_data	Get sampling data in auto sampling mode by 4 Channels.	
	APS_set_sampling_param_ex	Set sampling parameter by structure. It is an extension to 8 channels.	
	APS_get_sampling_param_ex	Get sampling parameter by structure. It is an extension to 8 channels.	
	APS_wait_trigger_sampling_ex	Waiting for sample data. It is an extension to 8 channels.	
	APS_wait_trigger_sampling_async_ex	Waiting for sample data asynchronously. It is an extension to 8 channels.	
	APS_get_sampling_data_ex	Get sampling data in auto sampling mode. It is an extension to 8 channels.	
12	<u>DIO & AIO</u>		
	APS_write_d_output	Set digital output value	
	APS_read_d_output	Read digital output value	
	APS_read_d_input	Read digital input value	
	APS_write_d_channel_output	Set digital output value by channel	
	APS_read_d_channel_output	Read digital output value by channel	
	APS_read_a_input_value(*1)	Read back analog input value by volt	

	APS_write_a_output_value(*1)	Set analog output value by volt	
13	<u>Point table motion</u>		
	APS_set_feeder_group	Set axes into a feeder group	
	APS_get_feeder_group	Return the configuration in one feeder group	
	APS_free_feeder_group	Free a feeder group and it's resources	
	APS_reset_feeder_buffer	Reset the feeder's point buffer	
	APS_set_feeder_point_2D	Add a point into feeder's buffer	
	APS_start_feeder_move	Start point table move and feed points.	
	APS_get_feeder_status	Get feeder status	
	APS_get_feeder_running_index	Get which point is in operation.	
	APS_get_feeder_feed_index	Get which point is set into point table.	
	APS_set_feeder_ex_pause	Motion paused(stopped) and feeder paused	
	APS_set_feeder_ex_rollback	Move back to the starting position of paused index	
	APS_set_feeder_ex_resume	Resume the point-table move	
14	<u>Advanced Point table</u>		
	APS_pt_enable	Enable point table.	
	APS_pt_disable	Disable point table.	
	APS_get_pt_info	Get information of point table.	
	APS_pt_set_vs	Set configuration of Vs to point table	
	APS_pt_get_vs	Get configuration of Vs in the point table	
	APS_pt_start	Set control command to point table	
	APS_get_pt_status	Get status of point table	
	APS_reset_pt_buffer	Reset buffer of point table	
	APS_pt_roll_back	Rollback to previous point	
	APS_get_pt_error	Get error code of point table	
	APS_pt_dwell	Push a dwell move into point buffer of point table.	
	APS_pt_line	Push a line move into point buffer of point table.	
	APS_pt_arc2_ca	Push a 2d arc move into point buffer of point table.	
	APS_pt_arc2_ce	Push a 2d arc move into point buffer of point table.	
	APS_pt_arc3_ca	Push a 3d arc move into point buffer of point table.	
	APS_pt_arc3_ce	Push a 3d arc move into point buffer of point table.	

15	APS_pt_spiral_ca	Push a helical move into point buffer of point table.	
	APS_pt_spiral_ce	Push a helical move into point buffer of point table.	
	APS_pt_ext_set_do_ch	Set Do extension command into command buffer. Command buffer is active when pushing a move into point table.	
	APS_pt_set_absolute	Set absolute profile into profile buffer.	
	APS_pt_set_relative	Set relative profile into profile buffer.	
	APS_pt_set_trans_buffered	Set transition to buffer mode in profile buffer.	
	APS_pt_set_trans_inp	Set transition to in-position mode in profile buffer.	
	APS_pt_set_trans_blend_dec	Set transition to blending mode with deceleration in profile buffer.	
	APS_pt_set_trans_blend_dist	Set transition to blending mode with residue distant in profile buffer.	
	APS_pt_set_trans_blend_pcnt	Set transition to blending mode with residue distant percetange in profile buffer.	
	APS_pt_set_acc	Set accerlation profile into profile buffer.	
	APS_pt_set_dec	Set deceleration profile into profile buffer.	
	APS_pt_set_acc_dec	Set accerlation / deceleration profile into profile buffer	
	APS_pt_set_s	Set S-factor profile into profile buffer.	
	APS_pt_set_vm	Set maximum velocity profile into profile buffer.	
	APS_pt_set_ve	Set end velocity profile into profile buffer.	
<u>Compare trigger</u>			
15	APS_set_trigger_param	Set compare trigger related parameter	
	APS_get_trigger_param	Get compare trigger related parameter	
	APS_set_trigger_linear	Set linear comparing function	
	APS_set_trigger_table	Set table comparing function	
	APS_set_trigger_manual	Manual output trigger	
	APS_set_trigger_manual_s	Manual output trigger synchronously	
	APS_get_trigger_table_cmp	Get current table comparing value	
	APS_get_trigger_linear_cmp	Get current linear comparing value	
	APS_get_trigger_count	Get triggered count	
	APS_reset_trigger_count	Reset triggered count	

	APS_get_timer_counter	Get timer count	
	APS_set_timer_counter	Set timer count	
	APS_set_multi_trigger_table	Set table comparing function	
	APS_get_multi_trigger_table_cmp	Get current table comparing value	
	APS_set_trigger_table_data	Set table comparator data (Fast table compare trigger function)	
	APS_get_trigger_table_status	Get table comparator status (Fast table compare trigger function)	
	APS_get_trigger_cmp_value	Get table comparator value (Fast table compare trigger function)	
	APS_enable_trigger_table	Enable table comparator (Fast table compare trigger function)	
	APS_reset_trigger_table	Reset table comparator (Fast table compare trigger function)	
16	<u>Program download(*1)</u>		
	APS_load_vmc_program	Load VMC file to task memory	
	APS_save_vmc_program	Save to VMC file from task memory	
	APS_set_task_mode	Set task run mode	
	APS_get_task_mode	Get task run mode	
	APS_start_task	Start task control command	
	APS_get_task_info	Get task information	
	APS_get_task_msg	Get message of all tasks	
17	<u>Gear / Gantry functions</u>		
	APS_start_gear	Enable/Disable a specified gear mode	
	APS_get_gear_status	Get gear status	
18	<u>Watch dog timer</u>		
	APS_wdt_start	Start / Stop watch dog timer	
	APS_wdt_get_timeout_period	Get a timeout period of watch dog timer	
	APS_wdt_reset_counter	Reset counter of watch dog timer	
	APS_wdt_get_counter	Get counter of watch dog timer	
	APS_wdt_set_action_event	Set action event of watch dog timer	
	APS_wdt_get_action_event	Get action event of watch dog timer	
19	<u>VAO/PWM functions(Laser function)</u>		
	APS_set_vao_param	Set parameter to VAO table	

	APS_get_vao_param	Get parameter of VAO table	
	APS_set_vao_table	Set VAO table	
	APS_switch_vao_table	Switch to specified VAO table	
	APS_start_vao	Enable VAO output channel	
	APS_get_vao_status	Get VAO status	
	APS_check_vao_param	Check parameters setting of specified VAO table	
	APS_set_vao_param_ex	Set table parameters via VAO structure	
	APS_get_vao_param_ex	Get table parameters via VAO structure	
	APS_set_pwm_on	Start to output PWM signal	
	APS_set_pwm_width	Set pulse width to a PWM channel	
	APS_set_pwm_frequency	Set pulse frequency to a PWM channel	
	APS_get_pwm_width	Get pulse width from a PWM channel	
	APS_get_pwm_frequency	Get pulse frequency from a PWM channel	
20	<u>Circular limit functions</u>		
	APS_set_circular_limit	Set circular limit configurations	
	APS_get_circular_limit	Get circular limit configurations	
21	<u>Manual pulser operation functions</u>		
	APS_manual_pulser_start	Start manual pulser operation	
	APS_manual_pulser_velocity_m ove	Start velocity move in manual pulser operation	
22	<u>Pitch error compensation functions</u>		
	APS_set_pitch_table	Set configurations and data of pitch error compensation table	
	APS_get_pitch_table	Get configurations and data of pitch error compensation table	
	APS_start_pitch_comp	Start pitch error compensation	
23	<u>Position latch functions</u>		
	APS_enable_ltc_fifo	Enable position latch process	
	APS_get_ltc_fifo_point	Get latch point array	

	APS_set_ltc_fifo_param	Set latch parameter value	
	APS_get_ltc_fifo_param	Get latch parameter value	
	APS_reset_ltc_fifo	Reset latch queue and fifo	
	APS_get_ltc_fifo_usage	Get latch queue used space	
	APS_get_ltc_fifo_free_space	Get latch queue free space	
	APS_get_ltc_fifo_status	Get latch queue and fifo status	
24	<u>Backlash functions</u>		
	APS_set_backlash_en	Enable/Disable backlash	
	<u>Table definition</u>		
	<u>Board parameters definition table</u>		
	<u>Axis parameters definition table</u>		
	<u>Sampling parameters table</u>		
	<u>Sampling source table</u>		
	<u>Motion IO status table</u>		
	<u>Motion status table</u>		
	<u>Interrupt factor table</u>		
	<u>Trigger parameter table</u>		
	<u>Latch parameter table</u>		
	<u>Device information table</u>		
	<u>VAO parameter table</u>		
	<u>Function Return Code</u>		

*1. AMP series don't support this feature.

3. System and Initialization

1	APS_initial	Device initialization
---	-------------	-----------------------

Descriptions:

This function is used to initialize all products on local controller supported by APS function library. It allocates system hardware resources for each board including I/O address, memory address, IRQ and DMA if needed. It retrieves a board ID for each board which is assigned by on-board switch or operating system. The board ID is a unique number for the board in the system. It is used for any other APS functions to access corresponding hardware.

If users choose on-board switch (Mode = manual ID) initial mode and there are some boards don't support this feature in the system, the board ID of these boards will be arranged after the boards having on-board switch automatically. The card ID (dip-switch) cannot be set the same when you used "manual-ID" or the function will return error.

Syntax:

C/C++:

```
I32 APS_initial(I32 *BoardID_InBits, I32 Mode);
```

Visual Basic:

```
APS_initial (BoardID_InBits As Long, ByVal Mode As Long) As Long
```

Parameters:

I32 * BoardID_InBits: Card ID information in bit format.

Example: If the value of BoardID_InBits is 0x11 which means that there are 2 cards in your system and those card's ID are 0 and 4.

I32 Mode:

Bit 0	Enable the On board dip switch (SW1) to decide the Card ID. [0:By system assigned, 1:By dip switch]
Bit 1	Parallel type axis indexing mode 0 : auto mode (default) 1 : fixed mode
Bit 2	Serial type axis indexing mode 0: auto mode (default) 1: fixed mode
Bit 4	Option of load system & axes parameters method.

Bit 5	(00B) 0: Do nothing, parameters keep current value. (01B) 1: Load from default (02B) 2: Load from flash(*1)
Bit 9	Option to select behavior of MDN bit of motion status. 0: MDN turns on when CSTP or ASTP occurs. (default) 1: MDN turns on when CSTP occurs.
Others	Reserved (set it to 0)

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret; // return value
I32 BoardID_InBits;
I32 Mode = 0; //By system assigned
ret = APS_initial( &BoardID_InBits, Mode);

...// Do something
ret = APS_close(); //Close all cards in the system
```

See also:

[APS_close\(\)](#), [APS_get_axis_info\(\)](#);

2	APS_close	Devices close
---	-----------	---------------

Descriptions:

This function is used to close all resources allocated by APS library. The resources include system hardware resource like I/O address, memory address, IRQ and DMA. It also deletes some objects, handles or memory allocated by APS library.

Syntax:

C/C++:

I32 APS_close()

Visual Basic:

APS_close() As Long

Parameters:

No parameter.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret; // return value
I32 BoardID_InBits;
I32 Mode = 0; //By system assigned
ret = APS_initial( &BoardID_InBits, Mode);

...// Do something
ret = APS_close(); //Close all cards in the system
```

See also:

APS_initial();

APS_version	Get the version of the library
-------------	--------------------------------

Descriptions :

This function is used to get APS library (DLL) version information.

Syntax:

C/C++:

```
I32 APS_version();
```

Visual Basic:

```
APS_version() As Long
```

Parameters:

No Parameters

Return Values:

Return library (DLL) version.

Example:

```
I32 version;
```

```
version = APS_version();
```

See also:

APS_device_driver_version	Get the driver's version of devices
---------------------------	-------------------------------------

Descriptions :

This function is used to get device driver version information of one board. The version information is the same for one type of board in system.

Syntax:

C/C++

I32 APS_device_driver_version(I32 Board_ID)

Visual Basic:

APS_device_driver_version(ByVal Board_ID As Long) As Long

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

Return Values:

Positive value: The device driver version number.

Negative value: Error code: Please refer to error code table.

Example:

I32 version;

```
//Get device driver version of board 0
version = APS_device_driver_version( 0 );
```

See also:

APS_get_axis_info	Get the information of the specified axis
-------------------	---

Descriptions:

This function is used to get information of one axis. The information includes attached board ID, serial port ID, serial module ID and module type. There are two categories for axis ID index: parallel and serial types. PCI-8254/58 is parallel type axis. MNET-4XMO-© and HSL-4XMO are serial type axis.

The parallel type axis ID indexing rule is according to the board ID. The formula is:

Axis ID = Board ID x Maximum number of axis within one board + Axis No

The Axis No parameter is the axis number within the board. The Maximum number of axis within one board parameter is an inside system variable of APS library. If APS system is running under auto mode, the value depends on board type. If APS system is running under fixed mode, the default value is 16. If the system has some boards without axes, it still counts the formula when indexing under fixed mode. Fixed mode is useful for users to remove/add some boards from system without rearranging axis index.

For example, a user has two boards: PCI-8258 and PCI-8253.

	PCI-8258 (ID=0), 8-axis	PCI-8253 (ID=1), 3-axis
Auto Mode	Axis ID ranges 0~7	Axis ID ranges 8~10
Fixed Mode	Axis ID ranges 0~15	Axis ID ranges 16~31

If the board ID is not continuous,

	PCI-8258 (ID=0), 8-axis	PCI-8253 (ID=2), 3-axis
Auto Mode	Axis ID ranges 0~7	Axis ID ranges 8~10
Fixed Mode	Axis ID ranges 0~15	Axis ID ranges 32~47

The serial type axis ID indexing rule is according to the module ID and assigned with a starting axis ID first. The formula of serial port axis would be:

Axis ID = Module ID x Maximum number of axis within one module + Starting Axis ID of port + Axis No

The Axis No parameter is the axis number within the module. The Maximum number of axis within this module parameter is an inside port variable of APS library. If APS field bus system is running under auto mode, the value depends on module type. If APS field bus system is running under fixed mode, the default value is 4. Starting Axis ID of port parameter is the starting axis ID of one port assigned by users when field bus starts. The default value is 0. In fixed mode, if the port has some modules without axis, it still counts the formula when indexing. Fixed mode is useful for users to remove/add some modules from system without rearranging axis index of other modules

For example, a user has 2 MNET modules on PCI-7856 with board ID=

	MNET-J3 (ID=0)	MNET-4XMO (ID=1), 4-axis
Auto Mode	Axis ID ranges 0 only	Axis ID ranges 1~4

Fixed Mode	Axis ID ranges 0~3	Axis ID ranges 4~7
------------	--------------------	--------------------

If the module ID is not continuous,

	MNET-J3 (ID=0)	MNET-4XMO (ID=2), 4-axis
Auto Mode	Axis ID ranges 0 only	Axis ID ranges 1~4
Fixed Mode	Axis ID ranges 0~3	Axis ID ranges 8~11

Syntax:

C/C++

```
I32 APS_get_axis_info( I32 Axis_ID, I32 *Board_ID, I32 *Axis_No, I32 *Port_ID, I32 *Module_ID );
```

Visual Basic:

```
APS_get_axis_info( ByVal Axis_ID As Long, Board_ID As Long, Axis_No As Long, Port_ID As Long,  
Module_ID As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535

I32 *Board_ID: The returned board ID for the Axis ID. Range is from 0 to 31.

I32 *Axis_No: The axis number within the board. Range is from 0 to maximum number of axis within this module.

I32 *Port_ID: The returned field bus port ID of board for the axis. Range is from 0 to 15. *Port_ID=-1 means no serial port exists.

I32 *Module_ID: The returned module ID of port for the axis. Range is from 0~65535. *Module_ID=-1 means no serial port exists.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 Axis_ID = 0;
```

```
I32 Board_ID, Axis_No, Port_ID, Module_ID;
```

```
//According to axis id, get related information
```

```
APS_get_axis_info( Axis_ID, & Board_ID, &Axis_No, &Port_ID, &Module_ID );
```

See also:

[APS_initial\(\)](#)

APS_get_system_timer	Get system timer counter
----------------------	--------------------------

Descriptions:

This function is used to get system timer counter. The counter will count up every cycle time after system is ready. Users can use this function to check if the system is under control or not.

Syntax:

C/C++:

```
I32 APS_get_system_timer( I32 Board_ID, I32 *Timer );
```

Visual Basic:

```
APS_get_system_timer( ByVal Board_ID As Long, Timer As Long ) As Long
```

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

I32 *Timer: return system timer.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 timer;
```

```
//Get system timer of board id 0
```

```
APS_get_system_timer( 0, &timer );
```

See also:

APS_set_board_param	Set board parameter
---------------------	---------------------

Descriptions:

This function is used to set all kinds of parameter which has relationship with a board. Please refer to the board parameter table for the definition and detail descriptions.

Syntax:

C/C++

I32 APS_set_board_param(I32 Board_ID, I32 BOD_Param_No, I32 BOD_Param);

Visual Basic:

APS_set_board_param (ByVal Board_ID As Long, ByVal BOD_Param_No As Long, ByVal BOD_Param As Long) As Long

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

I32 BOD_Param_No: Board parameter number. Please refer the board parameter table for definition.

I32 BOD_Param: Board parameter value. Refer to the board parameter table for detail.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
//Board id is 0. Set EMG logic to 1.
APS_set_board_param( 0, 0x00, 1 );
```

See also:

[APS_get_board_param\(\)](#)

APS_get_board_param	Get board parameter
---------------------	---------------------

Descriptions:

This function is used to get all kinds of parameter which has relationship with a board. Please refer to the board parameter table for the definition and detail descriptions.

Syntax:

C/C++:

```
I32 APS_get_board_param( I32 Board_ID, I32 BOD_Param_No, I32 *BOD_Param );
```

Visual Basic:

```
APS_get_board_param (ByVal Board_ID As Long, ByVal BOD_Param_No As Long, BOD_Param As Long)
As Long
```

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

I32 BOD_Param_No: Board parameter number. Please refer the board parameter table for definition.

I32 *BOD_Param: The returned board parameter value. Refer to board parameter table.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 paramVal;
```

```
//Board id is 0. Get EMG logic.
```

```
APS_get_board_param( 0, 0x00, & paramVal );
```

See also:

[APS_set_board_param\(\)](#)

APS_set_axis_param	Set axis parameter
--------------------	--------------------

Descriptions:

This function is used to set all kinds of parameter of one axis. The parameters include run mode, acceleration rate, deceleration rate, Jerk, motion I/O logic and so on. Please refer to the axis parameter table for the definition and detail descriptions.

Syntax:

C/C++

```
I32 APS_set_axis_param( I32 Axis_ID, I32 AXS_Param_No, I32 AXS_Param );
```

Visual Basic:

```
APS_set_axis_param( ByVal Axis_ID As Long, ByVal AXS_Param_No As Long, ByVal AXS_Param As Long )
As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 AXS_Param_No: Axis parameter number from 0 to 65535. Each parameter is defined by a unique symbol in 3~6 characters .Refer to axis parameter table.

I32 AXS_Param: Axis parameter value. Refer to axis parameter table

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
//Axis id is 0. Set EL logic to 1
APS_set_axis_param ( 0, 0x00, 1 );
```

See also:

[APS_get_axis_param\(\)](#)

APS_get_axis_param	Get axis parameter
--------------------	--------------------

Descriptions:

This function is used to set all kinds of parameter of one axis. The parameters include run mode, acceleration rate, deceleration rate, Jerk, motion I/O logic and so on. Please refer to the axis parameter table for the definition and detail descriptions.

Syntax:

C/C++:

```
I32 APS_get_axis_param( I32 Axis_ID, I32 AXS_Param_No, I32 *AXS_Param );
```

Visual Basic:

```
APS_get_axis_param (ByVal Axis_ID As Long, ByVal AXS_Param_No As Long, AXS_Param As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 AXS_Param_No: Axis parameter number from 0 to 65535. Each parameter is defined by a unique symbol in 3~6 characters .Refer to axis parameter table.

I32 *AXS_Param: Axis parameter value. Refer to axis parameter table

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 paramVal;  
//Axis id is 0. Get EL logic  
APS_get_axis_param ( 0, 0x00, &paramVal );
```

See also:

[APS_set_axis_param\(\)](#);

APS_set_axis_param_f	Set axis parameter by double
----------------------	------------------------------

Descriptions:

This function is used to set parameters by double. Those double parameters include acceleration rate, deceleration rate, Jerk and so on. Please refer to the axis parameter table for the definition and detailed descriptions.

Syntax:

C/C++

```
I32 APS_set_axis_param_f( I32 Axis_ID, I32 AXS_Param_No, F64 AXS_Param );
```

Visual Basic:

```
APS_set_axis_param_f(ByVal Axis_ID As Long, ByVal AXS_Param_No As Long, ByVal AXS_Param As  
Double) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 AXS_Param_No: Axis parameter number from 0 to 65535. Each parameter is defined by a unique symbol in 3~6 characters .Refer to axis parameter table.

F64 AXS_Param: Axis parameter value. (F64 type) Refer to axis parameter table.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
//Axis id is 0. Set acceleration to 100000.0  
APS_set_axis_param_f ( 0, 0x13, 100000.0 );
```

See also:

[APS_get_axis_param_f\(\)](#),[APS_set_axis_param\(\)](#), [APS_get_axis_param\(\)](#)

APS_get_axis_param_f	Get axis parameter by double
----------------------	------------------------------

Descriptions:

This function is used to get axis parameters by float. The double parameters include acceleration rate, deceleration rate, Jerk and so on. Please refer to the axis parameter table for the definition and detailed descriptions.

Syntax:

C/C++:

```
I32 APS_get_axis_param_f( I32 Axis_ID, I32 AXS_Param_No, F64 *AXS_Param );
```

Visual Basic:

```
APS_get_axis_param_f(ByVal Axis_ID As Long, ByVal AXS_Param_No As Long, AXS_Param As Double)
As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 AXS_Param_No: Axis parameter number from 0 to 65535. Each parameter is defined by a unique symbol in 3~6 characters .Refer to axis parameter table.

F64 *AXS_Param: Axis parameter value. (F64 type) Refer to axis parameter table

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
F64 paramVal;
```

```
//Axis id is 0. Get acceleration
```

```
APS_get_axis_param_f ( 0, 0x13, &paramVal );
```

See also:

[APS_set_axis_param_f\(\)](#),[APS_set_axis_param\(\)](#), [APS_get_axis_param\(\)](#)

APS_get_card_name	Get card name
-------------------	---------------

Descriptions:

This function is used to get card name. After executing APS_initial(), user could get each board's name by passing specified board id.

Syntax:

C/C++:

```
I32 APS_get_card_name ( I32 Board_ID, I32 *CardName );
```

Visual Basic:

```
APS_get_card_name (ByVal Board_ID As Long, CardName As Long) As Long
```

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

I32 * CardName: Board's name of specified board id.

```
0: PCI_8392,  1: PCI_825x,  2: PCI_8154,      3: PCI_785X
4: PCI_8158,  5: PCI_7856,  6: ISA_DPAC1000, 7: ISA_DPAC3000
8: PCI_8144,  9: PCI_8258,  10: PCI_8102,     11: PCI_V8258
12: PCI_V8254, 13: PCI_8158A, 14: PCI_20408C
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 CardName;
//Board id is 0. Get board's name
APS_get_card_name ( 0, & CardName);
```

See also:

APS_disable_device	Disable specified device. It is used to ignore the disabling device during initialization.
--------------------	--

Descriptions :

This function could disable specified board via its name. It is used to ignore the disabling device during initialization.

Syntax:

C/C++:

I32 APS_disable_device(I32 DeviceName);

Visual Basic:

APS_disable_device(ByVal DeviceName As Long) As Long

Parameters:

I32 DeviceName : Specify a device name.

0: PCI_8392, 1: PCI_825x, 2: PCI_8154, 3: PCI_785X
 4: PCI_8158, 5: PCI_7856, 6: ISA_DPAC1000, 7: ISA_DPAC3000
 8: PCI_8144, 9: PCI_8258, 10: PCI_8102, 11: PCI_V8258

Return Values:

I32 Error code: Please refer to error code table.

Example:

I32 BoardID_InBits;

```
//Disable PCI-8258
APS_disable_device( 9 );
```

```
//Initial all card, but PCI_8258
APS_initial( &BoardID_InBits, 0 );
```

See also:

APS_get_first_axisId	Get first axis id of specified board
----------------------	--------------------------------------

Descriptions:

This function is used to get first axis id of specified board.

Syntax:

C/C++:

```
I32 APS_get_first_axisId( I32 Board_ID, I32 *StartAxisID, I32 *TotalAxisNum );
```

Visual Basic:

```
APS_get_first_axisId (ByVal Board_ID As Long, StartAxisID As Long, TotalAxisNum As Long) As Long
```

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

I32 * StartAxisID: First aixs id of specified board id.

I32 * TotalAxisNum: Total axes on specidied board id

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 StartAxisID;
```

```
I32 TotalAxisNum;
```

//Board id is 0. Get axis info

```
APS_get_first_axisId ( 0, & StartAxisID, & TotalAxisNum );
```

See also:

APS_get_device_info	Get device information
---------------------	------------------------

Descriptions:

This function is used to get specified device (board) information. The information includes driver version, firmware version, PCB version and so on. Refer to [device information table](#).

Syntax:

C/C++

```
I32 APS_get_device_info( I32 Board_ID, I32 Info_No, I32 *Info );
```

Visual Basic:

```
APS_get_device_info( ByVal Board_ID As Long, ByVal Info_No As Long, Info As Long ) As Long
```

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

I32 Info_No: Reference to device information table.

I32 *Info: Reference to device information table.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
```

```
I32 Info;
```

```
//Board id is 0. Get fpga version
```

```
APS_get_device_info( 0, 0x21, &Info );
```

See also:

APS_save_parameter_to_flash	Save system parameters & axes parameters to flash
-----------------------------	---

Descriptions:

This function is used to save system parameters and axes parameters to flash.

Syntax:

C/C++:

```
I32 APS_save_parameter_to_flash( I32 Board_ID );
```

Visual Basic:

```
APS_save_parameter_to_flash( ByVal Board_ID As Long)As Long
```

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
ret = APS_save_parameter_to_flash ( 0 );
if( ret == ERR_NoError )
    // Load parameters success.
```

See also:

[APS_load_parameter_from_flash](#); [APS_load_parameter_from_default](#)

APS_load_parameter_from_flash	Load system parameters & axes parameters from flash
-------------------------------	---

Descriptions:

Load system parameters and axes parameters from flash.

Syntax:

C/C++:

I32 APS_load_parameter_from_flash(I32 Board_ID);

Visual Basic:

APS_load_parameter_from_flash(ByVal Board_ID As Long) As Long

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
ret = APS_load_parameter_from_flash ( 0 );
if( ret == ERR_NoError )
    // Load parameters success.
```

See also:

APS_save_parameter_to_flash; APS_load_parameter_from_default

APS_load_parameter_from_default	Load system parameters & axes parameters by default value.
---------------------------------	--

Descriptions:

Load default setting to system parameters & axes parameters.

Syntax:

C/C++:

```
I32 APS_load_parameter_from_default( I32 Board_ID );
```

Visual Basic:

```
APS_load_parameter_from_default( ByVal Board_ID As Long )As Long
```

Parameters:

I32 Board_ID: The Board's ID from 0 to 31.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
ret = APS_load_parameter_from_default( 0 );
if( ret == ERR_NoError )
    // Load parameters success.
```

See also:

APS_save_parameter_to_flash; APS_load_parameter_from_flash;

APS_load_param_from_file	Load parameters from file
--------------------------	---------------------------

Descriptions:

This function is used to load all parameters which are recoded in the input file (XML file).

You can use **Motion creator pro utility** to create or modify a XML files.

This function will process the XML file with following functions.

```
APS_set_axis_param()  
APS_set_board_param()  
APS_set_axis_param_f()
```

When it process an unrecognized parameter or a wrong parameter, the load process will be stopped immediately and return an error. So that the other parameters which after the unrecognized parameter will not be set into the devices. Therefore you must check the file validly before you load into your system.

Syntax:

C/C++:

```
I32 APS_load_param_from_file( const char *pXMLFile );
```

Visual Basic:

```
APS_load_param_from_file( pXMLFile As String ) As Long
```

Parameters:

const char *pXMLFile: Specified a XML file which created by MCPro2.exe.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 Ret;  
I32 BoardID_InBits;  
I32 Mode = 0; //By system assigned  
  
APS_initial( &BoardID_InBits, Mode);  
Ret = APS_load_param_from_file( "C:\\WINDOWS\\system32\\ApsParameters.xml" );  
If( Ret != ERR_NoError )
```

```
{ //Error load parameters from file.}
```

See also:

`APS_set_axis_param(); APS_set_board_param(); APS_set_axis_param_f()`

APS_get_curr_sys_ctrl_mode	Get current system control mode in FPGA
----------------------------	---

Descriptions:

This function is used to get current control mode in FPGA. There are three kinds of control mode. One is pulse mode, another is analog mode and the other is step mode. User could get control mode of specified axis.

Syntax:

C/C++:

```
I32 APS_get_curr_sys_ctrl_mode(I32 Axis_ID, I32 * Mode);
```

Visual Basic:

```
APS_get_curr_sys_ctrl_mode( ByVal Axis_ID As Long, Mode As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 * Mode: Control mode:

0: pulse mode

1: analog mode

2: step mode

Return Values:

I32 Error code: Refer to error code table.

Example:

```
I32 Mode = 0;
```

```
//Get control mode of each axix
```

```
APS_get_curr_sys_ctrl_mode( Axis_ID, &Mode );
```

See also:

4. Motion IO and motion status

APS_motion_status	Return motion status
-------------------	----------------------

Descriptions:

This function is used to get one axis' motion status. The status includes running, normal stop, abnormal stop by reasons, in waiting other axis, follow status, in some modes, in accelerating or decelerating and so on. Status can be more than two such like mode and running. Users need to use this function to check whether the 'Fire-and-forget' function is done in polling system. In even driven system, users can use interrupt event functions.

Please refer to the motion status table for detail description.

Syntax:

C/C++:

```
I32 APS_motion_status( I32 Axis_ID );
```

Visual Basic:

```
APS_motion_status (ByVal Axis_ID As Long) As Long
```

Parameters:

I32 Axis_ID:The Axis ID from 0 to 65535

Return Values:

Positive value:

The value of motion status. Please refer to motion status bit number definition table for the value meaning

Negative value:

Error Code: Please refer to error code table.

Example:

```
I32 MotionStatus;  
MotionStatus = APS_motion_status( Axis_ID ); //Get Motion status  
...
```

See also:

[APS_motion_io_status\(\)](#)

APS_motion_io_status	Return motion IO status
----------------------	-------------------------

Descriptions:

This function is used to get one axis' motion I/O information like ORG, PEL, MEL, SVON, INP and so on.

These statuses are connected to external switched or servo drivers.

Please refer to the [motion IO status table](#) for detail description.

Syntax:

C/C++:

I32 APS_motion_io_status(I32 Axis_ID);

Visual Basic:

APS_motion_io_status (ByVal Axis_ID As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535

Return Values:

Positive value:

The value of motion IO status, please refer to motion IO status bit number definition table for the value meaning

Negative value:

Error Code: Please refers to error code table.

Example:

```
I32 MotionIO;
MotionIO = APS_motion_io_status(Axis_ID ); //Get Motion IO status
```

...

See also:

[APS_motion_status \(\)](#);

APS_set_servo_on	Set servo ON/OFF
------------------	------------------

Descriptions:

This function is used to command servo driver of specified axis to starts controlling its servomotor.

Then motion function could be applied on this axis.

Syntax:

C/C++:

I32 APS_set_servo_on(I32 Axis_ID, I32 Servo_on);

Visual Basic:

APS_set_servo_on (ByVal Axis_ID As Long, ByVal ServoOn As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535

I32 Servo_on:

0: Servo OFF, 1: Servo ON

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
...//Initialization
APS_set_servo_on( Axis_ID, 1 ); // Set servo ON
... //Motion action
APS_set_servo_on(Axis_ID, 0); //Set servo OFF
...//Release
```

See also:

APS_get_position	Get feedback position
------------------	-----------------------

Descriptions:

This function is used to get the position counter of one axis. The counter is in unit of pulse.

Syntax:

C/C++:

```
I32 APS_get_position( I32 Axis_ID, I32 *Position );
```

Visual Basic:

```
APS_get_position (ByVal Axis_ID As Long, Position As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Position: Feedback position. Unit in pulse

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 Position;
APS_get_position(Axis_ID, &Position ); //Get feedback position
...
```

See also:

APS_get_command(); APS_set_position(); APS_set_command()

APS_set_position	Set feedback position
------------------	-----------------------

Descriptions:

This function is used to set the position counter of one axis. The counter is in unit of pulse. It assigns a new position at instance but the motor will not move due to this function.

Syntax:

C/C++:

I32 APS_set_position(I32 Axis_ID, I32 Position);

Visual Basic:

APS_set_position (ByVal Axis_ID As Long, ByVal Position As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Position: Set feedback position. Unit in pulse.

Return Values:

I32 Error code: Please refer to error code table.

Example:

...

APS_set_position(Axis_ID, 0); // Set feedback position to zero

See also:

APS_get_position(); APS_get_command(); APS_set_command()

APS_get_command	Get command position
-----------------	----------------------

Descriptions:

This function is used to get the command counter of one axis. The counter is in unit of pulse.

Syntax:

C/C++:

```
I32 APS_get_command( I32 Axis_ID, I32 *Command );
```

Visual Basic:

```
APS_get_command (ByVal Axis_ID As Long, Command As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Command: Command position. Unit in pulse.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 Command ;
```

```
APS_get_command(Axis_ID, &Command ); //Get command position.
```

```
...//
```

See also:

[APS_get_position\(\)](#); [APS_set_position\(\)](#); [APS_set_command\(\)](#)

APS_set_command	Set command position
-----------------	----------------------

Descriptions:

This function is used to set the command counter of one axis. The counter is in unit of pulse. It assigns a new command counter at instance but the motor will not move due to this function.

Syntax:

C/C++:

I32 APS_set_command(I32 Axis_ID, I32 Command);

Visual Basic:

APS_set_command (ByVal Axis_ID As Long, ByVal Command As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Command: Position command. Unit in pulse.

Return Values:

I32 Error code: Please refer to error code table.

Example:

...//

APS_set_command(Axis_ID, 0); //Set command position to zero.

See also:

APS_get_position(); APS_get_command(); APS_set_position();

APS_get_command_velocity	Get command velocity
--------------------------	----------------------

Descriptions:

This function is used to get command velocity. The minimum value depends on speed calculation resolution of system.

Syntax:

C/C++:

I32 APS_get_command_velocity(I32 Axis_ID, I32 *Velocity);

Visual Basic:

APS_get_command_velocity(ByVal Axis_ID As Long, Velocity As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Velocity: Return command velocity. Unit: pps

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Axis_ID = 0;
I32 Velocity;
ret = APS_get_command_velocity ( Axis_ID, &Velocity);
if( ret == ERR_NoError )
{
    //Velocity
}
```

See also:

APS_get_position(); APS_get_command();APS_get_feedback_velocity()

APS_get_feedback_velocity	Get feedback velocity
---------------------------	-----------------------

Descriptions:

This function is used to get feedback velocity. The minimum value depends on speed calculation resolution of system.

Syntax:

C/C++:

```
I32 APS_get_feedback_velocity(I32 Axis_ID, I32 *Velocity);
```

Visual Basic:

```
APS_get_feedback_velocity(ByVal Axis_ID As Long, Velocity As Long ) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Velocity: Return feedback velocity. Unit: pps

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Axis_ID = 0;
I32 Velocity;
ret = APS_get_feedback_velocity( Axis_ID, &Velocity);
if( ret == ERR_NoError )
{
    //Velocity
}
```

See also:

[APS_get_position\(\)](#); [APS_get_command\(\)](#); [APS_get_command_velocity \(\)](#);

APS_get_error_position	Get error position
------------------------	--------------------

Descriptions:

This function is used to get error position value. This value is defined as command minus feedback position

Syntax:

C/C++:

I32 APS_get_error_position(I32 Axis_ID, I32 *Err_Pos);

Visual Basic:

APS_get_error_position(ByVal Axis_ID As Long, Err_Pos As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Err_Pos: Return error position.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Axis_ID = 0;
I32 Err_Pos;
ret = APS_get_error_position(Axis_ID, &Err_Pos );
if( ret == ERR_NoError )
    //Show error position.
```

See also:

APS_get_position(); APS_get_command(); APS_get_command_velocity()
();APS_get_feedback_velocity()

APS_get_target_position	Get target position
-------------------------	---------------------

Descriptions:

This function is used to get target position record. In linear positioning mode, the value is target position. In circular positioning mode, the value is the same as command. In velocity and jog mode, the value is the same as command.

Syntax:

C/C++:

```
I32 APS_get_target_position( I32 Axis_ID, I32 *Targ_Pos );
```

Visual Basic:

```
APS_get_target_position(ByVal Axis_ID As Long, Targ_Pos As Long ) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Targ_Pos: Return target position.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Axis_ID = 0;
I32 Targ_Pos;
ret = APS_get_target_position(Axis_ID, &Targ_Pos );
if( ret == ERR_NoError )
    //Show target position.
```

See also:

APS_get_position(); APS_get_command(); APS_get_command_velocity()
();APS_get_feedback_velocity()

APS_get_position_f	Get feedback position by double
--------------------	---------------------------------

Descriptions:

This function is used to get the position counter of one axis by double. The counter is in unit of pulse.

Syntax:

C/C++:

```
I32 APS_get_position_f( I32 Axis_ID, F64 *Position );
```

Visual Basic:

```
APS_get_position_f( ByVal Axis_ID As Long, Position As Double ) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

F64 *Position: Feedback position. Unit in pulse

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
F64 Position;
```

```
APS_get_position_f(Axis_ID, &Position ); //Get feedback position
```

```
...
```

See also:

[APS_get_command_f\(\)](#); [APS_set_position_f\(\)](#); [APS_set_command_f\(\)](#)

APS_set_position_f	Set feedback position by double
--------------------	---------------------------------

Descriptions:

This function is used to set the position counter of one axis by double. The counter is in unit of pulse. It assigns a new position at instance but the motor will not move due to this function.

Syntax:

C/C++:

```
I32 APS_set_position_f(I32 Axis_ID, F64 Position);
```

Visual Basic:

```
APS_set_position_f(ById Axis_ID As Long, ByVal Position As Double) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

F64 Position: Set feedback position. Unit in pulse.

Return Values:

I32 Error code: Please refer to error code table.

Example:

...

```
APS_set_position_f(Axis_ID, 0.0 ); // Set feedback position to zero
```

See also:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_set_command_f\(\)](#)

APS_get_command_f	Get command position by double
-------------------	--------------------------------

Support Products: PCI-8258

Descriptions:

This function is used to get the command counter of one axis by double. The counter is in unit of pulse.

Syntax:

C/C++:

```
I32 APS_get_command_f( I32 Axis_ID, F64 *Command );
```

Visual Basic:

```
APS_get_command_f(ByVal Axis_ID As Long, Command As Double) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

F64 *Command: Command position. Unit in pulse.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
F64 Command ;
```

```
APS_get_command_f(Axis_ID, &Command ); //Get command position by double  
...//
```

See also:

APS_get_position_f(); APS_set_position_f(); APS_set_command_f()

APS_set_command_f	Set command position by double
-------------------	--------------------------------

Support Products: PCI-8258

Descriptions:

This function is used to set the command counter of one axis by double. The counter is in unit of pulse. It assigns a new command counter at instance but the motor will not move due to this function.

Syntax:

C/C++:

```
I32 APS_set_command_f(I32 Axis_ID, F64 Command);
```

Visual Basic:

```
APS_set_command_f(ById Axis_ID As Long, ByVal Command Double) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

F64 Command: Position command. Unit in pulse.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
...//
```

```
APS_set_command_f(Axis_ID, 0.0); //Set command position to zero.
```

See also:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_set_position_f\(\)](#);

APS_get_target_position_f	Get target position by double
---------------------------	-------------------------------

Descriptions:

This function is used to get target position record by double. In linear positioning mode, the value is target position. In circular positioning mode, the value is the same as command. In velocity and jog mode, the value is the same as command.

Syntax:

C/C++:

```
I32 APS_get_target_position_f( I32 Axis_ID, F64 *Targ_Pos );
```

Visual Basic:

```
APS_get_target_position_f(ByVal Axis_ID As Long, Targ_Pos As Double ) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

F64 *Targ_Pos: Return target position.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Axis_ID = 0;
F64 Targ_Pos;
ret = APS_get_target_position_f(Axis_ID, &Targ_Pos );
if( ret == ERR_NoError )
    //Show target position.
```

See also:

APS_get_position_f(); APS_get_command_f(); APS_get_command_velocity_f()
();APS_get_feedback_velocityf()

APS_get_error_position_f	Get error position by double
--------------------------	------------------------------

Descriptions:

This function is used to get error position record by double. This value is defined as command minus feedback position.

Syntax:

C/C++:

I32 APS_get_error_position_f(I32 Axis_ID, F64 *Err_Pos);

Visual Basic:

APS_get_error_position_f(ByVal Axis_ID As Long, Err_Pos As Double) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

F64 *Err_Pos: Return error position.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Axis_ID = 0;
F64 Err_Pos;
ret = APS_get_error_position_f(Axis_ID, &Err_Pos );
if( ret == ERR_NoError )
    //Show error position.
```

See also:

APS_get_position_f(); APS_get_command_f(); APS_get_command_velocity_f()
();APS_get_feedback_velocityf();APS_get_target_position_f

APS_get_command_velocity_f	Get command velocity by double
----------------------------	--------------------------------

Descriptions:

This function is used to get command velocity by double. The minimum value depends on speed calculation resolution of system.

Syntax:

C/C++:

I32 APS_get_command_velocity_f(I32 Axis_ID, F64 *Velocity);

Visual Basic:

APS_get_command_velocity_f(ByVal Axis_ID As Long, Velocity As Double) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

F64 *Velocity: Return command velocity. Unit: pps

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Axis_ID = 0;
F64 Velocity;
ret = APS_get_command_velocity_f ( Axis_ID, &Velocity);
if( ret == ERR_NoError )
{
    //Velocity
}
```

See also:

APS_get_position_f(); APS_get_command_f();APS_get_feedback_velocityf()

APS_get_feedback_velocity_f	Get feedback velocity by double
-----------------------------	---------------------------------

Descriptions:

This function is used to get feedback velocity by double. The minimum value depends on speed calculation resolution of system.

Syntax:

C/C++:

```
I32 APS_get_feedback_velocity_f(I32 Axis_ID, F64 *Velocity );
```

Visual Basic:

```
APS_get_feedback_velocity_f( ByVal Axis_ID As Long, Velocity As Double ) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

F64 *Velocity: Return feedback velocity. Unit: pps

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Axis_ID = 0;
F64 Velocity;
ret = APS_get_feedback_velocity_f ( Axis_ID, &Velocity);
if( ret == ERR_NoError )
{}
```

See also:

[APS_get_position_f\(\)](#); [APS_get_command_f\(\)](#); [APS_get_command_velocityf\(\)](#)

APS_get_mq_free_space	Get current free space of motion queue
-----------------------	--

Descriptions:

This function is used to get current free space of motion queue.

Each axis has own motion queue (FIFO) to buffer the motion commands.

Syntax:

C/C++:

```
I32 APS_get_mq_free_space( I32 Axis_ID, I32 *Sapce );
```

Visual Basic:

```
APS_get_mq_free_sapce (ByVal Axis_ID As Long, Sapce As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Sapce: Free space of motion queue of specified axis.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 Space;
```

```
APS_get_mq_free_space(Axis_ID, &Space ); //Get free space of motion queue
```

```
...//
```

See also:

[APS_get_mq_usage\(\);](#)

APS_get_mq_usage	Get current usage from motion queue
------------------	-------------------------------------

Descriptions:

This function is used to get current usage from motion queue.

Each axis has own motion queue (FIFO) to buffer the motion commands.

Syntax:

C/C++:

```
I32 APS_get_mq_usage( I32 Axis_ID, I32 *Usage );
```

Visual Basic:

```
APS_get_mq_usage(ByVal Axis_ID As Long, Usage As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Usage: The usage of motion queue of specified axis.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 Usage;
```

```
APS_get_mq_free_space(Axis_ID, &Usage ); //Get usage of motion queue
```

```
...//
```

See also:

[APS_get_mq_free_space\(\)](#);

APS_get_stop_code	Get stop code by axis
-------------------	-----------------------

Descriptions:

This function is used to get stop code. The stop code is a stop reason for an axis when a move is stopping.

The possible stop code is shown in following table. If axis performs a PTP move of single axis and stops normally, you will get a stop code belong to STOP_NORMAL.

Symbol	Code	Description
STOP_NORMAL	0	Stop normally
STOP_EMG	1	Stop when EMG is turn ON
STOP_ALM	2	Stop when ALM is turn ON
STOP_SVNO	3	Stop when servo is turn-OFF
STOP_PEL	4	Stop by PEL signal turn ON
STOP_MEL	5	Stop by MEL signal turn ON
STOP_SPEL	6	Stop by soft-limit condition – positive end limit
STOP_SMEL	7	Stop by soft-limit condition – minus end limit
STOP_USER_EMG	8	EMG stop by user
STOP_USER	9	Stop by user
STOP_GAN_L1	10	Stop by E-Gear gantry protect level 1 condition is met.
STOP_GAN_L2	11	Stop by E-Gear gantry protect level 2 condition is met
STOP_GEAR_SLAVE	12	Stop because gear slave axis
STOP_ERROR_LEVEL	13	Error position check level
STOP_DI	14	DI

Note: If motion is multi-axes motion (Interpolation), the stop code is only updated on the reference axis. Other axes keep previous stop code.

Syntax:

C/C++:

```
I32 APS_get_stop_code( I32 Axis_ID, I32 *Code );
```

Visual Basic:

```
APS_get_stop_code(ByVal Axis_ID As Long, Code As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 * Code: Stop code (stop reason).

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 Code;  
I32 Axis_ID;
```

```
APS_get_stop_code(Axis_ID, &Code ); //Get stop code  
...//
```

See also:

APS_get_encoder	Get raw encoder counter
-----------------	-------------------------

Descriptions:

This function is used to get raw encoder counter of one axis. It is read only and applied to debug or monitor raw counter.

Syntax:

C/C++:

I32 APS_get_encoder(I32 Axis_ID, I32 *Encoder);

Visual Basic:

APS_get_encoder(ByVal Axis_ID As Long, Encoder As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Encoder: Raw encoder counter. Unit in pulse.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 Encoder;
APS_get_encoder(Axis_ID, &Encoder ); //Get raw encoder counter.
...//
```

See also:

APS_get_command_counter	Get raw command counter
-------------------------	-------------------------

Descriptions:

This function is used to get raw command counter of one axis. It is read only and applied to debug or monitor raw counter.

Syntax:

C/C++:

I32 APS_get_command_counter(I32 Axis_ID, I32 *Counter);

Visual Basic:

APS_get_command_counter(ByVal Axis_ID As Long, Counter As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Counter: Raw command counter. Unit in pulse.

Return Values:

I32 Error code: Please refer to error code table.

Example:

I32 Counter;

```
APS_get_command_counter(Axis_ID, & Counter ); //Get raw command counter
...//
```

See also:

APS_get_axis_latch_data	Get ORG/EZ latch data
-------------------------	-----------------------

Descriptions:

Users can use this function to get latch data from the input signal belongs to an axis, such has ORG and EZ. This function's behavior for getting latch data is read clear which means when users call this function, the latched data will be retrieved and latch counter will be clear to zero for next latch. When users know there is a data latched, users can call this function to get the most update latch data from motor encoder.

Syntax:

C/C++:

```
I32 APS_get_axis_latch_data(I32 Axis_ID, I32 latch_channel, I32 *latch_data)
```

Visual Basic:

```
APS_get_axis_latch_data( ByVal Axis_ID As Long, ByVal latch_data As Long, latch_data As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to maximum axis number in one system.

I32 latch_channel: for channel selection: 0 for ORG and 1 for EZ

I32 *latch_data: a data pointer to get motor encoder latch data

Return Values:

I32 Error code: Please refer to error code table.

Example:

See also:

5. Single axis motion

APS_relative_move	Begin a relative distance move
-------------------	--------------------------------

Descriptions:

This function is used to start a single axis relative motion. Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed.

This command can't be overridden by other motion modes like Jog, home, manual pulse generation, contour motion. Users must stop axis motion before switching to those modes mentioned above.

Syntax:

C/C++:

```
I32 APS_relative_move( I32 Axis_ID, I32 Distance, I32 Max_Speed );
```

Visual Basic:

```
APS_relative_move (ByVal Axis_ID As Long, ByVal Distance As Long, ByVal Max_Speed As Long) As  
Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Distance: Relative distance. Unit is pulse.

I32 Max_Speed: The maximum speed of this move profile. Unit: pulse/sec.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
APS_set_axis_param(Axis_ID, PRA_ACC, 1000000 ); //Set acceleration rate  
APS_set_axis_param(Axis_ID, PRA_DEC, 1000000 ); //Set deceleration rate  
//Execute a relative move.  
APS_relative_move( Axis_ID, 10000, 10000 );
```

See also:

`APS_relative_move(); APS_absolute_move(); APS_velocity_move(); APS_home_move();`
`APS_stop_move(); APS_emg_stop();`

APS_absolute_move	Begin a absolute position move
-------------------	--------------------------------

Descriptions:

This function is used to start a single axis absolute positioning motion. Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed.

This command can't be overridden by other motion modes like Jog, home, manual pulse generation, contour motion. Users must stop axis motion before switching to those modes mentioned above.

Syntax:

C/C++:

```
I32 APS_absolute_move( I32 Axis_ID, I32 Position, I32 Max_Speed );
```

Visual Basic:

```
APS_absolute_move(ByVal Axis_ID As Long, ByVal Position As Long, ByVal Max_Speed As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Position: Absolute command position. Unit is pulse.

I32 Max_Speed: The maximum speed of this move profile. Unit: pulse/sec

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
APS_set_axis_param(Axis_ID, PRA_ACC, 1000000 ); //Set acceleration rate
APS_set_axis_param(Axis_ID, PRA_DEC, 1000000 ); //Set deceleration rate
//Execute an absolute move
APS_absolute_move( Axis_ID, 10000, 10000 );
```

See also:

APS_relative_move(); APS_absolute_move(); APS_velocity_move(); APS_home_move();
APS_stop_move(); APS_emg_stop();

APS_velocity_move	Begin a velocity move
-------------------	-----------------------

Descriptions:

This function is used to start a velocity move. The axis will stop when users issue stop move command.

The speed profile's acceleration and deceleration rate and curve are set by axis parameter function.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done after axis is stopped by command or abnormal situation.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed

This command can't be overridden by other motion modes like Jog, home, manual pulse generation, contour motion. Users must stop axis motion before switching to those modes mentioned above.

The velocity move is one kind of positioning control. The controller will try to make feedback position to catch up command position. That means if the axis is stopped, the controller will control axis's position to command because it is in position closed loop mode.

Syntax:

C/C++:

```
I32 APS_velocity_move( I32 Axis_ID, I32 Max_Speed );
```

Visual Basic:

```
APS_velocity_move (ByVal Axis_ID As Long, ByVal Max_Speed As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Max_Speed: The maximum speed of this move profile. Unit: pulse/sec

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
APS_set_axis_param(Axis_ID, PRA_ACC, 1000000 ); //Set acceleration rate
APS_set_axis_param(Axis_ID, PRA_DEC, 1000000 ); //Set deceleration rate
APS_velocity_move(Axis_ID, Max_Speed ); //Start velocity move
...
APS_stop_move(Axis_ID); //Stop velocity move
```

See also:

APS_relative_move(); APS_absolute_move(); APS_velocity_move(); APS_home_move();
APS_stop_move(); APS_emg_stop();

APS_home_move	Begin a home move
---------------	-------------------

Descriptions:

This function is used to start a HOME (ORG or DOG) position of the axis. There are several modes which can be selected by axis parameter setting functions. After it is done, the position of the axis will be renew base on the physical location of HOME.

This function is ‘fire-and-forget’ type. That means user’s program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

Users needn’t to write a home sequence to accomplish homing. All the sequences are controlled inside the board without CPU resource.

Syntax:

C/C++:

```
I32 APS_home_move( I32 Axis_ID );
```

Visual Basic:

```
APS_home_move (ByVal Axis_ID As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

Return Values:

I32 Error code: Please refer to error code table.

Example1:

```
//Set homing parameters
APS_set_axis_param( Axis_ID, PRA_HOME_MODE, 0 ); //Set home mode
APS_set_axis_param( Axis_ID, PRA_HOME_DIR, 1 ); //Set home direction
APS_set_axis_param_f( Axis_ID, PRA_HOME_CURVE, 0.5 ); //Set s-factor to 0.5
APS_set_axis_param_f( Axis_ID, PRA_HOME_ACC, 100000.0 ); //Set homing acceleration rate
APS_set_axis_param_f( Axis_ID, PRA_HOME_VM, 2000000.0 ); //Set homing maximum velocity.
APS_set_axis_param_f( Axis_ID, PRA_HOME_VO, 200000.0 ); //Set homing leave home velocity

APS_home_move(Axis_ID ); //Start homing
...//Check homing done(Motion done)
```

See also:

```
APS_set_axis_param(); APS_get_axis_param(); APS_stop_move(); APS_emg_stop();
```

APS_stop_move	Stop move
---------------	-----------

Descriptions:

This function is used to stop single or multiple axes motion at once. It can stop single axis homing, positioning and speed moving. It also can stop multiple axes interpolation motion when users place one of axis ID which is relative to interpolation moving. The deceleration profile is set by axis parameter function which is different from normal deceleration setting. The deceleration parameter is different from normal move profile. It can be set individually.

The stop function can't be overridden by other functions.

Syntax:

C/C++:

I32 APS_stop_move(I32 Axis_ID);

Visual Basic:

APS_stop_move (ByVal Axis_ID As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

Return Values:

I32 Error code: Refer to error code table.

Example:

```
APS_absolute_move(Axis_ID, Position, Max_Speed );
...
APS_stop_move(Axis_ID); //Stop move
```

See also:

APS_emg_stop();

APS_emg_stop	Emergency stop
--------------	----------------

Descriptions:

This function is used to stop single or multiple axes motion immediately. It can stop single axis homing, positioning and speed moving. It also can stop multiple axes interpolation motion when users place one of axis ID which is relative to interpolation moving. Because the stop function will stop axis accidentally, it will generate an abnormal stop interrupt event rather than normal stop event if interrupt factor is set. The motion status will also be set to an abnormal stop status. The abnormal stop status or event will be clear by next motion command. This function has no deceleration profile.

Syntax:

C/C++:

I32 APS_emg_stop(I32 Axis_ID);

Visual Basic:

APS_emg_stop (ByVal Axis_ID As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

Return Values:

I32 Error code: Refer to error code table.

Example:

```
APS_absolute_move(Axis_ID, Position, Max_Speed );
...
APS_emg_stop (Axis_ID); //EMG stop
```

See also:

APS_stop_move()

6. Jog move

APS_jog_start	Start / stop jog move
---------------	-----------------------

Descriptions:

This function is used to start / stop a jog move. Before start a jog move, you must enable the axis to jog mode.

Following parameters are shown to configure Jog parameter: The details refer to axis parameter table.

Symbol define	ParamNo	Description
PRA_JG_MODE	0x40	(I32) Jog mode [0:Continuous mode, 1:Step mode]
PRA_JG_DIR	0x41	(I32) Jog move direction [0: Positive direction, 1: Negative direction]
PRA_JG_SF	0x42	(F64) Jog S factor [0 ~ 1]
PRA_JG_ACC	0x43	(F64) Jog move acceleration [value > 0]
PRA_JG_DEC	0x44	(F64) Jog move deceleration [value > 0]
PRA_JG_VM	0x45	(F64) Jog move max velocity [value > 0]
PRA_JG_OFFSET	0x46	(F64) Jog offset, for step mode [value >= 0]
PRA_JG_DELAY	0x47	(I32) Jog delay, for step mode, microsecond [0 ~ 10,000,000]
PRA_JG_MAP_DI_EN	0x48	(I32) Enable Digital input map to jog command signal
PRA_JG_P_JOG_DI	0x49	(I32) Mapping configuration for positive jog and digital input.
PRA_JG_N_JOG_DI	0x4A	(I32) Mapping configuration for negative jog and digital input.
PRA_JG_JOG_DI	0x4B	(I32) Mapping configuration for jog and digital input.

Syntax:

C/C++:

```
I32 APS_jog_start( I32 Axis_ID, I32 STA_On );
```

Visual Basic:

```
APS_jog_start( ByVal Axis_ID As Long, ByVal STA_On As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 STA_On: 1:STA signal on, 0:STA signal off.

Return Values:

I32 Error code: Refer to error code table.

Example:

```
// Configure jog move parameter.  
APS_set_axis_param( Axis_ID, PRA_JG_MODE, 0 ); //Set to continuous mode  
APS_set_axis_param( Axis_ID, PRA_JG_DIR, 1 ); //Set jog to negative direction  
APS_set_axis_param_f( Axis_ID, PRA_JG_ACC, 100000.0 ); //Set jog move acceleration  
  
// perform jog move ... (APS_jog_start)  
APS_jog_start( Axis_ID, 1 ); //STA signal ON  
...  
APS_jog_start( Axis_ID, 0 ); //STA signal OFF
```

See also:

APS_set_axis_param(); APS_get_axis_param(); APS_set_axis_param_f(); APS_get_axis_param_f();

7. Interpolation

APS_absolute_linear_move	Begin a absolute position linear interpolation
--------------------------	--

Descriptions:

This function is used to start an absolute linear interpolation positioning motion. Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. Because the speed parameter is in vector direction, this function will take the master axis's acceleration and deceleration time constant to calculate. The master axis is the minimum axis number that user perform an interpolation.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed.

The overridden command must have the same dimension and axis ID of previous one. These two commands can't be overridden by other motion modes like home operation. Users must stop axis motion before switching to those modes mentioned above.

Note: The axes specified in Axis_ID_Array must be of the same card.

Syntax:

C/C++:

```
I32 APS_absolute_linear_move( I32 Dimension, I32 *Axis_ID_Array, I32 *Position_Array, I32  
Max_Linear_Speed );
```

Visual Basic:

```
APS_absolute_linear_move( ByVal Dimension As Long, Axis_ID_Array As Long, Position_Array As Long,  
ByVal Max_Linear_Speed As Long ) As Long
```

Parameters:

I32 Dimension: The dimension of interpolation axes. (2~6 axes)

I32 *Axis_ID_Array: The axis ID array from 0 to 65535.

I32 *Position_Array: Absolute position array. (unit: pulse)

I32 Max_Linear_Speed: Maximum linear interpolation speed (unit: pulse/sec)

Return Values:

I32 Error code: Refer to error code table.

Example:

```
//...Initial card
```

```
I32 Dimension = 4;
```

```
I32 Master_Axis_ID = 1; //Master axis
```

```
I32 Axis_ID_Array[4] = { 1, 2, 3, 4}; //Axis ID 1 is master axis.
```

```
I32 Position_Array [4] = {10000, 20000, 30000, 40000 };
```

```
I32 Max_Linear_Speed = 10000;
```

```
I32 Ret;
```

```
APS_set_axis_param_f( Master_Axis_ID, PRA_CURVE, 0.5 ); //Set S-factor
```

```
APS_set_axis_param_f( Master_Axis_ID, PRA_ACC, 100000.0 ); //Set acceleration
```

```
APS_set_axis_param_f( Master_Axis_ID, PRA_DEC, 100000.0 ); //Set deceleration
```

```
Ret = APS_absolute_linear_move ( Dimension, Axis_ID_Array, Position_Array, Max_Linear_Speed );
```

```
...
```

See also:

[APS_relative_linear_move](#)

APS_relative_linear_move	Begin a relative distance linear interpolation
--------------------------	--

Descriptions:

This function is used to start a relative linear interpolation positioning motion. Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. Because the speed parameter is in vector direction, this function will take the master axis's acceleration and deceleration time constant to calculate. The master axis is the minimum axis number that user perform an interpolation.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed.

The overridden command must have the same dimension and axis ID of previous one. These two commands can't be overridden by other motion modes like home operation. Users must stop axis motion before switching to those modes mentioned above.

Note: The axes specified in Axis_ID_Array must be of the same card.

Syntax:

C/C++:

```
I32 APS_relative_linear_move( I32 Dimension, I32 *Axis_ID_Array, I32 *Distance_Array, I32  
Max_Linear_Speed );
```

Visual Basic:

```
APS_relative_linear_move( ByVal Dimension As Long,  Axis_ID_Array As Long, Distance_Array As  
Long, ByVal Max_Linear_Speed As Long) As Long
```

Parameters:

I32 Dimension: The dimension of interpolation axes. (2~4 axes)

I32 *Axis_ID_Array: The Axis ID array from 0 to 65535.

I32 *Distance_Array: Relative distance array. (unit: pulse)

I32 Max_Linear_Speed: Maximum linear interpolation speed (unit: pulse/sec)

Return Values:

I32 Error code: Refer to error code table.

Example:

```
//...Initial card
I32 Dimension = 4;
I32 Master_Axis_ID = 0;
I32 Axis_ID_Array[4] = {0, 1, 2, 3}; //Axis ID 0 is master axis.
I32 Distance_Array[4] = {10000, 20000, 30000, 40000 };
I32 Max_Linear_Speed = 10000;
I32 Ret;

APS_set_axis_param_f( Master_Axis_ID, PRA_CURVE, 0.5 ); //Set S-curve
APS_set_axis_param_f( Master_Axis_ID, PRA_ACC, 100000.0 ); //Set acceleration
APS_set_axis_param_f( Master_Axis_ID, PRA_DEC, 100000.0 ); //Set deceleration

Ret = APS_relative_linear_move( Dimension, Axis_ID_Array, Distance_Array, Max_Linear_Speed );
...

```

See also:

[APS_relative_linear_move\(\)](#), [APS_set_axis_param\(\)](#),

APS_absolute_arc_move	Begin an absolute position circular interpolation
-----------------------	---

Descriptions:

This function is used to start an absolute circular interpolation positioning motion. User must specify absolute center position and traveling angle for circular interpolation. The speed profile's acceleration and deceleration rate are set by axis parameter function. The following axis parameter should be setting before you calling this function.

The details of parameters please refer the axis parameter table.

Symbol define	ParamNo	Description
PRA_SF	0x20	Move S-factor
PRA_ACC	0x21	Acceleration rate
PRA_DEC	0x22	Deceleration rate
PRA_VS	0x23	Start velocity
PRA_VM	0x24	Maximum velocity
PRA_VE	0x25	End velocity

Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. Because the speed parameter is in vector direction (Tangent to the circular), this function will take the master axis's acceleration and deceleration time constant to calculate. The master axis is the minimum axis number that user perform an interpolation. For example, Axis ID 2 and 3 are performing a circular interpolation. The Axis ID 2 is the master axis.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done. The motion status "circular interpolation signal (CIP)" of each axis performing a circular interpolation will be turn on when command is started and will be turned off at command is finished. If circular interpolation is stop normally, the normal stop signal (NSTP) will be turned on. On the contrary, if circular interpolation is stopped abnormally (such as ALM, EMG and so on is turned on), abnormal stop signal (ASTP) will be turned on.

During the axis traveling, users can start a new move command including stop command to override the previous one (The dimension and Axis_ID_Array must be the same). The axis will be switched to new command immediately according to new setting of target center position, new speed profile.

This command can't be overridden by other motion modes like home operation. Users must stop axis motion before switching to those modes mentioned above.

Note: The 2 axes specified in Axis_ID_Array must be of the same card.

Syntax:

C/C++:

```
I32 APS_absolute_arc_move( I32 Dimension, I32 *Axis_ID_Array, I32 *Center_Pos_Array, I32  
Max_Arc_Speed, I32 Angle );
```

Visual Basic:

```
APS_absolute_arc_move( ByVal Dimension As Long, Axis_ID_Array As Long, Center_Pos_Array As  
Long, ByVal Max_Arc_Speed As Long, ByVal Angle As Long )As Long
```

Parameters:

I32 Dimension: The dimension of interpolation axes. (The maximum dimensions refer to product specification)

I32 *Axis_ID_Array: The Axis ID array from 0 to 65535.

I32 *Center_Pos_Array: Absolute circular center position. Unit: pulse.

I32 Max_Arc_Speed: Maximum circular interpolation speed (Circular tangent speed). Unit: pulse/sec

I32 Angle: Travel angle. Value range: -360 ~360 degree. Positive for counterclockwise.

Return Values:

I32 Error code: Refer to error code table.

Example:

```
I32 Dimension = 2;  
I32 Axis_ID_Array[2] = { 2, 4 }; //Axis_ID 2 is the master axis.  
I32 Master_Axis_ID = 2; //Axis_ID 2 is the master axis.  
I32 Center_Pos_Array[2] = {100000, 0};  
I32 Max_Arc_Speed = 10000; // pulse/sec  
I32 Angle = -180; // clockwise 180 degree.  
I32 Ret; //Return code.
```

//Configure move profile

```
APS_set_axis_param_f( Master_Axis_ID, PRA_CURVE, 0.5 ); //Set S-factor to 0.5  
APS_set_axis_param_f( Master_Axis_ID, PRA_ACC, 100000.0 ); //Set acceleration  
APS_set_axis_param_f( Master_Axis_ID, PRA_DEC, 100000.0 ); //Set deceleration  
//Perform a circular interpolation  
Ret = APS_absolute_arc_move( Dimension, Axis_ID_Array, Center_Pos_Array, Max_Arc_Speed,  
Angle );
```

See also:

APS_relative_arc_move(),APS_set_axis_param (),APS_get_axis_param (), APS_motion_status(),
APS_stop_move(), APS_emg_stop()

APS_relative_arc_move	Begin a relative distance circular interpolation
-----------------------	--

Descriptions:

This function is used to start an relative circular interpolation positioning motion. User must specified a center position relative current command position and traveling angle for circular interpolation. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. The following axis parameter should be setting before you calling this function.

The details of parameters please refer the axis parameter table.

Symbol define	ParamNo	Description
PRA_SF	0x20	Move S-factor
PRA_ACC	0x21	Acceleration rate
PRA_DEC	0x22	Deceleration rate
PRA_VS	0x23	Start velocity
PRA_VM	0x24	Maximum velocity
PRA_VE	0x25	End velocity

Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. Because the speed parameter is in vector direction(tangent to the circular), this function will take the master axis's acceleration and deceleration time constant to calculate. The master axis is the minimum axis number that user perform an interpolation. For example, Axis ID 2 and 3 are performing a circular interpolation. Therefore, the Axis ID 2 is the master axis.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done. Motion status: in circular interpolation signal (CIP) will be turn on when it start and will be turn off at command is finished. If circular interpolation is stop normally, the normal stop signal (NSTP) will be turn on. On the contrary, circular interpolation is stopped abnormally (ALM, EMG, and so on), abnormal stop signal (ASTP) will be turn on.

During the axis traveling, users can start a new move command including stop command to override the previous one (The dimension and Axis_ID_Array must be the same). The axis will be switched to new command immediately according to new setting of target center position, new speed profile. This command can't be overridden by other motion modes like home operation. Users must stop axis motion before switching to those modes mentioned above.

Note: The 2 axes specified in Axis_ID_Array must be of the same card.

Syntax:

C/C++:

```
I32 APS_relative_arc_move( I32 Dimension, I32 *Axis_ID_Array, I32 *Center_Offset_Array, I32  
Max_Arc_Speed, I32 Angle );
```

Visual Basic:

```
APS_relative_arc_move( ByVal Dimension As Long, Axis_ID_Array As Long, Center_Offset_Array As  
Long, ByVal Max_Arc_Speed As Long, ByVal Angle As Long ) As Long
```

Parameters:

I32 Dimension: The dimension of interpolation axes. (The maximum dimensions refer to product specification)

I32 *Axis_ID_Array: The Axis ID array from 0 to 65535.

I32 *Center_Offset_Array: circular center position relative to current command position. Unit: pulse

I32 Max_Arc_Speed: Maximum circular interpolation speed (Circular tangent speed). Unit: pulse/sec

I32 Angle: Travel angle. Value range: -360 ~360 degree. Positive for counterclockwise.

Return Values:

I32 Error code: Refer to error code table.

Example:

```
I32 Dimension = 2;  
I32 Axis_ID_Array[2] = { 1, 3}; //Axis_ID 1 is the master axis.  
I32 Master_Axis_ID = 1; //Axis_ID 1 is the master axis.  
I32 Center_Offset_Array [2] = {300000, 0};  
I32 Max_Arc_Speed = 20000; // pulse/sec  
I32 Angle = 90; // counterclockwise 90 degree.  
I32 Ret; //Return code.  
  
//Configure move profile  
APS_set_axis_param_f( Master_Axis_ID, PRA_CURVE, 0.5 ); //Set S-curve  
APS_set_axis_param_f( Master_Axis_ID, PRA_ACC, 100000.0 ); //Set acceleration  
APS_set_axis_param_f( Master_Axis_ID, PRA_DEC, 100000.0 ); //Set deceleration  
//Perform a circular interpolation  
Ret = APS_relative_arc_move( Dimension, Axis_ID_Array, Center_Offset_Array, Max_Arc_Speed,  
Angle );
```

See also:

APS_absolute_arc_move (), APS_set_axis_param (),APS_get_axis_param (), APS_motion_status(),
APS_stop_move(), APS_emg_stop()

8. Advanced single move & interpolation

APS_ptp	Begin a single move
---------	---------------------

Descriptions:

This function is used to execute a single move by axis. No any suffix represents that no any motion profile is necessary to perform a single move. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_ptp( I32 Axis_ID, I32 Option, F64 Position, ASYNCALL *Wait);
```

Visual Basic:

```
APS_ptp( ByVal Axis_ID As Long, ByVal Option As Long, ByVal Position As Double, Wait As ASYNCALL)  
As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode					Force Abort	Wait trigger	

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9: This bit is used to assign profile abort behavior when different direction or insufficient decelerating distance happened.

0: Profile will change smoothly. (Default value)

1: Profile will change immediately.

Bit 10 ~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting

0001b(1): Buffered

0010b(2): Blending low

0011b(3): Blending previous
0100b(4): Blending next
0101b(5): Blending high
Bit 16~: Reserved for future, set to 0.

F64 Position: A value specifies how many position/distance to move.

ASYNCALL *Wait: A pointer to ASYNCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

```
I32 opt = 0x1000; //absolute, not wait trigger, Buffered mode
ASYNCALL *wait = NULL; //A waiting call
```

```
//An absolute move to position 10000
APS_ptp( Axis_ID, opt, 10000, wait );
```

See also:

APS_ptp_v	Begin a single move with Vm profile
-----------	-------------------------------------

Descriptions:

This function is used to execute a single move by axis. The _v suffix represents that only one motion profile, that is Vm, is necessary to perform a single move. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_ptp_v( I32 Axis_ID, I32 Option, F64 Position, F64 Vm, ASYNCALL *Wait);
```

Visual Basic:

```
APS_ptp_v( ByVal Axis_ID As Long, ByVal Option As Long, ByVal Position As Double, ByVal Vm As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode						Force Abort	Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9: This bit is used to assign profile abort behavior when different direction or insufficient decelerating distance happened.

0: Profile will change smoothly. (Default value)

1: Profile will change immediately.

Bit 10~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting

0001b(1): Buffered

0010b(2): Blending low

0011b(3): Blending previous

0100b(4): Blending next

0101b(5): Blending high

Bit 16~: Reserved for future, set to 0.

F64 Position: A value specifies how many position/distance to move.

F64 Vm: A value specifies the maximum velocity.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

```
I32 opt = 0x1000; //absolute, not wait trigger, Buffered mode
```

```
ASYNDCALL *wait = NULL; //A waiting call
```

```
//An absolute move to position(10000) with Vm(100000)
```

```
APS_ptp_v ( Axis_ID, opt, 10000, 100000, wait );
```

See also:

APS_ptp_all	Begin a single move with Vm profile
-------------	-------------------------------------

Descriptions:

This function is used to execute a single move. The _all suffix represents that all motion profiles, including Vs, Vm, Ve, Acc, Dec and SFac, are necessary to perform a single move.

Syntax:

C/C++:

```
I32 APS_ptp_all( I32 Axis_ID, I32 Option, F64 Position, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL *Wait);
```

Visual Basic:

```
APS_ptp_all(ByVal Axis_ID As Long, ByVal Option As Long, ByVal Position As Double, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode						Force Abort	Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9: This bit is used to assign profile abort behavior when different direction or insufficient decelerating distance happened.

0: Profile will change smoothly. (Default value)

1: Profile will change immediately.

Bit 10~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting

0001b(1): Buffered

0010b(2): Blending low

0011b(3): Blending previous

0100b(4): Blending next

0101b(5): Blending high

Bit 16~: Reserved for future, set to 0.

F64 Position: A value specifies how many position/distance to move.

F64 Vs: A value specifies the starting velocity.

F64 Vm: A value specifies the maximum velocity.

F64 Ve: A value specifies the ending velocity.

F64 Acc: A value specifies the acceleration.

F64 Dec: A value specifies the deceleration.

F64 SFac: A value specifies the s factor.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

```
I32 opt = 0x1000; //absolute, not wait trigger, Buffered mode
```

```
ASYNDCALL *wait = NULL; //A waiting call
```

```
//An absolute move to position(10000) with Vs(10), Vm(100000), Ve(20), Acc/Dec(200000), SFac(0.5)
```

```
APS_ptp_all( Axis_ID, opt, 10000, 10, 100000, 20, 200000, 200000, 0.5, wait );
```

See also:

APS_vel	Begin a velocity move
---------	-----------------------

Descriptions:

This function is used to execute a velocity move with Vm. No any suffix represents that no any motion profile is necessary to perform a velocity move. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_vel( I32 Axis_ID, I32 Option, F64 Vm, ASYNCALL *Wait);
```

Visual Basic:

```
APS_vel( ByVal Axis_ID As Long, ByVal Option As Long, ByVal Vm As Double, Wait As ASYNCALL) As  
Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							0: Positive / 1: Negative
15	14	13	12	11	10	9	8
						Force Abort	Wait trigger

Bit 0: 0: Positive direction, 1: Negative direction

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9: This bit is used to assign profile abort behavior when different direction happened.

0: Profile will change smoothly. (Default value)

1: Profile will change immediately.

Bit 10~: Reserved for future, set to 0.

F64 Vm: A value specifies the maximum velocity.

ASYNCALL *Wait: A pointer to ASYNCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

```
I32 opt = 0; // Positive direction, not wait trigger
ASYNDCALL *wait = NULL; //A waiting call

//Execute a velocity move with Vm(10000).
APS_vel( Axis_ID, opt, 10000, wait );
```

See also:

APS_vel_all	Begin a velocity move with all profile
-------------	--

Descriptions:

This function is used to execute a velocity move. The _all suffix represents that all motion profiles, including Position, Vs, Vm, Acc and SFac, are necessary to perform a velocity move.

Syntax:

C/C++:

```
I32 APS_vel_all( I32 Axis_ID, I32 Option, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac,  
ASYNDCALL *Wait);
```

Visual Basic:

```
APS_vel_all(ByVal Axis_ID As Long, ByVal Option As Long, ByVal Vs As Double, ByVal Vm As Double,  
ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As  
ASYNDCALL) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							0: Positive / 1: Negative
15	14	13	12	11	10	9	8
						Force Abort	Wait trigger

Bit 0: 0: Positive direction, 1: Negative direction

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9: This bit is used to assign profile abort behavior when different direction happened.

0: Profile will change smoothly. (Default value)

1: Profile will change immediately.

Bit 10~: Reserved for future, set to 0.

F64 Vs: A value specifies the starting velocity.

F64 Vm: A value specifies the maximum velocity.

F64 Ve: A value specifies the ending velocity.

F64 Acc: A value specifies the acceleration.

F64 Dec: A value specifies the deceleration.

F64 SFac: A value specifies the s factor.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

```
I32 opt = 0; // Positive direction, not wait trigger
```

```
ASYNDCALL *wait = NULL; //A waiting call
```

```
//Execute a velocity move with Vs(10), Vm(100000), Ve(20), Acc/Dec(200000), SFac(0.5)
```

```
APS_vel_all( Axis_ID, opt, 10, 100000, 20, 200000, 200000, 0.5, wait );
```

See also:

APS_line	Begin a line interpolation
----------	----------------------------

Descriptions:

This function is used to execute a line interpolation. No any suffix represents that no any motion profile is necessary to perform a line interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_line( I32 Dimension, I32 *Axis_ID_Array, I32 Option, F64 *PositionArray, F64 *TransPara,
ASYNDCALL *Wait);
```

Visual Basic:

```
APS_line (ByVal Dimension As Long, Axis_ID_Array As Long, ByVal Option As Long, PositionArray As
Double, TransPara As Double, Wait As ASYNDCALL) As Long
```

Parameters:

I32 Dimension: A value specifies axes dimension. Range is from 2 to 6.

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Aborting – stop then go ([TransPara_0](#) as deceleration. [dec >0] , if dec <= 0, kernel takes new coming deceleration.)

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *PositionArray: A pointer indicates the starting address of position array.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

```
I32 opt = 0x3000; //absolute, not wait trigger, Buffered mode
```

```
F64 TransPara = 0; //don't care in buffered mode
```

```
ASYNDCALL *wait = NULL; //A waiting call
```

```
//Execute a line move
```

```
APS_line ( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara, wait );
```

See also:

APS_line_v	Begin a line interpolation with Vm profile
------------	--

Descriptions:

This function is used to execute a line interpolation. The _v suffix represents that only one motion profile, that is Vm, is necessary to perform a line interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_line_v( I32 Dimension, I32 *Axis_ID_Array, I32 Option, F64 *PositionArray, F64 *TransPara,  
F64 Vm, ASYNCALL *Wait);
```

Visual Basic:

```
APS_line (ByVal Dimension As Long, Axis_ID_Array As Long, ByVal Option As Long, PositionArray As  
Double, TransPara As Double, ByVal Vm As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 Dimension: A value specifies axes dimension. Range is from 2 to 6.

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Aborting – stop then go ([TransPara_0](#) as deceleration. [dec >0] , if dec <= 0, kernel takes new coming deceleration.)

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *PositionArray: A pointer indicates the starting address of position array.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vm: A value specifies the maximum velocity.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

```
I32 opt = 0x3000; //absolute, not wait trigger, Buffered mode
F64 TransPara = 0; //don't care in buffered mode
ASYNDCALL *wait = NULL; //A waiting call

//Execute a line move with Vm(10000)
APS_line_v( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara, 10000, wait );
```

See also:

APS_line_all	Begin a line interpolation with all profile
--------------	---

Descriptions:

This function is used to execute a line interpolation. The _all suffix represents that all motion profiles, including Vs, Vm, Ve, Acc, Dec and SFac, are necessary to perform a line interpolation.

Syntax:

C/C++:

```
I32 APS_line_all( I32 Dimension, I32 *Axis_ID_Array, I32 Option, F64 *PositionArray, F64 *TransPara,  
F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL *Wait);
```

Visual Basic:

```
APS_line_all(ByVal Dimension As Long, Axis_ID_Array As Long, ByVal Option As Long, PositionArray As  
Double, TransPara As Double, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve As Double, ByVal  
Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 Dimension: A value specifies axes dimension. Range is from 2 to 6.

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Aborting – stop then go ([TransPara_0](#) as deceleration. [dec >0] , if dec <= 0, kernel takes new coming deceleration.)

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *PositionArray: A pointer indicates the starting address of position array.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vs: A value specifies the starting velocity.

F64 Vm: A value specifies the maximum velocity.

F64 Ve: A value specifies the ending velocity.

F64 Acc: A value specifies the acceleration.

F64 Dec: A value specifies the deceleration.

F64 SFac: A value specifies the s factor.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

```
I32 opt = 0x3000; //absolute, not wait trigger, Buffered mode
F64 TransPara = 0; //don't care in buffered mode
ASYNDCALL *wait = NULL; //A waiting call

//Execute a line move with Vs(10), Vm(100000), Ve(20), Acc/Dec(200000), SFac(0.5)
APS_line_all( Dimension, Axis_ID_Array, opt, PositionArray, &TransPara, 10, 100000, 20, 200000,
200000, 0.5, wait );
```

See also:

APS_arc2_ca	Begin an Arc2 move of angle type
-------------	----------------------------------

Descriptions:

This function is used to execute a 2D arc interpolation of angle type, named _arc2_ca. It follows with center position and angle. No any suffix represents that no any motion profile is necessary to perform a 2D arc interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_arc2_ca( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 Angle, F64 *TransPara,  
ASYNDCALL *Wait );
```

Visual Basic:

```
APS_arc2_ca( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, ByVal Angle As  
Double, TransPara As Double, Wait As ASYNDCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance ≥ 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as

residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 Angle: A value specifies the angle. Unit in radian.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc2_ca_v	Begin an Arc2 move of angle type with Vm profile
---------------	--

Descriptions:

This function is used to execute a 2D arc interpolation of angle type, named _arc2_ca. It follows with center position and angle. The _v suffix represents that only one motion profile, that is Vm, is necessary to perform a 2D arc interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_arc2_ca_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 Angle, F64 *TransPara, F64
Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc2_ca_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, ByVal Angle As
Double, TransPara As Double, ByVal Vm As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0,
kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance ≥ 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as

residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 Angle: A value specifies the angle. Unit in radian.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vm: A value specifies the maximum velocity.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc2_ca_all	Begin an Arc2 move of angle type with all profile
-----------------	---

Descriptions:

This function is used to execute a 2D arc interpolation of angle type, named _arc2_ca. It follows with center position and angle. The _all suffix represents that all motion profiles, including Vs, Vm, Ve, Acc, Dec and SFac, are necessary to perform a 2D arc interpolation.

Syntax:

C/C++:

```
I32 APS_arc2_ca_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 Angle, F64 *TransPara,  
F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc2_ca_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, ByVal Angle  
As Double, TransPara As Double, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve As Double, ByVal  
Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0,
kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as

residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 Angle: A value specifies the angle. Unit in radian.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vs: A value specifies the starting velocity.

F64 Vm: A value specifies the maximum velocity.

F64 Ve: A value specifies the ending velocity.

F64 Acc: A value specifies the acceleration.

F64 Dec: A value specifies the deceleration.

F64 SFac: A value specifies the s factor.

ASYNCALL *Wait: A pointer to ASYNCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc2_ce	Begin an Arc2 move of end position
-------------	------------------------------------

Descriptions:

This function is used to execute a 2D arc interpolation of end position type, named _arc2_ce. It follows with center position, end position and Dir. No any suffix represents that no any motion profile is necessary to perform a 2D arc interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_arc2_ce( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *EndArray, I16 Dir, F64  
*TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
I32 APS_arc2_ce( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, EndArray As  
Double, ByVal Dir As Short, TransPara As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc2_ce_v	Begin an Arc2 move of end position with Vm profile
---------------	--

Descriptions:

This function is used to execute a 2D arc interpolation of end position type, named _arc2_ce. It follows with center position, end position and Dir. The _v suffix represents that only one motion profile, that is Vm, is necessary to perform a 2D arc interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_arc2_ce_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *EndArray, I16 Dir, F64
*TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc2_ce_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, EndArray As
Double, ByVal Dir As Short, TransPara As Double, ByVal Vm As Double, Wait As ASYNCALL ) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vm: A value specifies the maximum velocity.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc2_ce_all	Begin an Arc2 move of end position with all profile
-----------------	---

Descriptions:

This function is used to execute a 2D arc interpolation of end position type, named _arc2_ce. It follows with center position, end position and Dir. The _all suffix represents that all motion profiles, including Vs, Vm, Ve, Acc, Dec and SFac, are necessary to perform a 2D arc interpolation.

Syntax:

C/C++:

```
I32 APS_arc2_ce_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *EndArray, I16 Dir, F64
*TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc2_ce_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, EndArray As
Double, ByVal Dir As Short, TransPara As Double, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve
As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As ASYNCALL) As
Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vs: A value specifies the starting velocity.

F64 Vm: A value specifies the maximum velocity.

F64 Ve: A value specifies the ending velocity.

F64 Acc: A value specifies the acceleration.

F64 Dec: A value specifies the deceleration.

F64 SFac: A value specifies the s factor.

ASYNCALL *Wait: A pointer to ASYNCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc3_ca	Begin an Arc3 move of angle type
-------------	----------------------------------

Descriptions:

This function is used to execute a 3D arc interpolation of angle type, named _arc3_ca. It follows with angle, center position and normal vector. No any suffix represents that no any motion profile is necessary to perform a 3D arc interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_arc3_ca( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *NormalArray, F64 Angle,  
F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ca( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray As  
Double, ByVal Angle As Double, TransPara As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 * NormalArray: A pointer indicates the starting address of normal vector array.

F64 Angle: A value specifies the angle. Unit in radian.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc3_ca_v	Begin an Arc3 move of angle type with Vm profile
---------------	--

Descriptions:

This function is used to execute a 3D arc interpolation of angle type, named _arc3_ca. It follows with angle, center position and normal vector. The _v suffix represents that only one motion profile, that is Vm, is necessary to perform a 3D arc interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_arc3_ca_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *NormalArray, F64 Angle,
F64 *TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ca_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray
As Double, ByVal Angle As Double, TransPara As Double, ByVal Vm As Double, Wait As ASYNCALL) As
Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 * NormalArray: A pointer indicates the starting address of normal vector array.

F64 Angle: A value specifies the angle. Unit in radian.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vm: A value specifies the maximum velocity.

ASYNCALL *Wait: A pointer to ASYNCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc3_ca_all	Begin an Arc3 move of angle type with all profile
-----------------	---

Descriptions:

This function is used to execute a 3D arc interpolation of angle type, named _arc3_ca. It follows with angle, center position and normal vector. The _all suffix represents that all motion profiles, including Vs, Vm, Ve, Acc, Dec and SFac, are necessary to perform a 3D arc interpolation.

Syntax:

C/C++:

```
I32 APS_arc3_ca_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *NormalArray, F64 Angle,
F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ca_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray
As Double, ByVal Angle As Double, TransPara As Double, ByVal Vs As Double, ByVal Vm As Double,
ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As
ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *NormalArray: A pointer indicates the starting address of normal vector array.

F64 Angle: A value specifies the angle. Unit in radian.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vs: A value specifies the starting velocity.

F64 Vm: A value specifies the maximum velocity.

F64 Ve: A value specifies the ending velocity.

F64 Acc: A value specifies the acceleration.

F64 Dec: A value specifies the deceleration.

F64 SFac: A value specifies the s factor.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc3_ce	Begin an Arc3 move of end position
-------------	------------------------------------

Descriptions:

This function is used to execute a 3D arc interpolation of end position type, named _arc3_ce. It follows with center position, end position and Dir. No any suffix represents that no any motion profile is necessary to perform a 3D arc interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_arc3_ce( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *EndArray, I16 Dir, F64  
*TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
I32 APS_arc3_ce( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, EndArray As  
Double, ByVal Dir As Short, TransPara As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc3_ce_v	Begin an Arc3 move of end position with Vm profile
---------------	--

Descriptions:

This function is used to execute a 3D arc interpolation of end position type, named _arc3_ce. It follows with center position, end position and Dir. The _v suffix represents that only one motion profile, that is Vm, is necessary to perform a 3D arc interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_arc3_ce_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *EndArray, I16 Dir, F64
*TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ce_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, EndArray As
Double, ByVal Dir As Short, TransPara As Double, ByVal Vm As Double, Wait As ASYNCALL ) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vm: A value specifies the maximum velocity.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_arc3_ce_all	Begin an Arc3 move of end position with all profile
-----------------	---

Descriptions:

This function is used to execute a 3D arc interpolation of end position type, named _arc3_ce. It follows with center position, end position and Dir. The _all suffix represents that all motion profiles, including Vs, Vm, Ve, Acc, Dec and SFac, are necessary to perform a 3D arc interpolation.

Syntax:

C/C++:

```
I32 APS_arc3_ce_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *EndArray, I16 Dir, F64
*TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL *Wait );
```

Visual Basic:

```
APS_arc3_ce_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, EndArray As
Double, ByVal Dir As Short, TransPara As Double, ByVal Vs As Double, ByVal Vm As Double, ByVal Ve
As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As Double, Wait As ASYNCALL) As
Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vs: A value specifies the starting velocity.

F64 Vm: A value specifies the maximum velocity.

F64 Ve: A value specifies the ending velocity.

F64 Acc: A value specifies the acceleration.

F64 Dec: A value specifies the deceleration.

F64 SFac: A value specifies the s factor.

ASYNCALL *Wait: A pointer to ASYNCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_spiral_ca	Begin a 3D spiral-helix move of angle type
---------------	--

Descriptions:

This function is used to execute a 3D spiral-helix interpolation of angle type, named _spiral_ca. It follows with angle, center position, normal vector, DeltaH and FinalR. No any suffix represents that no any motion profile is necessary to perform a 3D spiral-helix interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_spiral_ca( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *NormalArray, F64 Angle,  
F64 DeltaH, F64 FinalR, F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
APS_spiral_ca( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray As  
Double, ByVal Angle As Double, ByVal DeltaH As Double, ByVal FinalR As Double, TransPara As Double,  
Wait As ASYNCALL ) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *NormalArray: A pointer indicates the starting address of normal vector array.

F64 Angle: A value specifies the angle. Unit in radian.

F64 DeltaH: A value specifies the height.

F64 FinalR: A value specifies the distant from end position to normal vector.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_spiral_ca_v	Begin a 3D spiral-helix move of angle type with Vm profile
-----------------	--

Descriptions:

This function is used to execute a 3D spiral-helix interpolation of angle type, named _spiral_ca. It follows with angle, center position, normal vector, DeltaH and FinalR. The _v suffix represents that only one motion profile, that is Vm, is necessary to perform a 3D spiral-helix interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_spiral_ca_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *NormalArray, F64 Angle,
F64 DeltaH, F64 FinalR, F64 *TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_spiral_ca_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray
As Double, ByVal Angle As Double, ByVal DeltaH As Double, ByVal FinalR As Double, TransPara As
Double, ByVal Vm As Double, Wait As ASYNCALL ) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *NormalArray: A pointer indicates the starting address of normal vector array.

F64 Angle: A value specifies the angle. Unit in radian.

F64 DeltaH: A value specifies the height.

F64 FinalR: A value specifies the distant from end position to normal vector.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vm: A value specifies the maximum velocity.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_spiral_ca_all	Begin a 3D spiral-helix move of angle type with all profile
-------------------	---

Descriptions:

This function is used to execute a 3D spiral-helix interpolation of angle type, named _spiral_ca. It follows with angle, center position, normal vector, DeltaH and FinalR. The _all suffix represents that all motion profiles, including Vs, Vm, Ve, Acc, Dec and SFac, are necessary to perform a 3D spiral-helix interpolation.

Syntax:

C/C++:

```
I32 APS_spiral_ca_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *NormalArray, F64
Angle, F64 DeltaH, F64 FinalR, F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac,
ASYNDCALL *Wait );
```

Visual Basic:

```
APS_spiral_ca_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray
As Double, ByVal Angle As Double, F64 DeltaH, F64 FinalR, TransPara As Double, ByVal Vs As Double,
ByVal Vm As Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As
Double, Wait As ASYNDCALL ) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event
0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance **>= 0.0**)
0110b(6): Blending – Residue distance in travel distance's percentage ([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)
Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.
F64 *NormalArray: A pointer indicates the starting address of normal vector array.
F64 Angle: A value specifies the angle. Unit in radian.
F64 DeltaH: A value specifies the height.
F64 FinalR: A value specifies the distant from end position to normal vector.
F64 *TransPara: A pointer indicates the starting address of transfer parameters.
F64 Vs: A value specifies the starting velocity.
F64 Vm: A value specifies the maximum velocity.
F64 Ve: A value specifies the ending velocity.
F64 Acc: A value specifies the acceleration.
F64 Dec: A value specifies the deceleration.
F64 SFac: A value specifies the s factor.
ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**
Passing it with NULL defines a waiting call.
If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_spiral_ce	Begin a 3D spiral-helix move of end position
---------------	--

Descriptions:

This function is used to execute a 3D spiral-helix interpolation of end position type, named _spiral_ce. It follows with center position, normal vector, end position and Dir. No any suffix represents that no any motion profile is necessary to perform a 3D spiral-helix interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_spiral_ce( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64
*EndArray, I16 Dir, F64 *TransPara, ASYNCALL *Wait );
```

Visual Basic:

```
I32 APS_spiral_ce( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray
As Double, EndArray As Double, ByVal Dir As Short, TransPara As Double, Wait As ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance >= 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *NormalArray: A pointer indicates the starting address of the normal vector array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

ASYNDCALL *Wait: A pointer to ASYNDCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_spiral_ce_v	Begin a 3D spiral-helix move of end position with Vm profile
-----------------	--

Descriptions:

This function is used to execute a 3D spiral-helix interpolation of end position type, named _spiral_ce. It follows with center position, normal vector, end position and Dir. The _v suffix represents that only one motion profile, that is Vm, is necessary to perform a 3D spiral-helix interpolation. Other motion profiles are set in axis parameters. User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_spiral_ce_v( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *NormalArray, F64
*EndArray, I16 Dir, F64 *TransPara, F64 Vm, ASYNCALL *Wait );
```

Visual Basic:

```
APS_spiral_ce_v( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray
As Double, EndArray As Double, ByVal Dir As Short, TransPara As Double, ByVal Vm As Double, Wait
As ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance ≥ 0.0)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *NormalArray: A pointer indicates the starting address of the normal vector array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vm: A value specifies the maximum velocity.

ASYNCALL *Wait: A pointer to ASYNCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

APS_spiral_ce_all	Begin a 3D spiral-helix move of end position with all profile
-------------------	---

Descriptions:

This function is used to execute a 3D spiral-helix interpolation of end position type, named _spiral_ce. It follows with center position, normal vector, end position and Dir. The _all suffix represents that all motion profiles, including Vs, Vm, Ve, Acc, Dec and SFac, are necessary to perform a 3D spiral-helix interpolation.

Syntax:

C/C++:

```
I32 APS_spiral_ce_all( I32 *Axis_ID_Array, I32 Option, F64 *CenterArray, F64 *NormalArray, F64
*EndArray, I16 Dir, F64 *TransPara, F64 Vs, F64 Vm, F64 Ve, F64 Acc, F64 Dec, F64 SFac, ASYNCALL
*Wait );
```

Visual Basic:

```
APS_spiral_ce_all( Axis_ID_Array As Long, ByVal Option As Long, CenterArray As Double, NormalArray
As Double, EndArray As Double, ByVal Dir As Short, TransPara As Double, ByVal Vs As Double, ByVal
Vm As Double, ByVal Ve As Double, ByVal Acc As Double, ByVal Dec As Double, ByVal SFac As Double,
Wait As ASYNCALL) As Long
```

Parameters:

I32 *Axis_ID_Array: A pointer indicates the starting address of axes array.

I32 Option: A bit set specifies the option, which could enable specified parameters and functions.

7	6	5	4	3	2	1	Bit : 0
							Absolute(0) / Relative(1)
15	14	13	12	11	10	9	8
Buffer mode							Wait trigger

Bit 0: 1:Relative move, 0:Absolute move

Bit 1~7: Reserved for future, set to 0.

Bit 8: Set a move to waiting state. This axis will not move until triggered.

Bit 9~11: Reserved for future, set to 0.

Bit 12~15: Buffer mode:

0000b(0): Aborting – stop and blend ([TransPara_0](#) as deceleration. [dec >0], if dec <= 0, kernel takes new coming deceleration.)

0001b(1): Aborting – force abort

0010b(2): Reserved. **Note: if setting to mode 2, it will return error code.**

0011b(3): Buffered

0100b(4): Blending – Deceleration event

0101b(5): Blending – Residue distance ([TransPara_0](#) as residue distance **>= 0.0**)

0110b(6): Blending – Residue distance in travel distance's percentage([TransPara_0](#) as residue distance % value range: 0.0 ~ 1.0)

Bit 16~: Reserved for future, set to 0.

F64 *CenterArray: A pointer indicates the starting address of center array.

F64 *NormalArray: A pointer indicates the starting address of the normal vector array.

F64 *EndArray: A pointer indicates the starting address of end array.

I16 Dir: A value specifies the rotate direction. If dir set 0 means rotate in positive direction, dir = -1 rotate in negative direction. The total rotate $angle = theta + Dir \times 2\pi$, where $theta$ is the angle of two vectors: center to start and center to end.

F64 *TransPara: A pointer indicates the starting address of transfer parameters.

F64 Vs: A value specifies the starting velocity.

F64 Vm: A value specifies the maximum velocity.

F64 Ve: A value specifies the ending velocity.

F64 Acc: A value specifies the acceleration.

F64 Dec: A value specifies the deceleration.

F64 SFac: A value specifies the s factor.

ASYNCALL *Wait: A pointer to ASYNCALL structure. **Note: It is reserved for future.**

Passing it with NULL defines a waiting call.

If it is a valid pointer, the call is non-waiting and the functions returns immediately.

Return Values:

I32 Error code: Please refer to [APS Functions Return Code](#).

Example:

See also:

9. Multi-axes move trigger & stop

APS_move_trigger	Send a trigger to sync all waiting moves
------------------	--

Descriptions:

The function is used to send a trigger to sync all waiting moves. Refer to chapter [advanced single move & interpolation](#). User could set bit 8 of Option parameter by invoking advanced motion functions, so this move will set to waiting state.

Syntax:

C/C++

```
I32 APS_move_trigger( I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_move_trigger(ByVal Dimension As Long, Axis_ID_Array As Long ) As Long
```

Parameters:

I32 Dimension: The dimension of simultaneous axes.

I32 *Axis_ID_Array: The axis ID array from 0 to 65535.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 Axis_ID_Array[2] = { axis_id0, axis_id1 };
//Bit 8 set to 1. Be a waiting state.
I32 opt = 0x0100; //absolute, wait trigger, Aborting mode
ASYNCALL *wait = NULL;

//An absolute move to position 10000 in waiting state
APS_ptp( axis_id0, opt, 10000, wait );
APS_ptp( axis_id1, opt, 10000, wait );

// send a trigger to sync all waiting moves
APS_move_trigger( 2, Axis_ID_Array );
...
// Stop a simultaneous move
APS_stop_move_multi ( 2, Axis_ID_Array );
```

See also:

[APS_stop_move_multi](#)

APS_stop_move_multi	Multi-axes stop move
---------------------	----------------------

Descriptions:

This function is used to stop multiple axes motion at the same time. Generally speaking, it is used to stop synchronized move. The deceleration profile, defined to be PRA_SD_DEC, is set by invoking APS_set_axis_param_f(). User could refer to [axis parameter table](#) for the details.

Syntax:

C/C++:

```
I32 APS_stop_move_multi ( I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_stop_move_multi (ByVal Dimension As Long, Axis_ID_Array As Long) As Long
```

Parameters:

I32 Dimension: The dimension of stopped axes.

I32 *Axis_ID_Array: The axis ID array from 0 to 65535.

Return Values:

I32 Error code: Refer to error code table.

Example:

```
I32 Axis_ID_Array[2] = { axis_id0, axis_id1 };
//Bit 8 set to 1. Be a waiting state.
I32 opt = 0x0100; //absolute, wait trigger, Aborting mode
ASYNCALL *wait = NULL;

//An absolute move to position 10000 in waiting state
APS_ptp( axis_id0, opt, 10000, wait );
APS_ptp( axis_id1, opt, 10000, wait );

// send a trigger to sync all waiting moves
APS_move_trigger( 2, Axis_ID_Array );
...
// Stop a simultaneous move
APS_stop_move_multi ( 2, Axis_ID_Array );
```

See also:

APS_move_trigger()

APS_emg_stop_multi	Multi-axes emg stop move
--------------------	--------------------------

Descriptions:

This function is used to stop immediately multiple axes motion at the same time. Because the stop function will stop axis accidentally, it will generate an abnormal stop interrupt event rather than normal stop event if interrupt factor is set. The motion status will also be set to an abnormal stop status. The abnormal stop status or event will be clear by next motion command. This function has no deceleration profile.

Syntax:

C/C++:

```
I32 APS_emg_stop_multi ( I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_emg_stop_multi (ByVal Dimension As Long, Axis_ID_Array As Long) As Long
```

Parameters:

I32 Dimension: The dimension of stopped axes.

I32 *Axis_ID_Array: The axis ID array from 0 to 65535.

Return Values:

I32 Error code: Refer to error code table.

Example:

```
I32 Axis_ID_Array[2] = { axis_id0, axis_id1 };
//Bit 8 set to 1. Be a waiting state.
I32 opt = 0x0100; //absolute, wait trigger, Aborting mode
ASYNCALL *wait = NULL;

//An absolute move to position 10000 in waiting state
APS_ptp( axis_id0, opt, 10000, wait );
APS_ptp( axis_id1, opt, 10000, wait );

// send a trigger to sync all waiting moves
APS_move_trigger( 2, Axis_ID_Array );
...
// Emg stop a simultaneous move
APS_emg_stop_multi ( 2, Axis_ID_Array );
```

See also:

`APS_move_trigger()`

10. Interrupt

APS_int_enable	Interrupt main switch
----------------	-----------------------

Descriptions:

This function is used to enable/disable interrupt of one board to host computer. It is a hardware main switch of this board. Once it is disabled, host computer will not received any hardware interrupt even the interrupt factor is enabled. Users must enable this function before using any interrupt relative functions and disable this function when users do not use interrupt anymore.

Syntax:

C/C++:

```
I32 APS_int_enable( I32 Board_ID, I32 Enable );
```

Visual Basic:

```
APS_int_enable (ByVal Board_ID As Long, ByVal Enable As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Enable: Enable/Disable interrupt.

0: Disable. 1: Enable

Return Values:

I32 Error code: Refer to error code table.

Example:

```
I32 Int_No; //Interrupt number
I32 returnCode; // function return code
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //Enable the interrupt factor
APS_int_enable( Board_ID, 1 ); //Enable the interrupt main switch
returnCode = APS_wait_single_int( Int_No, Time_Out ); //Wait interrupt
if( returnCode == ERR_NoError )
{ //Interrupt occurred
    APS_reset_int( Int_No );
    ...//Do something
}
```

```
APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //Disable the interrupt factor  
APS_int_enable( Board_ID, 0 ); //Disable the interrupt main switch
```

See also:

[APS_set_int_factor\(\)](#); [APS_get_int_factor\(\)](#); [APS_wait_single_int\(\)](#); [APS_wait_multiple_int\(\)](#);
[APS_reset_int\(\)](#); [APS_set_int\(\)](#);

APS_set_int_factor	Enable/Disable interrupt factor and get interrupt handle.
--------------------	---

Descriptions :

This function is used to turn on/off the interrupt factor bit. If it is turned on, the function will return a notification event for this bit and return an I32 type event number. Users can wait this event by assigning corresponding event number into a wait function. The event number is unique in one system but it is not a event handler. It is just a virtual number of event APS converts.

The interrupt factor definition, please refer to the interrupt factor table.

Syntax:

C/C++:

```
I32 APS_set_int_factor( I32 Board_ID, I32 Item_No, I32 Factor_No, I32 Enable );
```

Visual Basic:

```
APS_set_int_factor (ByVal Board_ID As Long, ByVal Item_No As Long, ByVal Factor_No As Long, ByVal
Enable As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Item_No: Interrupt factor table item number. Refer to interrupt factor table.

I32 Factor_No: Factor number of one item. Refer to interrupt factor table.

I32 Enable: Enable interrupt factor. 0: Disable; 1:Enable

Return Values:

When:

[Enable = 1] : Enable the interrupt factor

Return positive value: I32 Interrupt event number.

Return negative value: I32 error code. Refer to error code table.

[Enable = 0] : Disable the interrupt factor

Return I32 error code. Refer to error code table.

Example:

```
<Set axis 2 IMDN interrupt of PCI-8254/58
```

```
I32 Int_No; //Interrupt number
```

```
I32 returnCode; // function return code
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No=2, Factor_No=BIT12, 1 ); //Enable the interrupt
factor
```

```
APS_int_enable( Board_ID, 1 ); //Enable the interrupt main switch
```

```
returnCode = APS_wait_single_int( Int_No, Time_Out ); //Wait interrupt
if( returnCode == ERR_NoError )
{ //Interrupt occurred
    APS_reset_int( Int_No );
    ...//Do something
}

APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the interrupt main switch
```

See also:

[APS_int_enable\(\)](#); [APS_get_int_factor\(\)](#); [APS_wait_single_int\(\)](#); [APS_wait_multiple_int\(\)](#);
[APS_reset_int\(\)](#); [APS_set_int\(\)](#);

APS_get_int_factor	Get interrupt factor enable or disable
--------------------	--

Descriptions :

This function is used to get the setting of interrupt factor.

Syntax:

C/C++:

```
I32 APS_get_int_factor( I32 Board_ID, I32 Item_No, I32 Factor_No, I32 *Enable );
```

Visual Basic:

```
APS_get_int_factor (ByVal Board_ID As Long, ByVal Item_No As Long, ByVal Factor_No As Long,  
Enable As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Item_No: Interrupt factor table item number. Refer to interrupt factor table.

I32 Factor_No: Factor number of one item. Refer to interrupt factor table.

I32 *Enable: Return enable or disable. 0: Disable, 1:Enable.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ReturnCode;  
I32 Enable;  
ReturnCode = APS_get_int_factor( Board_ID, Item_No, Factor_No, &Enable );  
...
```

See also:

APS_int_enable(); APS_set_int_factor(); APS_wait_single_int(); APS_wait_multiple_int();
APS_reset_int(); APS_set_int();

APS_wait_single_int	Wait single interrupt event
---------------------	-----------------------------

Descriptions:

When the user enabled the interrupt function for specified factors by “APS_set_int_factor”, it could use this function to wait a specific interrupt. When this function was running, the process would never stop until the event was triggered or the function was time out. This function returns when one of the following occurs:

1. The specified interrupt factor is in the signaled state.
2. The time-out interval elapses.

This function checks the current state of the specified interrupt factor. If the state is non-signaled, the calling thread enters the wait state. It uses no processor time while waiting for the INT state to become signaled or the time-out interval to elapse.

When the interrupt is occurred and the wait function is return. User should use **APS_reset_int ()** to reset the interrupt by themselves. If user does not reset the interrupt, the wait function will pass immediately next time.

Syntax:

C/C++:

```
I32 APS_wait_single_int( I32 Int_No, I32 Time_Out );
```

Visual Basic:

```
APS_wait_single_int (ByVal Int_No As Long, ByVal Time_Out As Long) As Long
```

Parameters:

I32 Int_No: Interrupt event number. Get from APS_set_int_factor() function.

I32 Time_Out: Wait timeout time. Unit is milli-second. If value is set -1, the function's time-out interval never elapses (infinite). If *Time_Out* is zero, the function tests the interrupt's state and returns immediately.

Return Values:

ERR_NoError(0): The event is wait success.

I32 error code. Refer to error code table.

Example:

```
I32 Int_No; //Interrupt number
I32 returnCode; // function return code
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //Enable the interrupt factor
```

```
APS_int_enable( Board_ID, 1 ); //Enable the interrupt main switch
returnCode = APS_wait_single_int( Int_No, Time_Out ); //Wait interrupt
if( returnCode == ERR_NoError )
{ //Interrupt occurred
    APS_reset_int( Int_No );
    ...//Do something
}

APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the interrupt main switch
```

See also:

[APS_int_enable\(\)](#); [APS_set_int_factor\(\)](#); [APS_get_int_factor\(\)](#); [APS_wait_multiple_int\(\)](#);
[APS_reset_int\(\)](#); [APS_set_int\(\)](#);

APS_wait_multiple_int	Wait multiple interrupt events
-----------------------	--------------------------------

Descriptions:

When the user enabled the interrupt function for specified factors by “**APS_set_int_factor()**”, users could use this function to wait specific interrupts. When this function was running, the process would never stop until the event was triggered or the function was time out. This function returns when one of the following occurs:

1. Either any one or all of the interrupt factors are in the signaled state.
2. The time-out interval elapses.

This function checks the current state of the specified interrupt factor. If the state is non-signaled, the calling thread enters the wait state. It uses no processor time while waiting for the INT state to become signaled or the time-out interval to elapse.

Users must use **APS_reset_int()** to reset the events themselves before wait the events next time.

Syntax:

C/C++:

```
I32 APS_wait_multiple_int( I32 Int_Count, I32 *Int_No_Array, I32 Wait_All, I32 Time_Out );
```

Visual Basic:

```
APS_wait_multiple_int (ByVal Int_Count As Long, Int_No_Array As Long, ByVal Wait_All As Long,  
ByVal Time_Out As Long) As Long
```

Parameters:

I32 Int_Count: Specifies the number of Interrupt. The maximum number of factors is 64.

I32 *Int_No_Array: Interrupt event number array. Get from APS_set_int_factor() function.

I32 Wait_All: Wait option.

 FALSE: (0) The function returns when the state of any one of the events in the array is signaled.

 TRUE: (1) The function returns when the state of all events in the array is signaled.

I32 Time_Out: Wait timeout time. Unit is milli-second. If value is set -1, the function’s time-out interval never elapses (infinite).

Return Values:

Positive value: (Int_Count – 1): The events are wait success.

 If Wait_All is FALSE (0), the return value indicates that the state of all specified objects is signaled.

 If WaitAll is FALSE(0), the return value indicates the array index of the object that satisfied the wait. If more than one event became peratio during the call, this is the array index of the peratio object with the smallest index value of all the peratio objects.

Negative value: I32 error code: Refer to error code table.

Example:

```
I32 Int_No[2]; //Interrupt number
```

```
I32 returnCode; // function return code
```

```
Int_No[0] = APS_set_int_factor( Board_ID, Item_No1, Factor_No1, 1 ); //Enable the interrupt factor
```

```
Int_No[1] = APS_set_int_factor( Board_ID, Item_No2, Factor_No2, 1 ); //Enable the interrupt factor
```

```
APS_int_enable( Board_ID, 1 ); //Enable the interrupt main switch
```

```
returnCode = APS_wait_multiple_int( 2, Int_No, 1, I32 Time_Out ); //Wait multiple interrupts, (wait all)
```

```
if( returnCode == ERR_NoError )
```

```
{ //Interruptions occurred
```

```
    APS_reset_int( Int_No[0] );
```

```
    APS_reset_int( Int_No[1] );
```

```
    ...//Do something
```

```
}
```

```
APS_set_int_factor( Board_ID, Item_No1, Factor_No1, 0 ); //Disable the interrupt factor
```

```
APS_set_int_factor( Board_ID, Item_No2, Factor_No2, 0 ); //Disable the interrupt factor
```

```
APS_int_enable( Board_ID, 0 ); //Disable the interrupt main switch
```

See also:

`APS_int_enable(); APS_set_int_factor(); APS_get_int_factor(); APS_wait_single_int(); APS_reset_int();`

`APS_set_int();`

APS_reset_int	Reset interrupt event to non-signaled state.
---------------	--

Descriptions :

This function is used to reset singled event to non-singled state.

Syntax:

C/C++:

```
I32 APS_reset_int( I32 Int_No );
```

Visual Basic:

```
APS_reset_int (ByVal Int_No As Long) As Long
```

Parameters:

I32 Int_No: Interrupt event number. Get from APS_set_int_factor() function.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 Int_No; //Interrupt number
```

```
I32 returnCode; // function return code
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //Enable the interrupt factor
```

```
APS_int_enable( Board_ID, 1 ); //Enable the interrupt main switch
```

```
returnCode = APS_wait_single_int( Int_No, Time_Out ); //Wait interrupt
```

```
if( returnCode == ERR_NoError )
```

```
{ //Interrupt occurred
```

```
    APS_reset_int( Int_No );
```

```
    ...//Do something
```

```
}
```

```
APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //Disable the interrupt factor
```

```
APS_int_enable( Board_ID, 0 ); //Disable the interrupt main switch
```

See also:

APS_int_enable(); APS_set_int_factor(); APS_get_int_factor(); APS_wait_single_int();
 APS_wait_multiple_int(); APS_set_int();

APS_set_int	Set interrupt event to signaled state.
-------------	--

Descriptions :

This function is used to signal the specified event interrupt. The wait function will return (pass) when this function is set.

Syntax:

C/C++:

I32 APS_set_int(I32 Int_No);

Visual Basic:

APS_set_int (ByVal Int_No As Long) As Long

Parameters:

I32 Int_No: Interrupt event number. Get from APS_set_int_factor() function.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 Int_No; //Interrupt number
```

```
I32 returnCode; // function return code
```

```
Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //Enable the interrupt factor
```

```
APS_int_enable( Board_ID, 1 ); //Enable the interrupt main switch
```

```
APS_set_int(Int_No ); //Signaled interrupt event.
```

```
returnCode = APS_wait_single_int( Int_No, Time_Out ); //Wait function will pass immediately
```

```
if( returnCode == ERR_NoError )
```

```
{ //Interrupt occurred
```

```
    APS_reset_int( Int_No );
```

```
    ...//Do something
```

```
}
```

```
APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //Disable the interrupt factor
```

```
APS_int_enable( Board_ID, 0 ); //Disable the interrupt main switch
```

See also:

```
APS_int_enable(); APS_set_int_factor(); APS_get_int_factor(); APS_wait_single_int();  
APS_wait_multiple_int(); APS_reset_int();
```

APS_set_int_factorH	Enable/Disable interrupt factor and get interrupt handle.(Win32)
---------------------	--

Descriptions :

This function is used to turn on/off the interrupt factor bit. If it is turned on, the function will return a notification event for this bit and return a HANDLE type (define in windows.h) event handle. Users can use this handle directly with win32 API functions. The event number is unique in one system.

The interrupt factor definition, please refer to the interrupt factor table.

Syntax:

C/C++:

```
HANDLE APS_set_int_factorH( I32 Board_ID, I32 Item_No, I32 Factor_No, I32 Enable );
```

Visual Basic:

```
APS_set_int_factorH (ByVal Board_ID As Long, ByVal Item_No As Long, ByVal Factor_No As Long,  
ByVal Enable As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Item_No: Interrupt factor table item number. Refer to interrupt factor table.

I32 Factor_No: Factor number of one item. Refer to interrupt factor table.

I32 Enable: Enable interrupt factor. 0: Disable; 1:Enable

Return Values:

When:

[Enable = 1] : Enable the interrupt factor

Return win32 event handle if function success, or return null(0) for failed.

[Enable = 0] : Disable the interrupt factor

Return null(0).

Example:

```
#include <windows.h>
```

```
HANDLE hInt; //Interrupt handle
```

```
DWORD returnCode; // function return code
```

```
hInt = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //Enable the interrupt factor
```

```
APS_int_enable( Board_ID, 1 ); //Enable the interrupt main switch
```

```
returnCode = WaitForSingleObject( hInt, 1000 );
```

```
if( returnCode == WAIT_OBJECT_0 )
{ //Interrupt occurred
    ResetEvent (hInt ); //Win32 SDK function
    ...//Do something
}

APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the interrupt main switch
```

See also:

APS_int_enable(); APS_get_int_factor(); APS_wait_single_int(); APS_wait_multiple_int();
APS_reset_int(); APS_set_int();

APS_int_no_to_handle	Convert interrupt event number to interrupt handle.(Win32)
----------------------	--

Descriptions :

This function is used to convert interrupt number to a HANDLE type (define in windows.h) event handle. User could get an I32 type event number by APS_set_factor(), then convert this number to a HANDLE.

Syntax:

C/C++:

HANDLE APS_int_no_to_handle(I32 Int_No);

Visual Basic:

APS_int_no_to_handle(ByVal Int_No As Long) As Long

Parameters:

I32 Int_No: Interrupt event number. Get from APS_set_int_factor() function.

Return Values:

Return win32 event handle.

Example:

```
#include <windows.h>
HANDLE hInt; //Interrupt handle
I32 Int_No;
DWORD returnCode; // function return code

Int_No = APS_set_int_factor( Board_ID, Item_No, Factor_No, 1 ); //Enable the interrupt factor
hInt = APS_int_no_to_handle( Int_No ); //Convert to a handle.
APS_int_enable( Board_ID, 1 ); //Enable the interrupt main switch

returnCode = WaitForSingleObject( hInt, 1000 );

if( returnCode == WAIT_OBJECT_0 )
{ //Interrupt occurred
    ResetEvent (hInt ); //Win32 SDK function
    ...//Do something
}
```

```
APS_set_int_factor( Board_ID, Item_No, Factor_No, 0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the interrupt main switch
```

See also:

APS_int_enable(); APS_set_int_factor(); APS_set_field_bus_int_factor_motion();

11. Sampling

APS_set_sampling_param	Set sampling parameter.
------------------------	-------------------------

Descriptions:

This function is used to set sampling parameters such as sampling rate, sampling channel source and so on. Please refer to the [sampling parameters table](#) for the definition and detail descriptions.

Sampling function is only for the boards have DSP or CPU inside. It is for real-time issue. The sampling functions guarantees each sampled point are record under hard realtime environment.

Syntax:

C/C++:

```
I32 APS_set_sampling_param( I32 Board_ID, I32 Param_No, I32 Param_Dat );
```

Visual Basic:

```
APS_set_sampling_param( ByVal Board_ID As Long, ByVal ParaNum As Long, ByVal ParaDat As Long )  
As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Param_No: Specified sampling parameter number, refer to [sampling parameter table](#) for definition.

I32 Param_Dat: The corresponding parameter value of sampling number. Refer to the sampling table.

Return Values:

I32 error code. Refer to error code table.

Example:

```
//... initial card.  
I32 Ret = APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 2 ); //Set sampling rate  
...
```

See also:

[APS_get_sampling_param\(\)](#); [APS_wait_trigger_sampling\(\)](#)

APS_get_sampling_param	Get sampling parameter.
------------------------	-------------------------

Descriptions:

This function is used to get sampling parameters such as sampling rate, sampling channel source and so on. Please refer to the [sampling parameters table](#) for the definition and detail descriptions.

Syntax:

C/C++:

```
I32 APS_get_sampling_param( I32 Board_ID, I32 ParaNum, I32 *ParaDat );
```

Visual Basic:

```
APS_get_sampling_param( ByVal Board_ID As Long, ByVal ParaNum As Long, ParaDat As Long ) As
Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 ParaNum: Sampling parameter number. Refer to the [sampling parameter table](#).

I32 *ParaDat: Return sampling parameter value. Refer to the [sampling parameter table](#).

Return Values:

I32 error code. Refer to error code table.

Example:

I32 ParaDat:

```
Ret = APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, & ParaDat ); //Get trigger edge
```

...

See also:

[APS_set_sampling_param\(\)](#); [APS_wait_trigger_sampling\(\)](#)

APS_wait_trigger_sampling	Waiting for sample data.
---------------------------	--------------------------

Descriptions:

This function is used to sample data from controller. When the function is issued, the program stating to sample the information and put the data to the internal buffer. Until the trigger signal is turned on, program fetched a mass of data which size is pre-trigger length from internal buffer to the user's data buffer and continuous sample the data until reach the length that users designated. In other hand, if the timeout time is reached and the trigger signal does not raised, this function will be timeout and return an error message.

Use [APS_stop_wait_sampling](#) to forced stop the wait sampling

Caution:

[APS_wait_trigger_sampling](#), [APS_wait_trigger_sampling_async](#) and [APS_auto_sampling](#) functions cannot be used at the same time.

Syntax:

C/C++:

```
I32 APS_wait_trigger_sampling( I32 Board_ID, I32 Length, I32 PreTrgLen, I32 TimeOutMs,
STR_SAMP_DATA_4CH *DataArr );
```

Visual Basic:

```
APS_wait_trigger_sampling(ByValBoard_ID As Long, ByVal Length As Long, ByVal PreTrgLen As Long,
 ByVal TimeOutMs As Long, DataArr As STR_SAMP_DATA_4CH ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to [APS_initial\(\)](#).

I32 Length: The number of sampling data. (array size)

I32 PreTrgLen: Pre-trigger length.

I32 TimeOutMs: Timeout time. Unit is millisecond.

STR_SAMP_DATA_4CH *DataArr: Get sampling data structure array. Array size must be larger than the parameter "Length".

Return Values:

I32 error code. Refer to error code table.

Example:

```
//... initial card.
```

```
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 2 ); //Set sampling rate
```

```

APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, 0 ); //Set trigger edge (rising edge)
APS_set_sampling_param( Board_ID, SAMP_PA_LEVEL, 1 ); //Set trigger level ( 1 )
APS_set_sampling_param( Board_ID, SAMP_PA_TRIGCH, 0 ); //Set trigger channel (channel 0)
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //Set channel_0 sampling
source.

APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH1, SAMP_MIO_INP ); //Set channel_1 sampling
source.

I32 Length = 1024; //Total sampling data array size.

I32 PreTrgLen = 100; //The number of pre-trigger points

STR_SAMP_DATA_4CH DataArr[1024];

I32 TimeOutMs = 10000; //10 second timeout

Ret =APS_wait_trigger_sampling( Board_ID, Length, PreTrgLen, TimeOutMs, DataArr );

If( Ret == ERR_NoError )
{
//Sampling successed
// DataArr are ready to used.
}

```

See also:

[APS_set_sampling_param](#); [APS_get_sampling_param](#); [APS_stop_wait_sampling](#);

APS_wait_trigger_sampling_async	Waiting for sample data asynchronously
---------------------------------	--

Descriptions:

This function is used to sample data from controller. This function will return immediately. And create a background thread to sampling the data.

Use [APS_get_sampling_count](#) function to get the count of data be sampled. When the sampled count reaches data **length**, it means sampling finish. If sample count = -1, it means wait failed.

Use [APS_stop_wait_sampling](#) to forced stop the asynchronous wait sampling. The sampling count than will become -1.

Caution:

[APS_wait_trigger_sampling](#), [APS_wait_trigger_sampling_async](#) and [APS_auto_sampling](#) functions cannot be used at the same time.

Syntax:

C/C++:

```
I32 APS_wait_trigger_sampling_async( I32 Board_ID, I32 Length, I32 PreTrgLen, I32 TimeOutMs,  
STR_SAMP_DATA_4CH *DataArr );
```

Visual Basic:

```
APS_wait_trigger_sampling_async(ByVal Board_ID As Long, ByVal Length As Long, ByVal PreTrgLen As  
Long, ByVal TimeOutMs As Long, DataArr As STR_SAMP_DATA_4CH )As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to [APS_initial\(\)](#).

I32 Length: The number of sampling data. (array size)

I32 PreTrgLen: Pre-trigger length.

I32 TimeOutMs: Timeout time. Unit is millisecond.

STR_SAMP_DATA_4CH *DataArr: Get sampling data structure array. Array size must be larger than the parameter "Length".

Return Values:

I32 error code. Refer to error code table.

Example:

```
//... initial card.
```

```
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 2 ); //Set sampling rate  
APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, 0 ); //Set trigger edge (rising edge)  
APS_set_sampling_param( Board_ID, SAMP_PA_LEVEL, 1 ); //Set trigger level ( 1 )
```

```

APS_set_sampling_param( Board_ID, SAMP_PA_TRIGCH, 0 ); //Set trigger channel (channel 0)
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //Set channel_0
sampling source.
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH1, SAMP_MIO_INP ); //Set channel_1
sampling source.

//Start a asynchronous wait sampling.
I32 Length = 1024; //Total sampling data array size.
I32 PreTrgLen = 100; //The number of pre-trigger points
STR_SAMP_DATA_4CH DataArr[1024];
I32 TimeOutMs = 10000; //10 second timeout
I32 Ret;

Ret =APS_wait_trigger_sampling_async( Board_ID, Length, PreTrgLen, TimeOutMs, DataArr );

if( Ret != ERR_NoError )
{
    //Show error message
}
else
{
    while( count < Length )
    {
        APS_get_sampling_count( Board_ID, &count );
        If( count == -1 )
        {
            //Sampling failed,
            // Break program.:
        }

        If( ForceStop )
        {
            APS_stop_wait_sampling(Board_ID);
        }
    }

    If( count == Length )
    { //Sampling successed
        // DataArr are ready to used.
    }
}

```

See also:

[APS_get_sampling_count](#); [APS_wait_trigger_sampling](#); [APS_stop_wait_sampling](#)

APS_get_sampling_count	Get sampled data count.
------------------------	-------------------------

Descriptions:

This function is used to get asynchronous wait sampling data count.

In the first way, start a trigger sampling operation using [APS_wait_trigger_sampling_async](#), user need to get sampling count to check the operation is finish success or failed.

In the second way, start a sampling operation using [APS_auto_sampling](#), user could get sampling count.

Syntax:

C/C++:

I32 APS_get_sampling_count(I32 Board_ID, I32 *SampCnt);

Visual Basic:

APS_get_sampling_count(ByVal Board_ID As Long, SampCnt As Long) As Long

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 *SampCnt: Return sampled data count. If return -1 mean sampling failed.

Return Values:

I32 error code. Refer to error code table.

Example:

Refer to [APS_wait_trigger_sampling_async](#) example.

See also:

APS_set_sampling_param; APS_get_sampling_param; APS_stop_wait_sampling;

APS_wait_trigger_sampling; APS_wait_trigger_sampling_async

APS_stop_wait_sampling	Force stop wait sampling
------------------------	--------------------------

Descriptions:

This function is used to forced stop [APS_wait_trigger_sampling](#) and [APS_wait_trigger_sampling_async](#) function.

Syntax:

C/C++:

```
I32 APS_stop_wait_sampling( I32 Board_ID );
```

Visual Basic:

```
APS_stop_wait_sampling(ByVal Board_ID As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to [APS_initial\(\)](#).

Return Values:

I32 error code. Refer to error code table.

Example:

Refer to [APS_wait_trigger_sampling_async](#) example

See also:

[APS_wait_trigger_sampling](#); [APS_wait_trigger_sampling_async](#)

APS_auto_sampling	Start/Stop auto sampling
-------------------	--------------------------

Descriptions:

This function is used to implement auto sampling operation. It creates a background thread to sample data.

User could use [APS_get_sampling_data](#) / [APS_get_sampling_data_ex](#) to get sampling data and monitor internal buffer status. Four states of the buffer could be monitored, that includes “STOP”, “WORK”, “EMPTY” and “FULL” states.

Use [APS_get_sampling_count](#) function to get the count of data be sampled. If sample count = -1, it means that auto sampling has already stopped.

After enabling, a thread of sampling data start to run and an internal buffer is also built to store sampling data. This buffer is finite space including up to 65535 sampling data, user needs to continuously get sampling data using [APS_get_sampling_data\(\)](#). It is necessary to continuously consume those data of the buffer, and the buffer could be reused to store more sampling data.

Generally speaking, the buffer is always in WORK state. It means that it is impossible to miss any sampling data and the polling frequency of getting data in user side is just suitable.

There is a chance of losing sampling data if the internal buffer is full of data, sampled from DSP side. In FULL state, for example, the sequential sampling data from DSP may be thrown until user gets buffer data to consume parts of data of the buffer.

On the other hand, if the internal buffer is in EMPTY status, it means that getting data form the buffer is faster than sampling from DSP side. User could eliminate frequency of getting data, changing polling timer to slower one. It would increase your CPU performance to do other things.

Caution: These is a set of API functions for auto sampling, including [APS_auto_sampling\(\)](#) and [APS_get_sampling_data\(\)](#) / [APS_get_sampling_data_ex\(\)](#). Don't mix with other trigger functions like [APS_wait_trigger_sampling\(\)](#), [APS_wait_trigger_sampling_async\(\)](#) and [APS_stop_wait_sampling\(\)](#).

Syntax:

C/C++:

```
I32 APS_auto_sampling( I32 Board_ID, I32 StartStop );
```

Visual Basic:

```
APS_auto_sampling (ByVal Board_ID As Long, ByVal StartStop As Long )As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to [APS_initial\(\)](#).

I32 StartStop: 1: Start auto sampling, 0: Stop auto sampling.

Return Values:

I32 error code. Refer to error code table.

Example:

```
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 1 ); //Set sampling rate
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //Set channel_0
sampling source.
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH1, SAMP_MIO_INP ); //Set channel_1
sampling source.

//Start auto sampling.
STR_SAMP_DATA_4CH DataArr[500]; //Be the same size with length
I32 ret;
I32 length = 500; //User specifies a length to get data
I32 retLength = 0; //Physical length of returned data
I32 status; //Monitor buffer state
I32 start = 1;

APS_auto_sampling( Board_ID, start ); //Auto sampling start processing

timer( 10ms )
{
    if( start == 1 )
    {
        Length = 500; //User specifies a length to get data
        APS_get_sampling_data(Board_ID, &length, DataArr, & status ); //return physical length
        // Monitor buffer status
        if(status == 1 ) //buffer is in "WORK" state
        {
            //get data – returned length depends on remain data in buffer
        }
        else if(status == 2 ) //buffer is full
        {
            //get data
            //Sampling data may be lost
        }
        else if(status == 3 ) //buffer is empty
        {
            // get no data – returned length is 0
        }

        For( i=0; i<length; i++ )
        {
            // DataArr are ready to used.
        }
    }
}

APS_auto_sampling( Board_ID, 0 ); //Stop auto sampling
```

See also:

[APS_get_sampling_data](#); [APS_get_sampling_count](#)

APS_get_sampling_data	Get sampling data in auto sampling mode
-----------------------	---

Descriptions:

This function is used to sample data after starting auto sampling. It is also used to monitor sampling status. Refer to [Aps_auto_sampling\(\)](#) for details. Four states are defined below:

State	Description
STOP (0)	Auto sampling stopped
WORK (1)	Auto sampling started
FULL (2)	The internal buffer is full. It is a caution for lost sampling data. Because buffer is full of data, newer sampling data are automatically thrown until user consumes parts of data from buffer.
EMPTY (3)	The internal buffer is empty. Returned length must be zero and get no data

Caution: These is a set of API functions for auto sampling, including [Aps_auto_sampling\(\)](#) and [APS_get_sampling_data\(\)](#). Don't mix with other trigger functions like [APS_wait_trigger_sampling](#), [APS_wait_trigger_sampling_async](#) and [APS_stop_wait_sampling](#).

Syntax:

C/C++:

```
I32 APS_get_sampling_data( I32 Board_ID, I32 *Length, STR_SAMP_DATA_4CH *DataArr, I32
*Status );
```

Visual Basic:

```
APS_get_sampling_data(ByVal Board_ID As Long, Length As Long, DataArr As STR_SAMP_DATA_4CH,
Status As Long )As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 *Length: Bi-direction. User need to specify a maximum size to get data, usually the same with array size of "DataArr". Returned length is physical size of get back sampling data.

STR_SAMP_DATA_4CH *DataArr: Get sampling data structure array. Array size must be equal with or larger than the parameter "*Length".

I32 *Status: The buffer state. 0: STOP state, 1: WORK state, 2: EMPTY state, 3: FULL state.

Return Values:

I32 error code. Refer to error code table.

Example:

```
//... initial card.

APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 1 ); //Set sampling rate
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //Set channel_0
sampling source.
APS_set_sampling_param( Board_ID, SAMP_PA_SRC_CH1, SAMP_MIO_INP ); //Set channel_1
sampling source.

//Start auto sampling.
STR_SAMP_DATA_4CH DataArr[500]; //Be the same size with length
I32 ret;
I32 length = 500; //User specifies a length to get data
I32 retLength = 0; //Physical length of returned data
I32 status; //Monitor buffer state
I32 start = 1;

APS_auto_sampling( Board_ID, start ); //Auto sampling start processing

Timer( 10ms )
{
    If( start == 1 )
    {
        Length = 500; //User specifies a length to get data
        APS_get_sampling_data(Board_ID, &length, DataArr, & status ); //return physical length
                                                               //Monitor buffer status
        If(status == 1 ) //buffer is in "WORK" state
        {
            //get data – returned length depends on remain data in buffer
        }
        Else if(status == 2 ) //buffer is full
        {
            //get data
            //some sampling data may be lost
        }
        Else if(status == 3 ) //buffer is empty
        {
            // get no data – returned length is 0
        }

        For( i=0; i<length; i++ )
        {
            // DataArr are ready to used.
        }
    }
    APS_auto_sampling( Board_ID, 0 ); //Stop auto sampling
}
```

See also:

[APS_auto_sampling](#); [APS_get_sampling_count](#)

APS_set_sampling_param_ex	Set sampling parameter. It is an extension to 8 channels.
---------------------------	---

Descriptions:

This function is used to set sampling parameters at once such as sampling rate, sampling channel source and so on. The related parameters of 8 channels are structured by SAMP_PARAM. There are common settings including sampling rate, sampling edge, sampling level and sampling channel in SAMP_PARAM structure. There are also other settings by channel, including sampling source & axis in SAMP_PARAM structure.

Sampling function is only for the boards have DSP inside. It is for real-time issue. The sampling functions guarantees each sampled point are record under hard realtime environment.

Syntax:

C/C++:

```
I32 APS_set_sampling_param_ex( I32 Board_ID, SAMP_PARAM *Param );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

SAMP_PARAM *Param: A structure for setting sampling parameters

```
typedef struct _SAMP_PARAM
```

```
{
```

```
    I32 rate;      //Sampling rate [ 1 ~ 65535 times of cycle]
```

```
    I32 edge;      //Trigger edge [ 0: Rising edge, 1: Faling edge ]
```

```
    I32 level;     //Trigger level [ -214743648 ~ 2147483647 ]
```

```
    I32 trigCh;    //Trigger channel [ 0 ~ 7 ]
```

```
    I32 sourceByCh[8][2]; //Sampling source by channel, named sourceByCh[a][b],
```

```
        //a: Specify a Channel. Total channels are 8 to be configured.
```

```
        //b: 0: Sampling source, refer to following table 1: Sampling axis
```

```
        //Sampling source: F64 data occupies two channels, I32 data occupies one channel.
```

```
}
```

```
SAMP_PARAM, *PSAMP_PARAM;
```

Source	Symbol Define	Data type	Value range
0x00	SAMP_SRC_COM_POS	I32	command position
0x01	SAMP_SRC_FBK_POS	I32	feedback position
0x02	SAMP_SRC_CMD_VEL	I32	command velocity

0x03	SAMP_SRC_FBK_VEL	I32	feedback velocity
0x04	SAMP_SRC_MIO	I32	motion IO
0x05	SAMP_SRC_MSTS	I32	motion status
0x06	SAMP_SRC_MSTS_ACC	I32	motion status acc
0x07	SAMP_SRC_MSTS_MV	I32	motion status at max velocity
0x08	SAMP_SRC_MSTS_DEC	I32	motion status at dec
0x09	SAMP_SRC_MSTS_CSTP	I32	motion status CSTP
0x0A	SAMP_SRC_MSTS_MDN	I32	motion status MDN
0x0B	SAMP_SRC_MIO_INP	I32	motion status INP
0x0D	SAMP_SRC_MIO_ORG	I32	motion status OGR
0x20	SAMP_SRC_CONTROL_VOL	I32	Control command voltage
0x22	SAMP_SRC_ENCODER_RAW	I32	Encoder raw data
0x23	SAMP_SRC_ERR_POS	I32	Error position
0x10	SAMP_SRC_COM_POS_F64	F64	Command position
0x11	SAMP_SRC_FBK_POS_F64	F64	Feedback position
0x12	SAMP_SRC_CMD_VEL_F64	F64	Command velocity
0x13	SAMP_SRC_FBK_VEL_F64	F64	Feedback velocity
0x14	SAMP_SRC_CONTROL_VOL_F64	F64	Control command voltage
0x15	SAMP_SRC_ERR_POS_F64	F64	Error position

Return Values:

I32 error code. Refer to error code table.

Example:

```
SAMP_PARAM Param;
Param.rate = 1; // Sampling rate
Param.edge = 0; // Rising edge
```

```
Param.level = 1000; // Trigger level
Param.trigCh = 0; //Channel 0
Param.sourceByCh[0][0] = 1; //Set axis 1 to sampling source of channel 0
Param.sourceByCh[0][1] = 0; //Set command position(I32) to sampling source of channel 0
Param.sourceByCh[1][0] = 0; //Set axis 0 to sampling source of channel 1
Param.sourceByCh[1][1] = 1; //Set feedback position(I32) to sampling source of channel 1
//....set other channels including channel 0 to channel 7

I32 Ret = APS_set_sampling_param_ex( Board_ID, &Param ); //Set sampling parameters
...
```

See also:

[APS_get_sampling_param_ex\(\)](#); [APS_wait_trigger_sampling_ex\(\)](#)

APS_get_sampling_param_ex	Get sampling parameter. It is an extension to 8 channels.
---------------------------	---

Descriptions:

This function is used to get sampling parameters at once such as sampling rate, sampling channel source and so on. Refer to APS_set_sampling_param_ex().

Syntax:

C/C++:

```
I32 APS_get_sampling_param_ex( I32 Board_ID, SAMP_PARAM *Param );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

SAMP_PARAM *Param: A structure for setting sampling parameters

Return Values:

I32 error code. Refer to error code table.

Example:

```
SAMP_PARAM Param;
```

```
Ret = APS_get_sampling_param_ex( Board_ID, &Param ); //Get all paramters
```

```
...
```

See also:

[APS_set_sampling_param_ex\(\);](#)[APS_wait_trigger_sampling_ex\(\)](#)

APS_wait_trigger_sampling_ex	Waiting for sample data. It is an extension to 8 channels.
------------------------------	--

Descriptions:

This function is used to sample data from controller. When the function is issued, the program stating to sample the information and put the data to the internal buffer. Until the trigger signal is turned on, program fetched a mass of data which size is pre-trigger length from internal buffer to the user's data buffer and continuous sample the data until reach the length that users designated. In other hand, if the timeout time is reached and the trigger signal does not raised, this function will be timeout and return an error message.

Use [APS_stop_wait_sampling](#) to forced stop the wait sampling

Caution:

[APS_wait_trigger_sampling_ex](#), [APS_wait_trigger_sampling_async_ex](#) and [APS_auto_sampling_ex](#) functions cannot be used at the same time.

Syntax:

C/C++:

```
I32 APS_wait_trigger_sampling_ex( I32 Board_ID, I32 Length, I32 PreTrgLen, I32 TimeOutMs,
STR_SAMP_DATA_8CH *DataArr );
```

Visual Basic:

```
APS_wait_trigger_sampling_ex(ByValBoard_ID As Long, ByVal Length As Long, ByVal PreTrgLen As
Long, ByVal TimeOutMs As Long, DataArr As STR_SAMP_DATA_8CH ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to [APS_initial\(\)](#).

I32 Length: The number of sampling data. (array size)

I32 PreTrgLen: Pre-trigger length.

I32 TimeOutMs: Timeout time. Unit is millisecond.

STR_SAMP_DATA_8CH *DataArr: Get sampling data structure array. Array size must be larger than the parameter "Length".

Return Values:

I32 error code. Refer to error code table.

Example:

```
//... initial card.
```

```
SAMP_PARAM Param;
```

```

Param.rate = 1; // Sampling rate
Param.edge = 0; // Rising edge
Param.level = 1000; // Trigger level
Param.trigCh = 0; //Channel 0
Param.sourceByCh[0][0] = 1; //Set axis 1 to sampling source of channel 0
Param.sourceByCh[0][1] = 0; //Set command position(I32) to sampling source of channel 0
Param.sourceByCh[1][0] = 0; //Set axis 0 to sampling source of channel 1
Param.sourceByCh[1][1] = 1; //Set feedback position(I32) to sampling source of channel 1
//....set other channels including channel 0 to channel 7

I32 Ret = APS_set_sampling_param_ex( Board_ID, &Param ); //Set sampling parameters

I32 Length = 1024; //Total sampling data array size.
I32 PreTrgLen = 100; //The number of pre-trigger points
STR_SAMP_DATA_8CH DataArr[1024];
I32 TimeOutMs = 10000; //10 second timeout

Ret =APS_wait_trigger_sampling_ex( Board_ID, Length, PreTrgLen, TimeOutMs, &DataArr );
If( Ret == ERR_NoError )
{
//Sampling successed
// DataArr are ready to used.
}

```

See also:

[APS_set_sampling_param_ex](#); [APS_get_sampling_param_ex](#); [APS_stop_wait_sampling_ex](#);

APS_wait_trigger_sampling_async_ex	Waiting for sample data asynchronously. It is an extension to 8 channels.
------------------------------------	---

Descriptions:

This function is used to sample data from controller. This function will return immediately. And create a background thread to sampling the data.

Use [APS_get_sampling_count](#) function to get the count of data be sampled. When the sampled count reaches data **length**, it means sampling finish. If sample count = -1, it means wait failed.

Use [APS_stop_wait_sampling](#) to forced stop the asynchronous wait sampling. The sampling count than will become -1.

Caution:

[APS_wait_trigger_sampling_ex](#), [APS_wait_trigger_sampling_async_ex](#) and [APS_auto_sampling_ex](#) functions cannot be used at the same time.

Syntax:

C/C++:

```
I32 APS_wait_trigger_sampling_async_ex( I32 Board_ID, I32 Length, I32 PreTrgLen, I32 TimeOutMs,
STR_SAMP_DATA_8CH *DataArr );
```

Visual Basic:

```
APS_wait_trigger_sampling_async_ex(ByVal Board_ID As Long, ByVal Length As Long, ByVal PreTrgLen
As Long, ByVal TimeOutMs As Long, DataArr As STR_SAMP_DATA_8CH )As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to [APS_initial\(\)](#).

I32 Length: The number of sampling data. (array size)

I32 PreTrgLen: Pre-trigger length.

I32 TimeOutMs: Timeout time. Unit is millisecond.

STR_SAMP_DATA_8CH *DataArr: Get sampling data structure array. Array size must be larger than the parameter "Length".

Return Values:

I32 error code. Refer to error code table.

Example:

//... initial card.

SAMP_PARAM Param;

```

Param.rate = 1; // Sampling rate
Param.edge = 0; // Rising edge
Param.level = 1000; // Trigger level
Param.trigCh = 0; //Channel 0
Param.sourceByCh[0][0] = 1; //Set axis 1 to sampling source of channel 0
Param.sourceByCh[0][1] = 0; //Set command position(l32) to sampling source of channel 0
Param.sourceByCh[1][0] = 0; //Set axis 0 to sampling source of channel 1
Param.sourceByCh[1][1] = 1; //Set feedback position(l32) to sampling source of channel 1
//....set other channels including channel 0 to channel 7

I32 Ret = APS_set_sampling_param_ex( Board_ID, &Param ); //Set sampling parameters

//Start a asynchronous wait sampling.
I32 Length = 1024; //Total sampling data array size.
I32 PreTrgLen = 100; //The number of pre-trigger points
STR_SAMP_DATA_8CH DataArr[1024];
I32 TimeOutMs = 10000; //10 second timeout
I32 Ret;

Ret =APS_wait_trigger_sampling_async_ex( Board_ID, Length, PreTrgLen, TimeOutMs, DataArr );

if( Ret != ERR_NoError )
{
    //Show error message
}else
{
    while( count < Length )
    {
        APS_get_sampling_count( Board_ID, &count );
        If( count == -1 )
        {
            //Sampling failed,
            // Break program.;

        }

        If( ForceStop )
        {
            APS_stop_wait_sampling(Board_ID);
        }
    }
    If( count == Length )
    { //Sampling successed
        // DataArr are ready to used.
    }
}

```

See also:

[APS_get_sampling_count](#); [APS_wait_trigger_sampling_ex](#); [APS_stop_wait_sampling](#)

APS_get_sampling_data_ex	Get sampling data in auto sampling mode. It is an extension to 8 channels.
--------------------------	--

Descriptions:

This function is used to sample data after starting auto sampling. It is also used to monitor sampling status. Refer to [APS_auto_sampling\(\)](#) for details. Four states are defined below:

State	Description
STOP (0)	Auto sampling stopped
WORK (1)	Auto sampling started
EMPTY (2)	The internal buffer is empty. Returned length must be zero and get no data
FULL (3)	The internal buffer is full. It is a caution for lost sampling data. Because buffer is full of data, newer sampling data are automatically thrown until user consumes parts of data from buffer.

Caution: These is a set of API functions for auto sampling, including [APS_auto_sampling\(\)](#) and [APS_get_sampling_data_ex\(\)](#). Don't mix with other trigger functions like [APS_wait_trigger_sampling_ex](#), [APS_wait_trigger_sampling_async_ex](#) and [APS_stop_wait_sampling](#).

Syntax:

C/C++:

```
I32 APS_get_sampling_data_ex( I32 Board_ID, I32 *Length, STR_SAMP_DATA_8CH *DataArr, I32
*Status );
```

Visual Basic:

```
APS_get_sampling_data_ex(ByVal Board_ID As Long, Length As Long, DataArr As
STR_SAMP_DATA_8CH, Status As Long )As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 *Length: Bi-direction. User need to specify a maximum size to get data, usually the same with array size of "DataArr". Returned length is physical size of get back sampling data.

STR_SAMP_DATA_8CH *DataArr: Get sampling data structure array. Array size must be equal with or larger than the parameter "*Length".

I32 *Status: The buffer state. 0: STOP state, 1: WORK state, 2: EMPTY state, 3: FULL state.

Return Values:

I32 error code. Refer to error code table.

Example:

```
//... initial card.  
SAMP_PARAM Param;  
  
Param.rate = 1; // Sampling rate  
Param.edge = 0; // Rising edge  
Param.level = 1000; // Trigger level  
Param.trigCh = 0; //Channel 0  
Param.sourceByCh[0][0] = 1; //Set axis 1 to sampling source of channel 0  
Param.sourceByCh[0][1] = 0; //Set command position(I32) to sampling source of channel 0  
Param.sourceByCh[1][0] = 0; //Set axis 0 to sampling source of channel 1  
Param.sourceByCh[1][1] = 1; //Set feedback position(I32) to sampling source of channel 1  
//....set other channels including channel 0 to channel 7  
  
I32 Ret = APS_set_sampling_param_ex( Board_ID, &Param ); //Set sampling parameters  
  
//Start auto sampling.  
STR_SAMP_DATA_8CH DataArr[500]; //Be the same size with length  
I32 ret;  
I32 length = 500; //User specifies a length to get data  
I32 retLength = 0; //Physical length of returned data  
I32 status; //Monitor buffer state  
I32 start = 1;  
  
APS_auto_sampling( Board_ID, start ); //Auto sampling start processing  
  
Timer( 10ms )  
{  
    If( start == 1 )  
    {  
        Length = 500; //User specifies a length to get data  
        APS_get_sampling_data_ex(Board_ID, &length, DataArr, & status ); //return physical length  
                                            //Monitor buffer status  
        If(status == 1 ) //buffer is in "WORK" state  
        {  
            //get data – returned length depends on remain data in buffer  
        }  
        Else if(status == 2 ) //buffer is full  
        {  
            //get data  
            //some sampling data may be lost  
        }  
        Else if(status == 3 ) //buffer is empty  
        {  
            // get no data – returned length is 0  
        }  
    }  
}
```

```
For( i=0; i<length; i++ )
{
    // DataArr are ready to used.
}
}
APS_auto_sampling( Board_ID, 0 ); //Stop auto sampling
```

See also:

[APS_auto_sampling](#); [APS_get_sampling_count](#)

12.DIO & AIO

APS_write_d_output	Set digital output value
--------------------	--------------------------

Descriptions:

This function is use to access on board general purpose digital output. If the channels are more than 32, users must assign a group number to access more I/O. The PCI-8254/58 has 24 output channels, user can assign group number to be constant 0.

Do channel defined by bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

Syntax:

C/C++:

```
I32 APS_write_d_output(I32 Board_ID, I32 DO_Group, I32 DO_Data);
```

Visual Basic:

```
APS_write_d_output (ByVal Board_ID As Long, ByVal DO_Group As Long, ByVal DO_Data as Long) As  
Long;
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 DO_Group: The digit output group number.

I32 DO_Data: The digit output data (Data type is bit type).

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 DO_Group = 0; // If DO channel less than 32
```

```
I32 DO_Data = 0x000F; // Assign bit 0,1,2,3 output.
```

```
I32 returnCode; // Function return code
```

```
returnCode = APS_write_d_output( Board_ID, DO_Group, DO_Data );
```

```
if( returnCode != 0 )
```

```
    MessageBox( "Set digit output function failed" );
```

See also:

`APS_read_d_input()`

APS_read_d_output	Read digital output value
-------------------	---------------------------

Descriptions:

This function is use to get on board general purpose digital output. If the channels are more than 32, users must assign a group number to access more I/O. The PCI-8254/58 has 24 output channels, user can assign group number to be constant 0.

Do channel defined by bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

Syntax:

C/C++:

```
I32 APS_read_d_output(I32 Board_ID, I32 DO_Group, I32 *DO_Data);
```

Visual Basic:

```
APS_read_d_output (ByVal Board_ID As Long, ByVal DO_Group As Long, DO_Data as Long) As Long;
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 DO_Group: The digit output group number.

I32 *DO_Data: The digit output data (Data type is bit type).

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 DO_Group = 0;           // If DO channel less than 32
I32 DO_Data = 0;           // Do data
I32 returnCode;             // Function return code
```

```
returnCode = APS_read_d_output( Board_ID, DO_Group, &DO_Data );
if( returnCode != 0 )
    MessageBox( "Get digit output function failed" );
```

See also:

[APS_write_d_output\(\)](#)

APS_read_d_input	Read digital input value
------------------	--------------------------

Descriptions:

This function is use to get on board general purpose digital input. If the channels are more than 32, users must assign a group number to access more I/O. The PCI-8254/58 has 24 input channels, user can assign group number to be constant 0.

Do channel defined by bit:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DIO
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

Syntax:

C/C++:

```
I32 APS_read_d_input(I32 Board_ID, I32 DI_Group, I32 *DI_Data);
```

Visual Basic:

```
APS_read_d_input (ByVal Board_ID As Long, ByVal DI_Group As Long, DI_Data as Long) As Long;
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 DI_Group: The digit input group number.

I32 *DI_Data: The returned digit input data

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 DI_Group = 0;           // If DI channel less than 32
I32 DI_Data = 0;           // Di data
I32 returnCode;            // Function return code
```

```
returnCode = APS_read_d_input( Board_ID, DI_Group, &DI_Data );
if( returnCode != 0 )
    MessageBox( "Get digit input function failed" );
```

See also:

[APS_write_d_output\(\)](#)

APS_write_d_channel_output	Set digital output value by channel
----------------------------	-------------------------------------

Descriptions:

This function is used to access on board general purpose digital output by channel. If those channels are more than 32, users must assign a group number to access more I/O. The PCI-8254/58 has 24 output channels in the group number 0.

The definition of Do channels in the group number 0:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ch 0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

Syntax:

C/C++:

```
I32 APS_write_d_channel_output(I32 Board_ID, I32 DO_Group, I32 Ch_No, I32 DO_Data);
```

Visual Basic:

```
APS_write_d_channel_output (ByVal Board_ID As Long, ByVal DO_Group As Long, ByVal Ch_No as Long, ByVal DO_Data as Long) As Long;
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 DO_Group: The digit output group number. Set to 0 on PCI-8254/8.

I32 Ch_No: The digit output channel(0~23)

I32 DO_Data: The digit output data by channel (0~1).

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 returnCode; // Function return code
//Turn on Do output channel 3 in group number 0
returnCode = APS_write_d_channel_output( Board_ID, 0, 3, 1 );
if( returnCode != 0 )
    MessageBox( "Set digit channel output function failed" );
```

See also:

[APS_read_d_channel_input\(\)](#)

APS_read_d_channel_output	Read digital output value
---------------------------	---------------------------

Descriptions:

This function is used to access on board general purpose digital output by channel. If those channels are more than 32, users must assign a group number to access more I/O. The PCI-8254/58 has 24 output channels in the group number 0.

The definition of Do channels in the group number 0:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ch 0
TTL7	TTL6	TTL5	TTL4	TTL3	TTL2	TTL1	TTL0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								TTL15	TTL14	TTL13	TTL12	TTL11	TTL10	TTL9	TTL8

Syntax:

C/C++:

```
I32 APS_read_d_channel_output(I32 Board_ID, I32 DO_Group, I32 Ch_No, I32 *DO_Data);
```

Visual Basic:

```
APS_read_d_channel_output (ByVal Board_ID As Long, ByVal DO_Group As Long, ByVal Ch_No as Long, DO_Data as Long) As Long;
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 DO_Group: The digit output group number. Set to 0 on PCI-8254/8.

I32 Ch_No: The digit output channel(0~23)

I32 *DO_Data: The digit output data

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 DO_Data = 0;           // Do data
I32 returnCode;           // Function return code
//Get status of Do output channel 3 in group number 0
returnCode = APS_read_d_channel_output( Board_ID, 0, 3, &DO_Data );
if( returnCode != 0 )
    MessageBox( "Get digit channel output function failed" );
```

See also:

[APS_write_d_channel_output\(\)](#)

APS_read_a_input_value	Read back analog input value by volt
------------------------	--------------------------------------

Descriptions:

This function is used to get on board general purpose analog input value, and the analog input value unit is volt. It could be voltage or current value.

Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_read_a_input_value(I32 Board_ID, I32 Channel_No, F64 *Convert_Data);
```

Visual Basic:

```
APS_read_a_input_value(ByVal Board_ID As Long, ByVal Channel_No As Long, Convert_Data as Double) As Long;
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Channel_No: The channel number. Range is from 0 to 65535.

F64 *Convert_Data: The returned converted analog data. Unit is volt and range is -10V to 10V.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 Board_ID = 0;  
I32 Channel_No = 0;  
F64 Convert_Data = 0.0;  
I32 returnCode;           // Function return code  
  
returnCode = APS_read_a_input_value( Board_ID, Channel_No, & Convert_Data );  
if( returnCode != 0 )  
    MessageBox( "Get analog input function failed" );
```

See also:

[APS_write_a_output_value\(\)](#)

APS_write_a_output_value	Set analog output value by volt
--------------------------	---------------------------------

Descriptions:

This function is used to access on board general purpose analog output raw data of one axis and the analog output value unit is volt. It could be voltage or current value. Please make sure axis **servo on signal is turn off** relative to channel number before use analog output function.

Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_write_a_output_value(I32 Board_ID, I32 Channel_No, F64 Convert_Data);
```

Visual Basic:

```
APS_write_a_output_value (ByVal Board_ID As Long, ByVal Channel_No As Long, ByVal Convert_Data  
as Double) As Long;
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Channel_No: The channel number. Range is from 0 to 65535.

F64 Convert_Data: The converted analog data to be output. Unit is volt and range is -10V to 10V

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 Channel_No = 1;           // Assign channel 1 to be output channel  
F32 Convert_Data = 5.2;      //Output 5.2 volt  
I32 returnCode;              // Function return code
```

```
returnCode = APS_write_a_output_value( Board_ID, Channel_No, Convert_Data );  
if( returnCode != 0 )  
    MessageBox( "Write analog output function failed" );
```

See also:

[APS_read_a_input_value\(\)](#)

13. Point table motion

APS_set_feeder_group	Set axes into a feeder group
----------------------	------------------------------

Descriptions:

This function is used to set axes into a feeder group. Before you used any other feeder function, you should assign some axes to a feeder group. When you no longer use the feeder, you should free the group by *APS_free_feeder_group()* function.

Note:

The current feeder only supports two dimension axis ID group.

Syntax:

C/C++:

```
I32 APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
```

Visual Basic:

```
APS_set_feeder_group(ById GroupId As Long, ByVal Dimension As Long, Axis_ID_Array As Long) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

I32 Dimension: The dimension of the axis ID array. Value range: 1~4

I32 *Axis_ID_Array: The Axis ID array from 0 to 65535. The array size must match the axis dimension.

The axis-ID in Axis_ID_Array[0] represent as the control axis which must the minimum ID number in the array.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
#include "APS168.h"
#include "ErrorCodeDef.h"

I32 ret; // Return code
I32 groupId = 0; // Feeder group ID [0,1]
I32 runIdx; // Which index of data is in operation
I32 fedIdx; // How much data is loaded into feeder module.
I32 msts; // Motion status
I32 dim = 2; // Group dimension
I32 ax[2] = { 0, 1, }; // Axes ID array
```

```

PNT_DATA_2D* pPnt  = NULL;    // Pointer of PNT_DATA_2D

ret = APS_set_feeder_group( groupId, dim, ax );
if( ret != ERR_NoError ) { //Exception handling }

ret = APS_reset_feeder_buffer(groupId );

ret = APS_set_feeder_point_2D(groupId, pPnt, cnt, 1 ); //or APS_set_feeder_point_2D_F64()
if( ret != ERR_NoError ) { //Exception handling }

// Start feeder and point table move
ret = APS_start_feeder_move( groupId );
if( ret != ERR_NoError ) { //Exception handling }

// Check whether the end of the point table move procedure
{
    ret = APS_get_feeder_running_index(groupId, &runIdx);
    if( ret != ERR_NoError ) break;
    ret = APS_get_feeder_feed_index(groupId, &feedIdx);
    if( ret != ERR_NoError ) break;
    msts = APS_motion_status( ax [0] );
    // Check motion status.

}while( runIdx != ( feedIdx -1 ) );

ret = APS_free_feeder_group(groupId);
if( ret != ERR_NoError ) { //Exception handling }

```

See also:

```

I32 APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 APS_free_feeder_group( I32 GroupId );
I32 APS_reset_feeder_buffer( I32 GroupId );
I32 APS_set_feeder_point_2D ( I32 GroupId, POINT_DATA_2D* PtArray, I32 Size, I32 LastFlag );
I32 APS_start_feeder_move( I32 GroupId );
I32 APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 APS_get_feeder_feed_index( I32 GroupId, I32 *Index );

```

APS_get_feeder_group	Return the configuration in one feeder group
----------------------	--

Descriptions:

This function is used to get the configuration of a specified feeder. The configuration include group dimension and which axis IDs in group.

Syntax:

C/C++:

I32 APS_get_feeder_group(I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array);

Visual Basic:

APS_get_feeder_group (ByVal GroupId As Long, Dimension As Long, Axis_ID_Array As Long) As Long

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

I32 *Dimension: Return group axes dimension. Possible return value [0~4].

I32 *Axis_ID_Array: Return the Axis ID from 0 to 65535. Please give a array of **constant size 4**.

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to the example of *APS_set_feeder_group()*

See also:

I32 APS_set_feeder_group(I32 GroupId, I32 Dimension, I32 *Axis_ID_Array);

I32 APS_free_feeder_group(I32 GroupId);

I32 APS_reset_feeder_buffer(I32 GroupId);

I32 APS_set_feeder_point_2D(I32 GroupId, POINT_DATA_2D* PtArray, I32 Size, I32 LastFlag);

I32 APS_start_feeder_move(I32 GroupId);

I32 APS_get_feeder_running_index(I32 GroupId, I32 *Index);

I32 APS_get_feeder_feed_index(I32 GroutId, I32 *Index);

APS_free_feeder_group	Free a feeder group and it's resources
-----------------------	--

Descriptions:

This function is used to free the axes from the feeder and free its resources. When you no longer to use the feeder, you must use this function to release the resources or it will keep the resources until the process be terminated.

Syntax:

C/C++:

```
I32 APS_free_feeder_group( I32 GroupId );
```

Visual Basic:

```
APS_free_feeder_group( ByVal GroupId As Long) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to the example of *APS_set_feeder_group()*

See also:

```
I32 APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 APS_reset_feeder_buffer( I32 GroupId );
I32 APS_set_feeder_point_2D( I32 GroupId, POINT_DATA_2D* PtArray, I32 Size, I32 LastFlag );
I32 APS_start_feeder_move( I32 GroupId );
I32 APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_reset_feeder_buffer	Reset the feeder's point buffer
-------------------------	---------------------------------

Descriptions:

This function is used to reset the 2D point table data buffer of a feeder.

Note:

1. When feeder is loading the data to controller, you cannot use this function to reset the feeder buffer.
2. When issue the *APS_set_feeder_point_[n]D()* and the LastFlag is set. Use this function to reset the buffer and clear LastFlag.

Syntax:

C/C++:

```
I32 APS_reset_feeder_buffer( I32 GroupId );
```

Visual Basic:

```
APS_reset_feeder_buffer ( ByVal GroupId As Long ) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to the example of *APS_set_feeder_group()*

See also:

```
I32 APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 APS_free_feeder_group( I32 GroupId );
I32 APS_set_feeder_point_2D( I32 GroupId, POINT_DATA_2D* PtArray, I32 Size, I32 LastFlag );
I32 APS_start_feeder_move( I32 GroupId );
I32 APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_set_feeder_point_2D	Add a point into feeder's buffer
-------------------------	----------------------------------

Descriptions:

This function is used to set two dimension trajectory data into the buffer of a feeder. The parameter “LastFlag” must be set when the last piece of trajectory data is set. After “LastFlag” is be set, the function “APS_start_feeder_move()” can be execute. When “LastFlag” is set, the trajectory data cannot be set into buffer until *APS_reset_feeder_buffer()* is called.

Syntax:

C/C++:

```
I32 APS_set_feeder_point_2D( I32 GroupId, PNT_DATA_2D * PtArray, I32 Size, I32 LastFlag );
```

Visual Basic:

```
APS_set_feeder_point_2D ( ByVal GroupId As Long, PtArray As PNT_DATA_2D, ByVal Size As Long,  
ByVal LastFlag As Long ) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

PNT_DATA_2D* PtArray: Two dimension trajectory information array.

I32 Size: PNT_DATA_2D array size. Value must large than 0. (Size > 0)

I32 LastFlag: Last point data flag. To notice the feeder the point array is the last one for feeder.

0: Not the last one

1: Last one.

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to the example of *APS_set_feeder_group()*

See also:

I32 APS_set_feeder_group(I32 GroupId, I32 Dimension, I32 *Axis_ID_Array);

I32 APS_get_feeder_group(I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array);

I32 APS_free_feeder_group(I32 GroupId);

I32 APS_reset_feeder_buffer(I32 GroupId);

I32 APS_start_feeder_move(I32 GroupId);

I32 APS_get_feeder_running_index(I32 GroupId, I32 *Index);

I32 APS_get_feeder_feed_index(I32 GroutId, I32 *Index);

APS_set_feeder_point_2D_ex	Add a point into feeder's buffer
----------------------------	----------------------------------

Descriptions:

This function is used to set two dimension trajectory data into the buffer of a feeder. The parameter “LastFlag” must be set when the last piece of trajectory data is set. After “LastFlag” is be set, the function “APS_start_feeder_move()” can be execute. When “LastFlag” is set, the trajectory data cannot be set into buffer until *APS_reset_feeder_buffer()* is called.

Caution:

APS_set_feeder_point_2D_ex() and APS_set_feeder_point_2D() functions cannot be used at the same time. APS_set_feeder_point_2D_ex() is used by F64 type, and APS_set_feeder_point_2D() is used by I32 type.

Syntax:

C/C++:

```
I32 APS_set_feeder_point_2D_ex( I32 GroupId, PNT_DATA_2D_F64 * PtArray, I32 Size, I32 LastFlag );
```

Visual Basic:

```
APS_set_feeder_point_2D_ex( ByVal GroupId As Long, PtArray As PNT_DATA_2D_F64, ByVal Size As Long, ByVal LastFlag As Long ) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

PNT_DATA_2D_F64* PtArray: Two dimension trajectory information array.

I32 Size: PNT_DATA_2D_F64 array size. Value must large than 0. (Size > 0)

I32 LastFlag: Last point data flag. To notice the feeder the point array is the last one for feeder.

0: Not the last one

1: Last one.

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to the example of *APS_set_feeder_group()*

See also:

I32 APS_set_feeder_group(I32 GroupId, I32 Dimension, I32 *Axis_ID_Array);

I32 APS_get_feeder_group(I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array);

```
I32 APS_free_feeder_group( I32 GroupId );
I32 APS_reset_feeder_buffer( I32 GroupId );
I32 APS_start_feeder_move( I32 GroupId );
I32 APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_start_feeder_move	Start point table move and feed points.
-----------------------	---

Descriptions:

The following items will be executed when this function is issued.

1. Load points into controller (Point table).
2. Start point table move.

This function will fail when the parameter “LastFlag” of function *APS_set_feeder_point_[n]D()* does not be set.

Syntax:

C/C++:

I32 APS_start_feeder_move(I32 GroupId);

Visual Basic:

APS_start_feeder_move (ByVal GroupId As Long) As Long

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to the example of *APS_set_feeder_group()*

See also:

```
I32 APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 APS_free_feeder_group( I32 GroupId );
I32 APS_reset_feeder_buffer( I32 GroupId );
I32 APS_set_feeder_point_2D ( I32 GroupId, PNT_DATA_2D * PtArray, I32 Size, I32 LastFlag );
I32 APS_get_feeder_running_index( I32 GroupId, I32 *Index );
I32 APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

APS_get_feeder_status	Get status of feeder
-----------------------	----------------------

Descriptions:

User could monitor status of a feeder, including feeder states and feeder error code.

There are three states of a feeder as following:

0: Feeder_Stop: Feeder is stopped.

1: Feeder_Run: Feeder is running.

2: Feeder_Pause: Feeder is paused by `APS_set_feeder_ex_pause()`.

Error code refers to [APS Functions Return Code](#).

Syntax:

C/C++:

`I32 APS_get_feeder_status(I32 GroupId, I32 *State, I32 *ErrCode);`

Visual Basic:

`APS_get_feeder_status(ByVal GroupId As Long, State As Long, ErrCode As Long) As Long`

Parameters:

`I32 GroupId`: Group ID. Value range: 0~1.

`I32 *State`: State of a feeder

0: Feeder_Stop: Feeder is stopped.

1: Feeder_Run: Feeder is running.

2: Feeder_Pause: Feeder is paused.

`I32 *ErrCode`: runtime error code of a feeder. Refer to [APS Functions Return Code](#).

Return Values:

`I32` Error code: Please refer to error code table.

Example:

```
I32 state = 0;
```

```
I32 errorCode = 0;
```

```
I32 ret = 0;
```

```
//Get feeder status
```

```
ret = APS_get_feeder_status( 0, &state, &errorCode );
```

See also:

APS_start_feeder_move()

APS_get_feeder_running_index	Get which point is in operation.
------------------------------	----------------------------------

Descriptions:

This function is used to observe which buffer index currently the controller being processed. The index of the buffer is the array index you feed to buffer.

This function is similar with *APS_get_running_point_index()*, but the different is the order of the index.

APS_get_running_point_index() return point table index which order by point table itself in peration

‘s ram; *APS_get_feeder_running_index()* return buffer index which order by feeder ‘s buffer in host ‘s ram.

Syntax:

C/C++:

```
I32 APS_get_feeder_running_index( I32 GroupId, I32 *Index );
```

Visual Basic:

```
APS_get_feeder_running_index ( ByVal GroupId As Long, Index As Long ) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

I32 *Index: Return which point is in operation.

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to the example of *APS_set_feeder_group()*

See also:

```
I32 APS_set_feeder_group( I32 GroupId, I32 Dimension, I32 *Axis_ID_Array );
I32 APS_get_feeder_group( I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array );
I32 APS_free_feeder_group( I32 GroupId );
I32 APS_reset_feeder_buffer( I32 GroupId );
I32 APS_set_feeder_point_2D ( I32 GroupId, PNT_DATA_2D * PtArray, I32 Size, I32 LastFlag );
I32 APS_start_feeder_move( I32 GroupId );
I32 APS_get_feeder_feed_index( I32 GroupId, I32 *Index );
```

APS_get_feeder_feed_index	Get which point is set into point table.
---------------------------	--

Descriptions:

This function will return which the latest buffer index in feeder is loaded into controller.

Syntax:

C/C++:

```
I32 APS_get_feeder_feed_index( I32 GroutId, I32 *Index );
```

Visual Basic:

```
APS_get_feeder_feed_index ( ByVal GroupId As Long, Index As Long ) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

I32 *Index: Return which buffer index is load into controller.

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to the example of *APS_set_feeder_group()*

See also:

I32 APS_set_feeder_group(I32 GroupId, I32 Dimension, I32 *Axis_ID_Array);

I32 APS_get_feeder_group(I32 GroupId, I32 *Dimension, I32 *Axis_ID_Array);

I32 APS_free_feeder_group(I32 GroupId);

I32 APS_reset_feeder_buffer(I32 GroupId);

I32 APS_set_feeder_point_2D(I32 GroupId, PNT_DATA_2D * PtArray, I32 Size, I32 LastFlag);

I32 APS_start_feeder_move(I32 GroupId);

I32 APS_get_feeder_running_index(I32 GroupId, I32 *Index);

APS_set_feeder_ex_pause	Motion paused(stopped) and feeder paused
-------------------------	--

Descriptions:

This function is used to pauses move when running point table. When pause command is issued, it will decelerate to stop and turn off I/O. The feeder also will be paused at the same time.

Syntax:

C/C++:

```
I32 APS_set_feeder_ex_pause( I32 GroupId );
```

Visual Basic:

```
APS_set_feeder_ex_pause ( ByVal GroupId As Long ) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
#include "APS168.h"
#include "ErrorCodeDef.h"

//When push pause button on user interface.
I32 ret;
I32 groupId = 0;
ret = APS_set_feeder_ex_pause( groupId );
if( ret != ERR_NoError ) {//Exception handling }
// Check the motion status has stopped.
...
```

See also:

```
I32 APS_set_feeder_ex_pause( I32 GroupId );
I32 APS_set_feeder_ex_rollback( I32 GroupId, I32 Max_Speed );
I32 APS_set_feeder_ex_resume( I32 GroupId );
```

APS_set_feeder_ex_rollback	Move back to the starting position of paused index
----------------------------	--

Descriptions:

This function is used to let the group of axes back to the last point position which is paused by *APS_set_feeder_ex_pause()*.

This function can **ONLY** be called after *APS_set_feeder_ex_pause()*. The behavior is not defined when this function is used in other situations.

Syntax:

C/C++:

```
I32 APS_set_feeder_ex_rollback( I32 GroupId, I32 Max_Speed );
```

Visual Basic:

```
APS_set_feeder_ex_rollback( ByVal GroupId As Long, ByVal Max_Speed As Long ) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

I32 Max_Speed: Maximum linear interpolation speed. Value > 0, Unit: pulse/sec.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
#include "APS168.h"
#include "ErrorCodeDef.h"

//When push "Go back" button on user interface.
I32 ret;
I32 groupId = 0;
I32 max_speed = 5000; // Pulse/sec
ret = APS_set_feeder_ex_rollback( groupId, max_speed );
if( ret != ERR_NoError ) { //Exception handling
    // Check the motion status has done.
    ...
}
```

See also:

```
I32 APS_set_feeder_ex_pause( I32 GroupId );
I32 APS_set_feeder_ex_resume( I32 GroupId );
```

APS_set_feeder_ex_resume	Resume the point-table move.
--------------------------	------------------------------

Descriptions:

This function is used to resume move from paused feeder running index. When passing through the pause position, it will keep I/O status.

This function can **ONLY** be called after *APS_set_table_move_ex_rollback()*. The behavior is not defined when this function is be used in other situation.

Syntax:

C/C++:

```
I32 APS_set_feeder_ex_resume ( I32 GroupId );
```

Visual Basic:

```
APS_set_feeder_ex_resume ( ByVal GroupId As Long ) As Long
```

Parameters:

I32 GroupId: Group ID. Value range: 0~1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
#include "APS168.h"
#include "ErrorCodeDef.h"

//When push "Resume" button on user interface.
I32 ret;
I32 groupId = 0;
ret = APS_set_feeder_ex_resume ( groupId );
if( ret != ERR_NoError ) {//Exception handling }
// Check the motion status has started.
...
```

See also:

```
I32 APS_set_feeder_ex_pause( I32 GroupId );
I32 APS_set_feeder_ex_rollback( I32 GroupId, I32 Max_Speed );
```

14. Advanced point table

APS_pt_enable	Enable point table.
---------------	---------------------

Descriptions:

This function is used to enable a point table. User could set related axes of board to specified point table. In a specified board, it is forbidden to set repeat axis to point table.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_enable( I32 Board_ID, I32 PtId, I32 Dimension, I32 *AxisArr );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

I32 Dimension:

I32 *AxisArr: the axis array of a specified board is from 0 to N.

For PCI-8254, it is from 0 to 3.

For PCI-8258, it is from 0 to 7.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0
I32 Dimension = 2; //2D Dimension
I32 AxisArr[2] = { 0, 1 }; //Set Axis 0 & Axis 1 to point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
ret = APS_pt_enable(Board_ID , PtId, Dimension, & AxisArr ); //Enable point table 0
```

See also:

APS_pt_disable	Disable point table.
----------------	----------------------

Descriptions:

This function is used to disable a point table.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_disable ( I32 Board_ID, I32 PtId );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0

//Disable
ret = APS_pt_disable(Board_ID , PtId ); //Disable point table 0
```

See also:

APS_get_pt_info	Get information of point table.
-----------------	---------------------------------

Descriptions:

This function is used to get information of point table. User could get information of dimension and axis array. If point table is disabled, the return information of dimension is defined to 0.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_get_pt_info( I32 Board_ID, I32 PtId, PPTINFO Info );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTINFO Info: A structure pointer for getting information of point table

typedef struct

{

 I32 Dimension; //How many dimension in spcfied point table

 I32 AxisArr[6]; //Axis array of point talbe. Maximun to 6 axes depended on dimension.

} PTINFO, *PPTINFO;

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0
PTINFO Info;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
ret = APS_get_pt_info(Board_ID , PtId, &Info ); //Get information of point table 0
....
```

See also:

APS_pt_set_vs	Set configuration of Vs to point table
---------------	--

Descriptions:

This function is used to set started speed (Vs) to point table. When point table is moving, Vs is only applied to first point.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_vs( I32 Board_ID, I32 PtId, F64 Vs );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().
 I32 PtId: the point table id is from 0 to 1.
 F64 Vs: The started speed of point table.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0
F64 Vs = 100.0;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
....
//Configure Vs to point table 0
ret = APS_pt_set_vs(Board_ID , PtId, Vs );
....
```

See also:

APS_pt_get_vs	Get configuration of Vs in the point table
---------------	--

Descriptions:

This function is used to get started speed (Vs) of point table.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_get_vs( I32 Board_ID, I32 PtId, F64 *Vs );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

F64 *Vs: The started speed of point table.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0
F64 Vs;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
....
//Get vs of point table 0
ret = APS_pt_get_vs(Board_ID , PtId, &Vs );
....
```

See also:

APS_pt_start	Start point table
--------------	-------------------

Descriptions:

This function is used to start point table.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_start( I32 Board_ID, I32 Ptbd );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Ptbd: the point table id is from 0 to 1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 Ptbd = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Configure Vs to point table 0, Push point to point table
//Start point table to move
ret = APS_pt_start(Board_ID , Ptbd );
```

See also:

APS_pt_stop	Stop point table
-------------	------------------

Descriptions:

This function is used to stop point table.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_stop( I32 Board_ID, I32 Ptbd );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Ptbd: the point table id is from 0 to 1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 Ptbd = 0; //Point table 0

//Stop point table to move
ret = APS_pt_stop(Board_ID , Ptbd );
```

See also:

APS_get_pt_status	Get status of point table
-------------------	---------------------------

Descriptions:

This function is used to get status of point table. There are information including the state of point table, the point buffer status, the usage and free space of point buffer, and the running counts. The detail is described as follows:

- The state includes START and STOP states of point table.
- The buffer status includes FULL or EMPTY status of point buffer.
- The usage space is a counter of consuming buffer size.
- The free space is a counter of remaining buffer size.
- The running count means how many points are executed after point table is enabled.

There are totally 50 point buffers in a point table. User could push a specified move, including line move, arc move, or helical move, into the buffer. Then, user could monitor status of point table for buffer status & running status. If buffer is not full, user could push more moves in to buffer. If buffer is already full, invoking Sleep() for a while to wait points to be consumed.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_get_pt_status( I32 Board_ID, I32 PtId, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTSTS Status: The status of point table

typedef struct

{

```
    U16 BitSts; //b0: Is PT work? [1:working, 0:Stopped]
                //b1: Is point buffer full? [1:full, 0:not full]
                //b2: Is point buffer empty? [1:empty, 0:not empty]
                //b3, b4, b5: Reserved for future, Don't care.

    U16 PntBuffFreeSpace; // Free space of point buffer
    U16 PntBufUsageSpace; //Usage space of point buffer
```

```
    U32 RunningCnt; //How many points be executed after point table is enabled
} PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0
PTSTS Status;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Get status of point table 0
ret = APS_get_pt_status(Board_ID , PtBld, &Status );
....
```

See also:

APS_reset_pt_buffer	Reset related buffer of point table
---------------------	-------------------------------------

Descriptions:

This function is used to reset some buffers of point table. There are three buffers in the point table, including a move buffer, a command buffer, and a profile buffer.

The move buffer could queue up to 50 moves.

The command buffer could control Do with a move.

The profile buffer could change speed profile with a move, including Acc, Dec, S-factor, Vm, Ve and so on.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_reset_pt_buffer( I32 Board_ID, I32 PtId );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtId = 0; //Point table 0  
  
//Enable point table 0 to 2d dimension with aixs 0 and axis 1.  
//Reset buffer of point table 0  
ret = APS_reset_pt_buffer(Board_ID , PtId );
```

See also:

APS_pt_roll_back	Rollback to previous point
------------------	----------------------------

Descriptions:

This function is used to rollback to previous point. After invoking APS_pt_stop() to pause point table, user could rollback point table back to previous point. Then, re-start point table to execute unfinished moves by invoking APS_pt_start().

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_roll_back( I32 Board_ID, I32 Ptbd, F64 Max_Speed );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Ptbd: the point table id is from 0 to 1.

F64 Max_Speed: Max speed by float.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 Ptbd = 0; //Point table 0
F64 Max_Speed = 10000.0;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Push points into point table.
//Start point table. Then, pause point table.
//Rollback to previous point
ret = APS_pt_roll_back( Board_ID, Ptbd, Max_Speed );
//Then, restart point table.
```

See also:

APS_get_pt_error	Get error code of point table
------------------	-------------------------------

Descriptions:

This function is used to get error code of point table. If point table has error in operation, error code will be recorded. Error code will be reset when re-enabling point table. Error code refers to “ErrorCodeDef.h” to get physical meanings.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_get_pt_error( I32 Board_ID, I32 PtBld, I32 *ErrCode );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtBld: the point table id is from 0 to 1.

I32 *ErrCode: Error code of running point table.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0
I32 ErrCode = 0;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Get error code of running point table
ret = APS_get_pt_error(Board_ID , PtBld, &ErrCode );
....
```

See also:

APS_pt_dwell	Push a dwell move into point buffer of point table.
--------------	---

Descriptions:

This function is used to push a dwell move into point buffer. There are up to 50 point buffer to pre-stored points in a point table. User could monitor usage / free space of point buffer to push moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_dwell( I32 Board_ID, I32 Ptbd, PPTDWL Prof, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Ptbd: the point table id is from 0 to 1.

PPTDWL Prof: The profile of dwell move

```
typedef struct
{
    F64          DwTime; //Set dwell time, unit is ms.
} PTDWL, *PPTDWL;
```

PPTSTS Status: The status of point table

```
typedef struct
{
    U16 BitSts; //b0: Is PT work? [1:working, 0:Stopped]
                //b1: Is point buffer full? [1:full, 0:not full]
                //b2: Is point buffer empty? [1:empty, 0:not empty]
                //b3, b4, b5: Reserved for future. Don't care.
                //b6~: Be always 0
    U16 PntBufFreeSpace; // Free space of point buffer
    U16 PntBufUsageSpace; //Usage space of point buffer
    U32 RunningCnt; //How many points be executed after point table is enabled
}
```

```
PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0
PTDWL Prof;
PTSTS Status;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Get status of point table 0
ret = APS_get_pt_status(Board_ID , PtBld, &Status );
if ( !( Status.BitSts & 0x02 ) ) //Point buffer is not full
{
    //Push move into point buffer
    Prof.DwTime = 100; //100ms
    ret = APS_pt_dwell( Board_ID, PtBld, &Prof, &Status );
}
//Start point table move
APS_pt_start( Board_ID, PtBld, 0 );
```

See also:

APS_pt_line	Push a line move into point buffer of point table.
-------------	--

Descriptions:

This function is used to push a line move into point buffer. There are up to 50 point buffer to pre-stored points in a point table. User could monitor usage / free space of point buffer to push moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_line( I32 Board_ID, I32 PtId, PPTLINE Prof, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTLINE Prof: The profile of line move

```
typedef struct
{
    I32      Dim; //dimension
    F64      Pos[6]; //position array for line move
} PTLINE, *PPTLINE;
```

PPTSTS Status: The status of point table

```
typedef struct
{
    U16 BitSts; //b0: Is PT work? [1:working, 0:Stopped]
                //b1: Is point buffer full? [1:full, 0:not full]
                //b2: Is point buffer empty? [1:empty, 0:not empty]
                //b3, b4, b5: Reserved for future. Don't care.
                //b6~: Be always 0
    U16 PntBufFreeSpace; // Free space of point buffer
    U16 PntBufUsageSpace; //Usage space of point buffer
    U32 RunningCnt; //How many points be executed after point table is enabled
} PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtBld = 0; //Point table 0  
PTLINE Prof;  
PTSTS Status;  
  
//Enable point table 0 to 2d dimension with axis 0 and axis 1.  
//Get status of point table 0  
ret = APS_get_pt_status(Board_ID, PtBld, &Status );  
if ( !( Status.BitSts & 0x02 ) ) //Point buffer is not full  
{  
    //Push move into point buffer  
    Prof.Dim = 2;  
    Prof.Pos[0] = 10000;  
    Prof.Pos[1] = 10000;  
    ret = APS_pt_line( Board_ID, PtBld, &Prof, &Status );  
}  
//Start point table move  
APS_pt_start( Board_ID, PtBld, 0 );
```

See also:

APS_pt_arc2_ca	Push a 2d arc move into point buffer of point table.
----------------	--

Descriptions:

This function is used to push a 2d arc move with angle into point buffer. There are up to 50 point buffer to pre-stored points in a point table. User could monitor usage / free space of point buffer to push moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_arc2_ca( I32 Board_ID, I32 PtId, PPTA2CA Prof, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTA2CA Prof: The profile of arc move

```
typedef struct
{
    U8      Index[2]; // [0 ~ dimension of point table] Which axis index in point table
    F64     Center[2]; // Center position
    F64     Angle; // Angle, unit is radian
} PTA2CA, *PPTA2CA;
```

PPTSTS Status: The status of point table

```
typedef struct
{
    U16 BitSts; // b0: Is PT work? [1:working, 0:Stopped]
                // b1: Is point buffer full? [1:full, 0:not full]
                // b2: Is point buffer empty? [1:empty, 0:not empty]
                // b3, b4, b5: Reserved for future. Don't care.
                // b6~: Be always 0
    U16 PntBufFreeSpace; // Free space of point buffer
    U16 PntBufUsageSpace; // Usage space of point buffer
    U32 RunningCnt; // How many points be executed after point table is enabled
} PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0
PTA2CA Prof;
PTSTS Status;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Get status of point table 0
ret = APS_get_pt_status(Board_ID , PtBld, &Status );
if ( !( Status.BitSts & 0x02 ) ) //Point buffer is not full
{
    //Push move into point buffer
    Prof.Index[0] = 0; //pick dimension 0
    Prof.Index[1] = 1; //pick dimension 1
    Prof.Center[0] = 10000;
    Prof.Center[1] = 10000;
    Prof.Angle = 3.14159265;
    ret = APS_pt_arc2_ca( Board_ID, PtBld, &Prof, &Status );
}
//Start point table move
APS_pt_start( Board_ID, PtBld );
```

See also:

APS_pt_arc2_ce	Push a 2d arc move into point buffer of point table.
----------------	--

Descriptions:

This function is used to push a 2d arc move with end position into point buffer. There are up to 50 point buffer to pre-stored points in a point table. User could monitor usage / free space of point buffer to push moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_arc2_ce( I32 Board_ID, I32 PtId, PPTA2CE Prof, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTA2CE Prof: The profile of arc move

```
typedef struct
{
    U8      Index[2]; // [0 ~ dimension of point table] Which axis index in point table
    F64     Center[2]; // Center position
    F64     End[2]; // End position
    I16     Dir; // A value specifies the rotate direction, If Dir set 0 means rotate in
                // positive direction, dir = -1 rotate in negative direction.
} PTA2CE, *PPTA2CE;
```

PPTSTS Status: The status of point table

```
typedef struct
{
    U16 BitSts; // b0: Is PT work? [1:working, 0:Stopped]
                // b1: Is point buffer full? [1:full, 0:not full]
                // b2: Is point buffer empty? [1:empty, 0:not empty]
                // b3, b4, b5: Reserved for future. Don't care.
                // b6~: Be always 0
    U16 PntBufFreeSpace; // Free space of point buffer
    U16 PntBufUsageSpace; // Usage space of point buffer
```

```
    U32 RunningCnt; //How many points be executed after point table is enabled
} PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0
PTA2CE Prof;
PTSTS Status;

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Get status of point table 0
ret = APS_get_pt_status(Board_ID, PtBld, &Status );
if ( !( Status.BitSts & 0x02 ) ) //Point buffer is not full
{
    //Push move into point buffer
    Prof.Index[0] = 0; //pick dimension 0
    Prof.Index[1] = 1; //pick dimension 1
    Prof.Center[0] = 10000;
    Prof.Center[1] = 10000;
    Prof.End[0] = 0;
    Prof.End[1] = 0;
    Prof.Dir = 0; //Positvie direction
    ret = APS_pt_arc2_ce( Board_ID, PtBld, &Prof, &Status );
}

//Start point table move
APS_pt_start( Board_ID, PtBld );
```

See also:

APS_pt_arc3_ca	Push a 3d arc move into point buffer of point table.
----------------	--

Descriptions:

This function is used to push a 3d arc move with angle into point buffer. There are up to 50 point buffer to pre-stored points in a point table. User could monitor usage / free space of point buffer to push moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_arc3_ca( I32 Board_ID, I32 PtId, PPTA3CA Prof, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTA2CA Prof: The profile of 3d arc move with angle

```
typedef struct
{
    U8      Index[3]; // [0 ~ dimension of point table] Which axis index in point table
    F64     Center[3]; // Center position
    F64     Noraml[3]; // Normal vector
    F64     Angle; // Angle, unit is radian
} PTA3CA, *PPTA3CA;
```

PPTSTS Status: The status of point table

```
typedef struct
{
    U16 BitSts; // b0: Is PT work? [1:working, 0:Stopped]
                // b1: Is point buffer full? [1:full, 0:not full]
                // b2: Is point buffer empty? [1:empty, 0:not empty]
                // b3, b4, b5: Reserved for future. Don't care.
                // b6~: Be always 0
    U16 PntBufFreeSpace; // Free space of point buffer
    U16 PntBufUsageSpace; // Usage space of point buffer
    U32 RunningCnt; // How many points be executed after point table is enabled
```

```
 } PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtBld = 0; //Point table 0  
PTA3CA Prof;  
PTSTS Status;  
  
//Enable point table 0 to 2d dimension with axis 0 and axis 1.  
//Get status of point table 0  
ret = APS_get_pt_status(Board_ID, PtBld, &Status );  
if ( !( Status.BitSts & 0x02 ) ) //Point buffer is not full  
{  
    //Push move into point buffer  
    Prof.Index[0] = 0; //pick dimension index 0  
    Prof.Index[1] = 1; //pick dimension index 1  
    Prof.Index[2] = 2; //pick dimension index 2  
    Prof.Center[0] = 10000;  
    Prof.Center[1] = 10000;  
    Prof.Center[2] = 10000;  
    Prof.Normal[0] = 0;  
    Prof.Normal[1] = 0;  
    Prof.Normal[2] = 1;  
    Prof.Angle = 3.14159265; //In radian  
    ret = APS_pt_arc3_ca( Board_ID, PtBld, &Prof, &Status );  
}  
  
//Start point table move  
APS_pt_start( Board_ID, PtBld );
```

See also:

APS_pt_arc3_ce	Push a 3d arc move into point buffer of point table.
----------------	--

Descriptions:

This function is used to push a 3d arc move with end position into point buffer. There are up to 50 point buffer to pre-stored points in a point table. User could monitor usage / free space of point buffer to push moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_arc3_ce( I32 Board_ID, I32 PtId, PPTA3CE Prof, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTA3CE Prof: The profile of 3d arc move with end position

```
typedef struct
{
    U8      Index[3]; // [0 ~ dimension of point table] Which axis index in point table
    F64     Center[3]; // Center position
    F64     End[3]; // End position
    I16     Dir; // A value specifies the rotate direction, If Dir set 0 means rotate in
                // positive direction, dir = -1 rotate in negative direction.
} PTA3CE, *PPTA3CE;
```

PPTSTS Status: The status of point table

```
typedef struct
{
    U16 BitSts; // b0: Is PT work? [1:working, 0:Stopped]
                // b1: Is point buffer full? [1:full, 0:not full]
                // b2: Is point buffer empty? [1:empty, 0:not empty]
                // b3, b4, b5: Reserved for future. Don't care.
                // b6~: Be always 0
    U16 PntBufFreeSpace; // Free space of point buffer
    U16 PntBufUsageSpace; // Usage space of point buffer
```

```
U32 RunningCnt; //How many points be executed after point table is enabled  
} PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to APS_pt_arc3_ca().

See also:

APS_pt_spiral_ca	Push a helical move into point buffer of point table.
------------------	---

Descriptions:

This function is used to push a helical move with angle into point buffer. There are up to 50 point buffer to pre-stored points in a point table. User could monitor usage / free space of point buffer to push moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_spiral_ca( I32 Board_ID, I32 PtId, PPTHCA Prof, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTHCA Prof: The profile of helical move with angle

```
typedef struct
{
    U8      Index[3]; // [0 ~ dimension of point table] Which axis index in point table
    F64     Center[3]; // Center position
    F64     Noraml[3]; // Normal vector
    F64     Angle; // Angle, unit is radian
    F64     DeltaH; // The height of helical move, User unit,
    F64     FinalR; // The distant from end position to normal vector, User unit
} PTHCA, *PPTHCA;
```

PPTSTS Status: The status of point table

```
typedef struct
{
    U16 BitSts; // b0: Is PT work? [1:working, 0:Stopped]
                // b1: Is point buffer full? [1:full, 0:not full]
                // b2: Is point buffer empty? [1:empty, 0:not empty]
                // b3, b4, b5: Reserved for future. Don't care.
                // b6~: Be always 0
    U16 PntBufFreeSpace; // Free space of point buffer
    U16 PntBufUsageSpace; // Usage space of point buffer
```

```
U32 RunningCnt; //How many points be executed after point table is enabled  
} PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to APS_pt_arc3_ca().

See also:

APS_pt_spiral_ce	Push a helical move into point buffer of point table.
------------------	---

Descriptions:

This function is used to push a helical move with end position into point buffer. There are up to 50 point buffer to pre-stored points in a point table. User could monitor usage / free space of point buffer to push moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_spiral_ce( I32 Board_ID, I32 PtId, PPTHCE Prof, PPTSTS Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

PPTHCE Prof: The profile of helical move with end position

```
typedef struct
{
    U8      Index[3]; // [0 ~ dimension of point table] Which axis index in point table
    F64     Center[3]; // Center position
    F64     Noraml[3]; // Normal vector
    F64     End[3]; // End position
    I16     Dir; // A value specifies the rotate direction, if Dir set 0 means rotate in
                // positive direction, dir = -1 rotate in negative direction.
} PTHCE, *PPTHCE;
```

PPTSTS Status: The status of point table

```
typedef struct
{
    U16 BitSts; // b0: Is PT work? [1:working, 0:Stopped]
                // b1: Is point buffer full? [1:full, 0:not full]
                // b2: Is point buffer empty? [1:empty, 0:not empty]
                // b3, b4, b5: Reserved for future. Don't care.
                // b6~: Be always 0
    U16 PntBufFreeSpace; // Free space of point buffer
```

```
U16 PntBufUsageSpace; //Usage space of point buffer
U32 RunningCnt; //How many points be executed after point table is enabled
} PTSTS, *PPTSTS;
```

Return Values:

I32 Error code: Please refer to error code table.

Example:

Refer to APS_pt_arc3_ca().

See also:

APS_pt_ext_set_do_ch	Set do extension command into command buffer. Command buffer is active when pushing a move into point table.
----------------------	--

Descriptions:

This function is used to set do extension command into command buffer. Command buffer is active when pushing a move into point table. After pushing a move, command buffer will be automatically cleared. User could write up to 7 commands into command buffer. Then, pushing a move into point table will take those commands into point table together. Now, do command is supported. Other commands are reserved for future.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_ext_set_do_ch( I32 Board_ID, I32 PtId, I32 Channel, I32 OnOff );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

I32 Channel: Do channel

I32 OnOff: Do on/off. 1: On, 0: Off.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set do extension command to command buffer //Set do channel 0 to turn on
ret = APS_pt_ext_set_do_ch( Board_ID, PtId, 0, 1 );
//Push a move into point buffer
```

See also:

APS_pt_ext_set_table_no	Set VAO table No. extension command into command buffer. Command buffer is active when pushing a move into point table.
-------------------------	---

Descriptions:

This function is used to set VAO table No. extension command into command buffer. Command buffer is active when pushing a move into point table. After pushing a move, command buffer will be automatically cleared. User could write up to 7 commands into command buffer. Then, pushing a move into point table will take those commands into point table together. Now, table No. command is supported. Other commands are reserved for future.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_ext_set_table_no( I32 Board_ID, I32 PtBld, I32 CtrlNo, I32 TableNo );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtBld: the point table id is from 0 to 1.

I32 CtrlNo: Control No. is from 0 to 1.

I32 TableNo: VAO Table No. is from -1 to 7.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set table No. extension command to command buffer //Set VAO table No. to 0, and set control No.
to 0
ret = APS_pt_ext_set_table_no( Board_ID, PtBld, 0, 0 );
//Push a move into point buffer
```

See also:

APS_pt_set_absolute	Set absolute profile into profile buffer.
---------------------	---

Descriptions:

This function is used to set absolute profile into profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_absolute ( I32 Board_ID, I32 Ptbd );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Ptbd: the point table id is from 0 to 1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 Ptbd = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set absolute profile to profilr buffer
ret = APS_pt_set_absolute ( Board_ID, Ptbd );
//Push a move into point buffer
```

See also:

APS_pt_set_relative	Set relative profile into profile buffer.
---------------------	---

Descriptions:

This function is used to set relative profile into profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_relative ( I32 Board_ID, I32 PtBld );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtBld: the point table id is from 0 to 1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set relative profile to profilr buffer
ret = APS_pt_set_relative ( Board_ID, PtBld );
//Push a move into point buffer
```

See also:

APS_pt_set_trans_buffered	Set transition to buffer mode in profile buffer
---------------------------	---

Descriptions:

This function is used to set transition to buffer mode in profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_trans_buffered( I32 Board_ID, I32 Ptbd );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Ptbd: the point table id is from 0 to 1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;  
I32 Board_ID = 0;  
I32 Ptbd = 0; //Point table 0  
  
//Enable point table 0 to 2d dimension with aixs 0 and axis 1.  
// Set to buffered mode  
ret = APS_pt_set_trans_buffered( Board_ID, Ptbd );  
//Push a move into point buffer
```

See also:

APS_pt_set_trans_inp	Set transition to in-position mode in profile buffer
----------------------	--

Descriptions:

This function is used to set transition to in-position mode in profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 FNTYPE APS_pt_set_trans_inp( I32 Board_ID, I32 Ptbd );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Ptbd: the point table id is from 0 to 1.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 Ptbd = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
// Set to in-position mode
ret = APS_pt_set_trans_inp( Board_ID, Ptbd );
//Push a move into point buffer
```

See also:

APS_pt_set_trans_blend_dec	Set transition to blending mode with deceleration in profile buffer
----------------------------	---

Descriptions:

This function is used to set transition to blending mode with deceleration in profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_trans_blend_dec( I32 Board_ID, I32 PtId, F64 Bp );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

F64 Bp: Deceleration rate. [Bp > 0, unit/s^2]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
// Set to blending mode with deceleration.
ret = APS_pt_set_trans_blend_dec( Board_ID, PtId, 10000 );
//Push a move into point buffer
```

See also:

APS_pt_set_trans_blend_dist	Set transition to blending mode with residue distant in profile buffer
-----------------------------	--

Descriptions:

This function is used to set transition to blending mode with residue distant in profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_trans_blend_dist( I32 Board_ID, I32 Ptbd, F64 Bp );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Ptbd: the point table id is from 0 to 1.

F64 Bp: Residue distance. Unit is user unit, generally is pulse. [Bp >= 0]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 Ptbd = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
// Set to blending mode with residue distant.
ret = APS_pt_set_trans_blend_dist( Board_ID, Ptbd, 100 );
//Push a move into point buffer
```

See also:

APS_pt_set_trans_blend_pcnt	Set transition to blending mode with residue distant percentage in profile buffer
-----------------------------	---

Descriptions:

This function is used to set transition to blending mode with residue distant percentage in profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_trans_blend_pcnt( I32 Board_ID, I32 PtBld, F64 Bp );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtBld: the point table id is from 0 to 1.

F64 Bp: Residue distance in travel distance's percentage. Unit is %. [Bp: 0.0 ~ 1.0]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
// Set to blending mode with residue distant percentage
ret = APS_pt_set_trans_blend_pcnt( Board_ID, PtBld, 0.05 );
//Push a move into point buffer
```

See also:

APS_pt_set_acc	Set acceleration profile into profile buffer.
----------------	---

Descriptions:

This function is used to set acceleration profile into profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_acc( I32 Board_ID, I32 PtId, F64 Acc );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

F64 Acc: Acceleration rate. [unit/s^2, > 0]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set acceleration to 10000
ret = APS_pt_set_acc( Board_ID, PtId, 10000 );
//Push a move into point buffer
```

See also:

APS_pt_set_dec	Set deceleration profile into profile buffer.
----------------	---

Descriptions:

This function is used to set deceleration profile into profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke `APS_pt_enable()` first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_dec( I32 Board_ID, I32 PtBld, F64 Dec );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to `APS_initial()`.

I32 PtBld: the point table id is from 0 to 1.

F64 Dec: Deceleration rate. [unit/s^2, > 0]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set deceleration to 10000
ret = APS_pt_set_dec( Board_ID, PtBld, 10000 );
//Push a move into point buffer
```

See also:

APS_pt_set_acc_dec	Set acceleration & deceleration profile into profile buffer.
--------------------	--

Descriptions:

This function is used to set acceleration and deceleration profile into profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_acc_dec( I32 Board_ID, I32 PtBld, F64 AccDec );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtBld: the point table id is from 0 to 1.

F64 AccDec: Acceleration/deceleration rate. [unit/s^2, > 0]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set acceleration / deceleration to 10000
ret = APS_pt_set_acc_dec( Board_ID, PtBld, 10000 );
//Push a move into point buffer
```

See also:

APS_pt_set_s	Set S-factor profile into profile buffer.
--------------	---

Descriptions:

This function is used to set S-factor profile into profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_s( I32 Board_ID, I32 PtBld, F64 Sf );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtBld: the point table id is from 0 to 1.

F64 Sf: s-factor [0 ~ 1]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;  
I32 Board_ID = 0;  
I32 PtBld = 0; //Point table 0  
  
//Enable point table 0 to 2d dimension with aixs 0 and axis 1.  
//Set s-factor to 0.5  
ret = APS_pt_set_s( Board_ID, PtBld, 0.5 );  
//Push a move into point buffer
```

See also:

APS_pt_set_vm	Set maximum velocity profile into profile buffer.
---------------	---

Descriptions:

This function is used to set maximum velocity profile into profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_vm( I32 Board_ID, I32 PtId, F64 Vm );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtId: the point table id is from 0 to 1.

F64 Vm : Max. velocity [Vm >= 0]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtId = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set Vm to 10000
ret = APS_pt_set_vm( Board_ID, PtId, 10000 );
//Push a move into point buffer
```

See also:

APS_pt_set_ve	Set end velocity profile into profile buffer.
---------------	---

Descriptions:

This function is used to set end velocity profile into profile buffer. Profile buffer is active when pushing a move into point table. User usually sets profile all together in the beginning after enabling point table. If user wants to change some profile, user could modify them before pushing a move into point buffer. On the contrary, if user doesn't want to modify profile, the same profile will be automatically maintained for following moves.

Note: It is necessary to invoke APS_pt_enable() first to enable point table. Otherwise, it will be return error code.

Note: Don't invoke Pt functions with Feeder fuctions at the same time. They are exclusive.

Syntax:

C/C++:

```
I32 APS_pt_set_ve( I32 Board_ID, I32 PtBld, F64 Ve );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PtBld: the point table id is from 0 to 1.

F64 Ve: end velocity [Ve >= 0]

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret;
I32 Board_ID = 0;
I32 PtBld = 0; //Point table 0

//Enable point table 0 to 2d dimension with aixs 0 and axis 1.
//Set Ve to 100
ret = APS_pt_set_ve( Board_ID, PtBld, 100 );
//Push a move into point buffer
```

See also:

15. Compare trigger

APS_set_trigger_param	Set compare trigger related parameter
-----------------------	---------------------------------------

Descriptions:

This function is used to set comparing trigger related parameters. All definitions of trigger parameters are described in trigger parameter table.

You can also get parameter setting using “APS_get_trigger_param()” function.

Syntax:

C/C++:

```
I32 APS_set_trigger_param( I32 Board_ID, I32 Param_No, I32 Param_Val );
```

Visual Basic:

```
APS_set_trigger_param( ByVal Board_ID As Long, ByVal Param_No As Long, ByVal Param_Val As Long)  
As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Param_No: Parameter number. Refer to trigger parameter table.

I32 Param_Val: Parameter value. Refer to trigger parameter table.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 BoardId = 0;  
APS_set_trigger_param(BoardId, 0x0, 0 ); //Set linear compare source
```

See also:

APS_get_trigger_param();

APS_get_trigger_param	Get compare trigger related parameter
-----------------------	---------------------------------------

Descriptions:

This function is used to get comparing trigger related parameters. All definitions of trigger parameters are described in trigger parameter table.

You can also set parameter using “APS_set_trigger_param()” function.

Syntax:

C/C++:

```
I32 APS_get_trigger_param( I32 Board_ID, I32 Param_No, I32 *Param_Val );
```

Visual Basic:

```
APS_get_trigger_param( ByVal Board_ID As Long, ByVal Param_No As Long, Param_Val As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Param_No: Parameter number. Refer to trigger parameter table.

I32 Param_Val: Return parameter value. Refer to trigger parameter table.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 BoardId = 0;
```

```
I32 Param_Val = 0;
```

```
APS_get_trigger_param(BoardId, 0x0, &Param_Val ); //Get linear compare source
```

See also:

[APS_set_trigger_param\(\)](#)

APS_set_trigger_linear	Set linear comparing function
------------------------	-------------------------------

Descriptions:

This function is used to set linear comparing function.

When the linear trigger operation is completed, the total compared point will be:

Total compared point number = RepeatTimes. (StartPoint as first trigger point)

Syntax:

C/C++:

```
I32 APS_set_trigger_linear( I32 Board_ID, I32 LCmpCh, I32 StartPoint, I32 RepeatTimes, I32 Interval );
```

Visual Basic:

```
APS_set_trigger_linear(ByVal Board_ID As Long, ByVal LCmpCh As Long, ByVal StartPoint As Long,
ByVal RepeatTimes As Long, ByVal Interval As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 LCmpCh: Linear compare set channel. Zero base. Range is from 0 to 3.

I32 StartPoint: Start linear trigger point.

I32 RepeatTimes: Trigger repeat times.

I32 Interval: Trigger interval. (-16777215 ~ 16777215, unit is pulse)

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 BoardId = 0;
APS_set_trigger_param(BoardId, 0x0, 0 ); //Set linear compare source
APS_set_trigger_param(BoardId, 0x10, 0 ); //Set LCMP0 as TRG0's source
APS_set_trigger_linear(BoardId, 0, 100, 49999, 10 ); //Set LCMP0 linear compare algorithm.
// Start point = 100, RepeatTimes = 49999, Interval = 10.
APS_set_trigger_param(BoardId, 0x04, 1 ); //Enable LCMP0
// Trigger operation.
```

APS_set_trigger_param(0, 0x04, 0); //Disable LCMP0

See also:

APS_set_trigger_table;

APS_set_trigger_table	Set table comparing function
-----------------------	------------------------------

Descriptions:

This function is used to configure the specified comparing table.

Syntax:

C/C++:

```
I32 APS_set_trigger_table( I32 Board_ID, I32 TCmpCh, I32 *DataArr, I32 ArraySize );
```

Visual Basic:

```
APS_set_trigger_table( ByVal Board_ID As Long, ByVal TCmpCh As Long, DataArr As Long, ByVal
ArraySize As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TCmpCh: Specified comparing table number. Zero base. Range is from 0 to 3.

I32 *DataArr: Comparing data array.

I32 ArraySize The size of comparing data array. Please refer to product's specification.

Return Values:

I32 error code. Refer to error code table.

Example:

```
#define POINTS 1000
I32 ret;
I32 data[POINTS];
I32 i;
for( i = 0; i < POINTS; i++ )
    data[i] = 10 + i * 10;
```

```
APS_set_trigger_param(BoardId, 0x2, 0 ); //Set encoder counter 0 as TCMP0's source.
```

```
APS_set_trigger_param(BoardId, 0x10, 4 ); //Set TCMP0 as TRG0's source
```

```
ret = APS_set_trigger_table( 0, 0, data, POINTS );
```

```
APS_set_trigger_param(BoardId, 0x06, 1 ); //Enable TCMP0
```

```
// Trigger operation...
```

```
//When finish the trigger operation.
```

```
APS_set_trigger_param(BoardId, 0x06, 0 ); //Enable TCMP0
```

See also:

`APS_set_trigger_linear;`

APS_set_trigger_manual	Manual output trigger
------------------------	-----------------------

Descriptions:

This function is used to forced output a trigger at specified trigger output channel.

Syntax:

C/C++:

```
I32 APS_set_trigger_manual( I32 Board_ID, I32 TrgCh );
```

Visual Basic:

```
APS_set_trigger_manual( ByVal Board_ID As Long, ByVal TrgCh As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TrgCh: Trigger output channel (TRG) number. Zero based. Range is from 0 to 3.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 Board_ID = 0;  
I32 ret;  
ret = APS_set_trigger_manual( Board_ID, 1); //TRG1
```

See also:

[APS_set_trigger_manual_s](#)

APS_set_trigger_manual_s	Manual output trigger synchronously
--------------------------	-------------------------------------

Descriptions:

This function is used to forced to output a trigger pulse. It is designed to output one or more channels of trigger synchronously and manually.

Syntax:

C/C++:

```
I32 APS_set_trigger_manual_s( I32 Board_ID, I32 TrgChInBit );
```

Visual Basic:

```
APS_set_trigger_manual_s( ByValBoard_ID As Long, ByValTrgChInBit As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;  
ret = APS_set_trigger_manual_s( 0, 0xF ); //4 channels output trigger simultaneously.  
ret = APS_set_trigger_manual_s( 0, 0x2 ); //TRG1 outputs trigger.  
ret = APS_set_trigger_manual_s( 0, 0x3 ); //TRG0 and TRG1 output trigger simultaneously.  
//...
```

See also:

[APS_set_trigger_manual](#)

APS_get_trigger_table_cmp	Get current table comparing value
---------------------------	-----------------------------------

Descriptions:

This function is used to get current comparing value in the specified table comparator.

Syntax:

C/C++:

```
I32 APS_get_trigger_table_cmp( I32 Board_ID, I32 TCmpCh, I32 *CmpVal );
```

Visual Basic:

```
APS_get_trigger_table_cmp(ByVal Board_ID As Long, ByVal TCmpCh As Long, CmpVal As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TCmpCh: Specified the table comparator channel number. Zero base. Range is from 0 to 3.

I32 *CmpVal: Return the current comparing value in the comparator.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
I32 CmpVal;
ret = APS_get_trigger_table_cmp ( 0, 0, &CmpVal );
If( ret != ERR_NoError )
{ // Error, show message.
}
```

See also:

[APS_get_trigger_linear_cmp](#);

APS_get_trigger_linear_cmp	Get current linear comparing value
----------------------------	------------------------------------

Descriptions:

This function is used to get current comparing value in the specified linear comparator.

Syntax:

C/C++:

```
I32 APS_get_trigger_linear_cmp( I32 Board_ID, I32 LCmpCh, I32 *CmpVal );
```

Visual Basic:

```
APS_get_trigger_linear_cmp(ByVal Board_ID As Long, ByVal LCmpCh As Long, CmpVal As Long ) As  
Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 LCmpCh: Specified the linear comparator channel number. Zero base. Range is from 0 to 3.

I32 *CmpVal: Return the current comparing value in the comparator.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;  
I32 CmpVal;  
ret = APS_get_trigger_linear_cmp( 0, 0, &CmpVal );  
If( ret != ERR_NoError )  
{ // Error, show message.  
}
```

See also:

[APS_get_trigger_table_cmp](#);

APS_get_trigger_count	Get triggered count
-----------------------	---------------------

Descriptions:

This function is used to get the triggered counter value. This value means total triggered pulses from last counter reset. It is useful to check compared times.

Syntax:

C/C++:

```
I32 APS_get_trigger_count( I32 Board_ID, I32 TrgCh, I32 *TrgCnt );
```

Visual Basic:

```
APS_get_trigger_count(ByVal Board_ID As Long, ByVal TrgCh As Long, TrgCnt As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TrgCh: Specified trigger output counter channel number. Zero base. Range is from 0 to 3.

I32 *TrgCnt: Return trigger counter value.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 Ret;
I32 TrgCnt;
Ret = APS_get_trigger_count( 0, 0, &TrgCnt );
If( ret != ERR_NoError )
{ // Error, show message.
}
```

See also:

[APS_reset_trigger_count\(\)](#)

APS_reset_trigger_count	Reset triggered count.
-------------------------	------------------------

Descriptions:

This function is used to reset the triggered counter to zero.

Syntax:

C/C++:

```
I32 APS_reset_trigger_count( I32 Board_ID, I32 TrgCh );
```

Visual Basic:

```
APS_reset_trigger_count( ByVal Board_ID As Long, ByVal TrgCh As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TrgCh: Trigger counter channel number. Zero based. Range is from 0 to 3.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
```

```
ret = APS_reset_trigger_count( 0, 0 );
ret = APS_reset_trigger_count( 0, 1 );
ret = APS_reset_trigger_count( 0, 2 );
ret = APS_reset_trigger_count( 0, 3 );
...
```

See also:

[APS_get_trigger_count](#);

APS_get_timer_counter	Get timer count
-----------------------	-----------------

Descriptions:

This function is used to get the timer counter value.

Syntax:

C/C++:

```
I32 APS_get_timer_counter ( I32 Board_ID, I32 TmrCh, I32 *TmrCnt );
```

Visual Basic:

```
APS_get_timer_counter (ByVal Board_ID As Long, ByVal TmrCh As Long, TmrCnt As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TmrCh: Specified timer channel number. Zero base.

Only channel 0 is available.

I32 *TmrCnt: Return timer counter value.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 Ret;
I32 TmrCnt;
Ret = APS_get_timer_counter( 0, 0, &TmrCnt ); //Get counter from timer channel 0
If( ret != ERR_NoError )
{ // Error, show message.
}
```

See also:

[APS_set_timer_counter\(\)](#)

APS_set_timer_counter	Set timer count
-----------------------	-----------------

Descriptions:

This function is used to set timer counter.

Syntax:

C/C++:

```
I32 APS_set_timer_counter ( I32 Board_ID, I32 TmrCh, I32 TmrCnt );
```

Visual Basic:

```
APS_set_timer_counter ( ByVal Board_ID As Long, ByVal TmrCh As Long , ByVal TmrCnt As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TmrCh: Timer counter channel number. Zero based.

Only one channel is available in PCI-8258.

I32 TmrCnt: Specify timer counter value.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
```

```
//set timer counter channel 0 to 100
ret = APS_set_timer_counter( 0, 0, 100 );
```

...

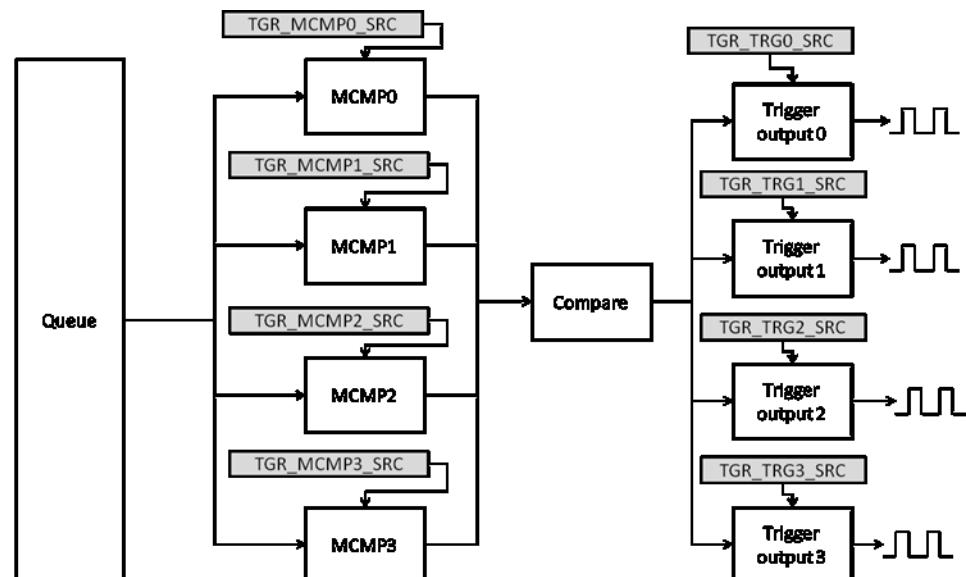
See also:

[APS_get_timer_counter](#);

APS_set_multi_trigger_table	Set table for comparing
-----------------------------	-------------------------

Descriptions:

This function is used to push data in table (FIFO) for comparing. There are four comparators designed for multi-dimension comparing application. The comparing points are pushed in the queue initially. User can use trigger parameter to select comparator source from encoder 0~7. Specify arbitrary trigger channel to generate PWM is allowed. Once the point is compared, the specified trigger channel will generate one PWM signal and its corresponding counter will add 1 simultaneously.



Comparator Configuration:

Dimension	Configuration
2	Select source in trigger parameter TGR_MCMPO_SRC / TGR_MCMPI_SRC
3	Select source in trigger parameter TGR_MCMPO_SRC / TGR_MCMPI_SRC / TGR_MCMPI2_SRC
4	Select source in trigger parameter TGR_MCMPO_SRC / TGR_MCMPI_SRC / TGR_MCMPI2_SRC / TGR_MCMPI3_SRC

Trigger Output Configuration

Channel	Configuration
0	Set bit 6 in trigger parameter TGR_TRG0_SRC
1	Set bit 6 in trigger parameter TGR_TRG1_SRC

2	Set bit 6 in trigger parameter TGR_TRG2_SRC
3	Set bit 6 in trigger parameter TGR_TRG3_SRC

Syntax:

C/C++:

```
I32 APS_set_multi_trigger_table( I32 Board_ID, I32 Dimension, MCMP_POINT *Point, I32 PointSize,
I32 Window );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Dimension: 2~4 dimension

MCMP_POINT *Point: Point array used for comparator. See description below for details.

```
// Multi-dimension comparator
typedef struct
{
    F64 axisX; // x axis data for multi-dimension comparator 0
    F64 axisY; // y axis data for multi-dimension comparator 1
    F64 axisZ; // z axis data for multi-dimension comparator 2
    F64 axisU; // u axis data for multi-dimension comparator 3
    U32 chInBit; // pwm output channel in bit format
}MCMP_POINT;
```

I32 PointSize: The size of point array.

I32 Window: Specify comparing range

Return Values:

I32 error code. Refer to error code table.

Example:

```
void main()
{
    I32 ret = 0;
    U32 i = 0;
    I32 BoardID_InBits;
    I32 BoardID = 0;
    I32 Mode = 0; //By system assigned
    I32 msts; // Motion status
    MCMP_POINT DataArr[10000];
    I32 data = 0;
    U32 totalPoint = 5000;
    U32 window = 10;
    U32 dimension = 2;
    I32 Axis_ID_Array[2] = {0, 1};
    I32 Distance_Array[2] = {1100, 2200 };
    I32 Max_Linear_Speed = 20000;
```

```

MCMP_POINT Point;
printf("\n");
// ****
// Initialization
// ****
ret = APS_initial( &BoardID_InBits, Mode);
if(ret)
{
    printf("APS initial fail\n");
    goto TEST_END;
}
printf("APS version = %d\n", (I32)APS_version() );
// ****
// Set trigger parameter
// ****
// Set comparator source: encode 0 for comparator 0 and encoder 1 for comparator 1
ret = APS_set_trigger_param( BoardID, TGR_MCMPO_SRC, 0 );
if(ret)
{
    printf("APS_set_trigger_param1 fail\n");
    goto TEST_END;
}
ret = APS_set_trigger_param( BoardID, TGR_MCMPI_SRC, 1 );
if(ret)
{
    printf("APS_set_trigger_param fail\n");
    goto TEST_END;
}

// Set PWM output channel 0
ret = APS_set_trigger_param( BoardID, TGR_TRG0_SRC, 0x40 );
if(ret)
{
    printf("APS_set_trigger_param fail\n");
    goto TEST_END;
}

// Set PWM output channel 1
ret = APS_set_trigger_param( BoardID, TGR_TRG1_SRC, 0x40 );
if(ret)
{
    printf("APS_set_trigger_param fail\n");
    goto TEST_END;
}

ret = APS_set_trigger_param( BoardID, TGR_TRG_EN, 0xF );
ret = APS_set_trigger_param( BoardID, TGR_TRG2_SRC, 0x40 );
ret = APS_set_trigger_param( BoardID, TGR_TRG3_SRC, 0x40 );

// Enable all trigger output channel
ret = APS_set_trigger_param( BoardID, TGR_TRG_EN, 0xF );

// ****
// Reset and read trigger count
// ****
// Reset PWM channel 0 trigger count
ret = APS_reset_trigger_count( BoardID, 0 );

```

```

ret = APS_reset_trigger_count( BoardID, 1 );
ret = APS_reset_trigger_count( BoardID, 2 );
ret = APS_reset_trigger_count( BoardID, 3 );
if(ret)
{
    printf("APS_reset_trigger_count fail\n");
    goto TEST_END;
}

// ****
// Set servo on
// ****

// Set axes servo ON
ret = APS_set_servo_on( 0, 1 );
if(ret)
{
    printf("Servo on fail\n");
    goto TEST_END;
}
ret = APS_set_servo_on( 1, 1 );
if(ret)
{
    printf("Servo on fail\n");
    goto TEST_END;
}

// Reset command
ret = APS_set_command( 0, 0 );
if(ret)
{
    printf("APS_set_command fail\n");
    goto TEST_END;
}
ret = APS_set_command( 1, 0 );
if(ret)
{
    printf("APS_set_command fail\n");
    goto TEST_END;
}

// ****
// Set compare points
// ****

// Prepare compare points
for(i=0; i<totalPoint; i++)
{
    DataArr[i].axisX = i * 10 + 10;
    DataArr[i].axisY = i * 20 + 20;
    DataArr[i].axisZ = 0;
    DataArr[i].axisU = 0;
    DataArr[i].chInBit = 0xF;
}

// Set compare points into queue
ret = APS_set_multi_trigger_table( BoardID, dimension, DataArr, totalPoint, window );

```

```

if(ret)
{
    printf("APS_set_multi_trigger_table fail\n");
    goto TEST_END;
}

// Check comparator data
ret = APS_get_multi_trigger_table_cmp( BoardID, dimension, &Point );
if(ret)
{
    printf("APS_get_trigger_table_cmp fail\n");
    goto TEST_END;
}
printf("Point in comparator: axisX = %f  axisY = %f\n", Point.axisX, Point.axisY );

// ****
// Start motor and read status
// ****

// Start interpolation
ret = APS_relative_linear_move( dimension, Axis_ID_Array, Distance_Array, Max_Linear_Speed );
if(ret)
{
    printf("APS_relative_linear_move fail\n");
    goto TEST_END;
}

// Check CSTP
while(1)
{
    F64 data1,data2;
    I32 data3, data4, data6, data7;
    U32 data5 = 0;
    U32 data8;
    msts = APS_motion_status(0);

    //ret = APSI_8258_read_fpga( 0, 1, 0x37c, &data5 );
    APS_get_trigger_count( BoardID, 0, &data3 );
    APS_get_trigger_count( BoardID, 1, &data4 );
    APS_get_trigger_count( BoardID, 2, &data6 );
    APS_get_trigger_count( BoardID, 3, &data7 );
    APS_get_position_f( 0, &data1 );
    APS_get_position_f( 1, &data2 );
    printf("fbk0 = %f fbk1 = %f cnt0 = %d cnt1 = %d cnt2 = %d cnt3 = %d ch = 0x%x\n", data1, data2,
data3, data4,data6,data7, data5);
    if( msts & 0x1)
        break;
    Sleep(10);
}

// ****
// Read final PWM count
// ****

// Check PWM channel 0 trigger counter
ret = APS_get_trigger_count( BoardID, 0, &data );
if(ret)
{

```

```

        printf("APS_get_trigger_count fail\n");
        goto TEST_END;
    }
    printf("Final pwm count 0 = %d\n", data );

    // Check PWM channel 1 trigger counter
    ret = APS_get_trigger_count( BoardID, 1, &data );
    if(ret)
    {
        printf("APS_get_trigger_count fail\n");
        goto TEST_END;
    }
    printf("Final pwm count 1 = %d\n", data );

    // Check PWM channel 2 trigger counter
    ret = APS_get_trigger_count( BoardID, 2, &data );
    if(ret)
    {
        printf("APS_get_trigger_count fail\n");
        goto TEST_END;
    }
    printf("Final pwm count 2 = %d\n", data );

    // Check PWM channel 3 trigger counter
    ret = APS_get_trigger_count( BoardID, 3, &data );
    if(ret)
    {
        printf("APS_get_trigger_count fail\n");
        goto TEST_END;
    }
    printf("Final pwm count 3 = %d\n", data );

TEST_END:

    // Set axes servo off
    ret = APS_set_servo_on( 0, 0 );
    ret = APS_set_servo_on( 1, 0 );
    ret = APS_close();
    system("PAUSE");
}

```

APS_get_multi_trigger_table_cmp	Get current table comparing value
---------------------------------	-----------------------------------

Descriptions:

This function is used to get current comparing value in the specified table comparator.

Syntax:

C/C++:

```
I32 APS_get_trigger_table_cmp( I32 Board_ID, I32 Dimension, MCMP_POINT *Point );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Dimension: 2~4 dimension

MCMP_POINT *Point: Return the current comparing value in comparator. See type_define.h for details.

Return Values:

I32 error code. Refer to error code table.

Example:

APS_set_trigger_table_data	Set table compator data (Fast table compare trigger function)
----------------------------	---

Descriptions:

This function is belong to fast table compare trigger function and it is used to set comparing data to comparing table. The size of comparing data is constrained by the FIFO free size, and its maximum value is 40.

Syntax:

C/C++:

```
I32 APS_set_trigger_table_data( I32 Board_ID, I32 TCmpCh, I32 *DataArr, I32 ArraySize );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TCmpCh: Specified comparing table number. Zero base. Range is from 0 to 3.

I32 *DataArr: Comparing data array.

I32 ArraySize The size of comparing data array. Its range is 1~40.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 FreeSize = 0;
I32 FifoSts = 0
I32 Data = 0;
I32 DataArr[1000];
U32 usedPoint=0;
U32 residualPoint=0;
U32 totalPoint = 500;

// Enable table compare trigger
ret = APS_enable_trigger_table( 0, 0, 1 );

// Reset FIFO compare data
ret = APS_reset_trigger_table( 0, 0 );

// Generate compare data
for(i=0; i<totalPoint; i++)
    DataArr[i] = i * 100 + 100;

while(1)
{
    // Get residual data size
    residualPoint = totalPoint - usedPoint;

    // Get FIFO status
    ret = APS_get_trigger_table_status( 0, 0, &FreeSize, &FifoSts );

    // Get current FIFO compare data
    APS_get_trigger_cmp_value( 0, 0, &Data );
```

```

// // Set compare data to FIFO
if(FreeSize >= 40)
{
    if(residualPoint >= 40)
    {
        ret = APS_set_trigger_table_data( 0, 0, &DataArr[usedPoint], 40 );
        if(ret ==0)
            usedPoint += 40;
    }
    else
    {
        ret = APS_set_trigger_table_data( 0, 0, &DataArr[usedPoint], residualPoint );
        if(ret ==0)
            usedPoint += residualPoint;
    }
}
else
{
    if(FreeSize >= residualPoint)
    {
        ret = APS_set_trigger_table_data( 0, 0, &DataArr[usedPoint], residualPoint );
        if(ret ==0)
            usedPoint += residualPoint;
    }
    else
    {
        ret = APS_set_trigger_table_data( 0, 0, &DataArr[usedPoint], FreeSize );
        if(ret ==0)
            usedPoint += FreeSize;
    }
}
}

// Complete set compare data to FIFO
if(usedPoint == totalPoint)
    break;

Sleep(1);
}

```

See also:

[APS_set_trigger_table_data](#)
[APS_get_trigger_table_status](#)
[APS_get_trigger_cmp_value](#)
[APS_enable_trigger_table](#)
[APS_reset_trigger_table](#)

APS_get_trigger_table_status	Get table comparator status (Fast table compare trigger function)
------------------------------	---

Descriptions:

This function is belong to fast table compare trigger function and it is used to get the FIFO status of table comparator.

Syntax:

C/C++:

```
I32 APS_get_trigger_table_status( I32 Board_ID, I32 TCmpCh, I32 *FreeSpace, I32 *FifoSts );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TCmpCh: Specified comparing table number. Zero base. Range is from 0 to 3.

I32 *FreeSpace: The free size of FIFO. The total free size is 1254.

I32 *FifoSts: The FIFO status: bit 0 = 1 indicates FIFO is full and bit 1 = 1 indicates FIFO is empty.

Return Values:

I32 error code. Refer to error code table.

Example:

See also:

[APS_set_trigger_table_data](#)

[APS_get_trigger_table_status](#)

[APS_get_trigger_cmp_value](#)

[APS_enable_trigger_table](#)

[APS_reset_trigger_table](#)

APS_get_trigger_cmp_value	Get table comparator value (Fast table compare trigger function)
---------------------------	--

Descriptions:

This function is belong to fast table compare trigger function and it is used to get the current comparing value of table comparator.

Syntax:

C/C++:

```
I32 FNTYPE APS_get_trigger_cmp_value( I32 Board_ID, I32 TCmpCh, I32 *CmpVal );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TCmpCh: Specified comparing table number. Zero base. Range is from 0 to 3.

I32 *CmpVal: The current coparing value of table comparator.

Return Values:

I32 error code. Refer to error code table.

Example:

See also:

[APS_set_trigger_table_data](#)

[APS_get_trigger_table_status](#)

[APS_get_trigger_cmp_value](#)

[APS_enable_trigger_table](#)

[APS_reset_trigger_table](#)

APS_enable_trigger_table	Enable table comparator (Fast table compare trigger function)
--------------------------	---

Descriptions:

This function is belong to fast table compare trigger function and it is used to enable the table comparator.

Syntax:

C/C++:

```
I32 APS_enable_trigger_table( I32 Board_ID, I32 TCmpCh, I32 Enable );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TCmpCh: Specified comparing table number. Zero base. Range is from 0 to 3.

I32 Enable: Set Enable = 1 to begin table comparator.

Return Values:

I32 error code. Refer to error code table.

Example:

See also:

[APS_set_trigger_table_data](#)
[APS_get_trigger_table_status](#)
[APS_get_trigger_cmp_value](#)
[APS_enable_trigger_table](#)
[APS_reset_trigger_table](#)

APS_reset_trigger_table	Reset table comparator (Fast table compare trigger function)
-------------------------	--

Descriptions:

This function is belong to fast table compare trigger function and it used to reset the FIFO of table comparator.

Syntax:

C/C++:

```
I32 FNTYPE APS_reset_trigger_table( I32 Board_ID, I32 TCmpCh );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TCmpCh: Specified comparing table number. Zero base. Range is from 0 to 3.

Return Values:

I32 error code. Refer to error code table.

Example:

See also:

[APS_set_trigger_table_data](#)
[APS_get_trigger_table_status](#)
[APS_get_trigger_cmp_value](#)
[APS_enable_trigger_table](#)
[APS_reset_trigger_table](#)

16. Program download

APS_load_vmc_program	Load VMC file to task memory
----------------------	------------------------------

Descriptions:

This function is used to get VMC file to task memory.

Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_load_vmc_program ( I32 Board_ID, I32 TaskNum, const char *pFile, I32 Password);
```

Visual Basic:

```
APS_load_vmc_program (ByVal Board_ID As Long, ByVal TaskNum As Long, pFile As String, ByVal  
Password As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TaskNum: Specify a task number from 0 to 7.

I32 *pFile: Specified a VMC file which created by MCPro2.exe.

I32 Password: Input a specified password for security.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;  
I32 boardId = 0;  
I32 taskNum = 0;  
  
//Load a VMC file named "BubbleSort.txt" to specified task.  
ret = APS_load_vmc_program( boardId, taskNum, "BubbleSort.txt", 0 );
```

See also:

[APS_save_vmc_program\(\)](#)

APS_save_vmc_program	Save to VMC file from task memory
----------------------	-----------------------------------

Descriptions:

This function is used to save task program from task memory to VMC file.

Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_save_vmc_program( I32 Board_ID, I32 TaskNum, const char *pFile, I32 Password);
```

Visual Basic:

```
APS_save_vmc_program (ByVal Board_ID As Long, ByVal TaskNum As Long, pFile As String, ByVal  
Password As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TaskNum: Specify a task number from 0 to 7.

I32 *pFile: Specify a VMC file to save.

I32 Password: Input a specified password for security.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;  
I32 boardId = 0;  
I32 taskNum = 0;
```

```
//Save task program to a VMC file named "BubbleSort.txt".  
ret = APS_save_vmc_program( boardId, taskNum, "BubbleSort.txt", 0 );
```

See also:

[APS_load_vmc_program\(\)](#)

APS_set_task_mode	Set task run mode
-------------------	-------------------

Descriptions:

This function is used to set task run mode.

Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_set_task_mode( I32 Board_ID, I32 TaskNum, U8 Mode, U16 LastIP );
```

Visual Basic:

```
APS_set_task_mode (ByVal Board_ID As Long, ByVal TaskNum As Long, ByVal Mode As Byte, ByVal
LastIP As Integer) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TaskNum: Specify a task number from 0 to 7.

U8 Mode: Two run mode to set.

0: Normal mode. 1: Repeat mode.

U16 LastIP: Last instruction offset. It is only available in repeat mode.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
I32 boardId = 0;
I32 taskNum = 0;
U16 lastIP = 0;

//Set task 0 to normal mode. IP is ignored in normal mode.
ret = APS_set_task_mode ( boardId, taskNum, 0, &lastIP );
```

See also:

[APS_get_task_mode \(\)](#)

APS_get_task_mode	Get task run mode
-------------------	-------------------

Descriptions:

This function is used to get task run mode.

Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_get_task_mode( I32 Board_ID, I32 TaskNum, U8 *Mode, U16 *LastIP );
```

Visual Basic:

```
APS_get_task_mode (ByVal Board_ID As Long, ByVal TaskNum As Long, Mode As Byte, ByVal LastIP As Integer) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TaskNum: Specify a task number from 0 to 7.

U8 *Mode: Two run mode.

0: Normal mode. 1: Repeat mode.

U16 *LastIP: Last instruction offset. It is only available in repeat mode.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
I32 boardId = 0;
I32 taskNum = 0;
U8 mode = 0;
U16 lastIP = 0;

//Get run mode from task 0. IP is ignored in normal mode.
ret = APS_get_task_mode ( boardId, taskNum, &mode, &lastIP);
```

See also:

[APS_set_task_mode \(\)](#)

APS_start_task	Start task control command
----------------	----------------------------

Descriptions:

This function is used to start task control command.

Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_start_task( I32 Board_ID, I32 TaskNum, I32 CtrlCmd );
```

Visual Basic:

```
APS_start_task (ByVal Board_ID As Long, ByVal TaskNum As Long, ByVal CtrlCmd As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TaskNum: Specify a task number from 0 to 7.

I32 CtrlCmd: Control command.

- 0: TSK_RESET. Reset task. (Not start Program)
- 1: TSK_RESTART. Restart task. (Start program at the same time)
- 2: TSK_STOP. Stop Program.
- 3: TSK_RUN. Start program.
- 4: TSK_STEP. Step(Run) one instruction. Then stop.
- 5: TSK_STEP_P. Run until parallel bit == 0

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
I32 boardId = 0;
I32 taskNum = 0;
I32 CtrlCmd = 3;

//Run the program of task 0
ret = APS_start_task ( boardId, taskNum, CtrlCmd );
```

See also:

APS_get_task_info(); APS_get_task_msg()

APS_get_task_info	Get task information
-------------------	----------------------

Descriptions:

This function is used to get task information.

Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_get_task_info( I32 Board_ID, I32 TaskNum, TSK_INFO *Info );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TaskNum: Specify a task number from 0 to 7.

TSK_INFO *Info: Task information.

```
typedef struct _TSK_INFO
{
    U16 State;          //Task state: 0:Stop, 1: Run, 2: Step, 3:Step_p 4: ???
    U16 RunTimeErr;    //runtime error code when state is in ERROR state.
    U16 IP; //Register IP
    U16 SP; //Register SP
    U16 BP; //Register BP
    U16 MsgQueueSts; //Message queue status, refer to following definition
} TSK_INFO, *PTSK_INFO;
```

U16 MsgQueueSts: (Note: All tasks share only one message queue.)

BitNum	Status description
0	MPU_MSG_EMPTY: Message queue empty
1	MPU_MSG_FULL: Queue full
2	MPU_MSG_NOT_EMPTY
3~15	Reserved. 0

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
I32 boardId = 0;
```

```
I32 taskNum = 0;  
TSK_INFO info;  
  
//Get information of task 0  
ret = APS_get_task_info( boardId, taskNum, &info );
```

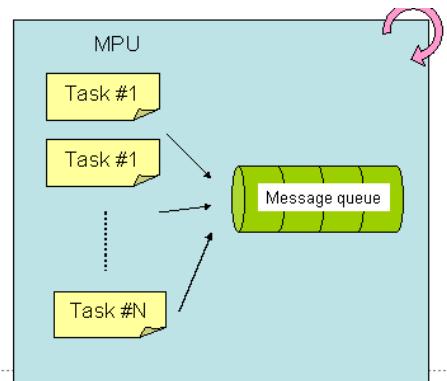
See also:

[APS_start_task\(\)](#); [APS_get_task_msg\(\)](#)

APS_get_task_msg	Get message of all tasks
------------------	--------------------------

Descriptions:

This function is used to get task message. All tasks share only one message queue. This is useful for debug. User could output some debug string to message queue.



Notice: AMP series don't support this function.

Syntax:

C/C++:

```
I32 APS_get_task_msg( I32 Board_ID, U16 *QueueSts, U16 *ActualSize, U8 *CharArr );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

U16 *QueueSts: Message queue status. Refer to following chart.

U16 *ActualSize: Actual return message size. [0~128]

U8 *CharArr: Return char message. Maximum array size is 128 bytes. (U8 CharArr[n = 128])

It depends on ActualSize, if n > = ActualSize, data is meaningless and could be ignored.

QueueSts definition:

BitNum	Status description
0	MPU_MSG_EMPTY: Message queue empty
1	MPU_MSG_FULL: Queue full
2	MPU_MSG_NOT_EMPTY
3~15	Reserved. 0

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;  
I32 boardId = 0;  
U16 queueSts = 0;  
U16 actualSize = 0;  
U8 charArr[128];  
  
//Get message of all tasks  
ret = APS_get_task_msg(boardId, &queueSts, actualSize, &charArr );;
```

See also:

[APS_start_task\(\)](#); [APS_get_task_info\(\)](#)

17. Gear / Gantry functions

APS_start_gear	Enable/Disable a specified gear mode
----------------	--------------------------------------

Descriptions:

This function is used to enable a specified gear mode. Two gear modes, including standard and gantry, are available for specified application.

Syntax:

C/C++:

```
I32 APS_start_gear (I32 Axis_ID, I32 Mode);
```

Visual Basic:

```
APS_start_gear (ByVal Axis_ID As Long, ByVal Mode As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Mode: Gear mode.

0: Disable, 1: Standard mode, 2: Gantry mode.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
...//  
APS_start_gear(Axis_ID, 0); //Disable gear.  
...//  
APS_start_gear(Axis_ID, 1); //Enable a standard gear mode.
```

See also:

[APS_get_gear_status\(\)](#)

APS_get_gear_status	Get gear status
---------------------	-----------------

Descriptions:

This function is used to get status of gear applicaiton.

Syntax:

C/C++:

I32 APS_get_gear_status(I32 Axis_ID, I32 *Status);

Visual Basic:

APS_get_gear_status(ByVal Axis_ID As Long, Status As Long) As Long

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Status: Gear status.

- 0: In disabling status.
- 1: In enabling status of standard mode.
- 2: In enabling status of gantry mode.

Return Values:

I32 Error code: Please refer to error code table.

Example:

I32 Status;

```
APS_get_gear_status(Axis_ID, &Status ); //Get Gear status
...//
```

See also:

[APS_start_gear\(\)](#)

18. Watch dog timer

APS_wdt_start	Start / Stop watch dog timer
---------------	------------------------------

Descriptions:

This function is used to start / stop watch dog timer.

Syntax:

C/C++:

```
I32 APS_wdt_start( I32 Board_ID, I32 TimerNo, I32 TimeOut );
```

Visual Basic:

```
APS_wdt_start (ByVal Board_ID As Long, ByVal TimerNo As Long, ByVal TimeOut As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TimerNo: Specify a timer number.

In PCI-8254/58, Timer No is 0.

I32 TimeOut:

Set 0 to disable watch dog timer.

Set a value by N(1 ~ 100) to enable watch dog timer.

TimeOut = N * 100 ms

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
```

```
I32 boardId = 0;
```

```
//Enable watch dog timer, TimeOut is 2 sec.
```

```
ret = APS_wdt_start( boardId, 0, 20 );
```

See also:

APS_wdt_get_timeout_period	Get a timeout period of watch dog timer
----------------------------	---

Descriptions:

This function is used to get a timeout period of watch dog timer. If timeout period is 0, watch dog timer is disabled. If timeout period is not 0, watch dog timer is enabled.

Syntax:

C/C++:

```
I32 APS_wdt_get_timeout_period( I32 Board_ID, I32 TimerNo, I32 *TimeOut );
```

Visual Basic:

```
APS_wdt_get_timeout_period(ByVal Board_ID As Long, ByVal TimerNo As Long, TimeOut As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TimerNo: Specify a timer number.

In PCI-8254/58, Timer No is 0.

I32 TimeOut:

0 means that watch dog timer is disabled.

A value N(1 ~ 100) means that watch dog timer is enabled.

TimeOut = N * 100 ms

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
I32 boardId = 0;
I32 timeOut = 0;
```

```
// Get a timeout period of watch dog timer
ret = APS_wdt_get_timeout_period ( boardId, 0, &timeOut );
```

See also:

APS_wdt_reset_counter	Reset counter of watch dog timer
-----------------------	----------------------------------

Descriptions:

This function is reset counter of watch dog timer to 0. The counter adds one count every DSP cycle. When the watch dog timer is enabled, user could periodoly reset the counter of watch dog timer to advoid trigger action event.

Syntax:

C/C++:

I32 APS_wdt_reset_counter(I32 Board_ID, I32 TimerNo);

Visual Basic:

APS_wdt_reset_counter (ByVal Board_ID As Long, ByVal TimerNo As Long) As Long

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TimerNo: Specify a timer number.

In PCI-8254/58, Timer No is 0.

Return Values:

I32 Error code: Please refer to error code table.

Example:

I32 ret = 0;

I32 boardId = 0;

//Reset the counter of watch dog timer

ret = APS_wdt_reset_counter (boardId, 0);

See also:

APS_wdt_get_counter	Get counter of watch dog timer
---------------------	--------------------------------

Descriptions:

This function is used to get counter of watch dog timer. If enabled, the counter adds one count every DSP cycle. If disabled, the counter shows zero.

Syntax:

C/C++:

```
I32 APS_wdt_get_counter( I32 Board_ID, I32 TimerNo, I32 *Counter );
```

Visual Basic:

```
APS_wdt_get_counter (ByVal Board_ID As Long, ByVal TimerNo As Long, Counter As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TimerNo: Specify a timer number.

In PCI-8254/58, Timer No is 0.

I32 Counter: If enabled, the counter adds one count every DSP cycle. If disabled, the counter shows zero.

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
I32 boardId = 0;
I32 Counter = 0;
```

```
// Get the counter of watch dog timer
ret = APS_wdt_get_counter ( boardId, 0, &Counter );
```

See also:

APS_wdt_set_action_event	Set action event of watch dog timer
--------------------------	-------------------------------------

Descriptions:

This function is used to set action event of watch dog timer. If time out, action event will be triggered.

Syntax:

C/C++:

```
I32 APS_wdt_set_action_event( I32 Board_ID, I32 TimerNo, I32 EventByBit );
```

Visual Basic:

```
APS_wdt_set_action_event (ByVal Board_ID As Long, ByVal TimerNo As Long, ByVal EventByBit As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TimerNo: Specify a timer number.

In PCI-8254/58, Timer No is 0.

I32 EventByBit: Set events

Bit0: Motor servo off

Bit1: Digital output off

Bit2: PWM off

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
```

```
I32 boardId = 0;
```

```
//Set action event. If time out, motor servo turns off.
```

```
ret = APS_wdt_set_action_event( boardId, 0, 1);
```

See also:

APS_wdt_get_action_event	Get action event of watch dog timer
--------------------------	-------------------------------------

Descriptions:

This function is used to get action event of watch dog timer.

Syntax:

C/C++:

```
I32 APS_wdt_get_action_event( I32 Board_ID, I32 TimerNo, I32 *EventByBit );
```

Visual Basic:

```
APS_wdt_get_action_event (ByVal Board_ID As Long, ByVal TimerNo As Long, EventByBit As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 TimerNo: Specify a timer number.

In PCI-8254/58, Timer No is 0.

I32 *EventByBit: Get events

Bit0: Motor servo off

Bit1: Digital output off

Bit2: PWM off

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
I32 ret = 0;
```

```
I32 boardId = 0;
```

```
I32 EventByBit = 0;
```

```
// Get action event of watch dog timer
```

```
ret = APS_wdt_get_action_event( boardId, 0, &EventByBit );
```

See also:

19. VAO/PWM functions (Laser function)

APS_set_vao_param	Set parameter to VAO table
-------------------	----------------------------

Descriptions:

The VAO module is a laser control application. It provides analog output and PWM signal according to corresponding linear speed.

This function is used to set VAO related parameters. All definitions of VAO parameters are described in [VAO parameter table](#).

You can also get VAO parameter setting using “APS_get_vao_param ()” function.

Syntax:

C/C++:

```
I32 APS_set_vao_param( I32 Board_ID, I32 Param_No, I32 Param_Val );
```

Visual Basic:

```
APS_set_vao_param (ByVal Board_ID As Long, ByVal Param_No As Long, ByVal Param_Val As Long) As  
Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Param_No: Parameter number. Refer to VAO parameter table.

I32 Param_Val: Parameter value. Refer to VAO parameter table.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
```

```
//Set output type of PWM mode to VAO table 0  
ret = APS_set_vao_param(Board_ID, 0x00, 1);
```

See also:

[APS_get_vao_param\(\)](#);

APS_get_vao_param	Get parameter of VAO table
-------------------	----------------------------

Descriptions:

The VAO module is a laser control application. It provides analog output and PWM signal according to corresponding linear speed.

This function is used to get VAO related parameters. All definitions of VAO parameters are described in [VAO parameter table](#).

You can also set VAO parameter using “APS_set_vao_param()” function.

Syntax:

C/C++:

```
I32 APS_get_vao_param( I32 Board_ID, I32 Param_No, I32 *Param_Val );
```

Visual Basic:

```
APS_get_vao_param(ByVal Board_ID As Long, ByVal Param_No As Long, Param_Val As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Param_No: Parameter number. Refer to VAO parameter table.

I32 Param_Val: Return parameter value. Refer to VAO parameter table.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
I32 Output_Type;

//Get output type of VAO table 0
ret = APS_set_vao_param(Board_ID, 0x00, &Output_Type );
```

See also:

[APS_set_vao_param\(\)](#);

APS_set_vao_table	Set VAO table
-------------------	---------------

Descriptions:

This function is used to set a set of VAO table. Users can implement a VAO application according to this table. User configures related minimum velocity, velocity interval, total points, and mapping output value for laser application. Therefore, “Velocity to Power” mapping lookup table will be built.

Notice that the mapping output value will be checked according to VAO output type when executing APS_check_vao_param(). If the mapping output value is invalid, it returns “ERR_ParametersInvalid”.

For example, if output type was set to voltage mode, the mapping output voltage can't large than 10000 mV. The range of the mapping output value is described as below:

Output type (0~3)	Output Range
0: Voltage(Reserved)	0 ~ 10000 mv Unit: 1 mv
1: PWM mode	0 ~ 2000 (0.0% ~ 100%) Unit: 0.05%
2: PWM frequency mode with fixed width	3 ~ 50M Hz Unit: 1 Hz
3: PWM frequency mode with fixed duty cycle	3 ~ 50M Hz Unit: 1 Hz

Syntax:

C/C++:

```
I32 FNTYPE APS_set_vao_table( I32 Board_ID, I32 Table_No, I32 MinVelocity, I32 VelInterval, I32
TotalPoints, I32 *MappingdataArray );
```

Visual Basic:

```
APS_set_vao_table ( ByVal Board_ID As Long , ByVal Table_No As Long, ByVal MinVelocity As Long,
ByVal VelInterval As Long, ByVal TotalPoints As Long, MappingdataArray As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Table_No: VAO table number. Range is 0 ~ 7.

I32 MinVelocity: Minimum linear speed.

I32 VellInterval: Speed interval.

I32 TotalPoints : Total points. Range is 1 ~ 32.

I32 *MappingdataArray: Output data array.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;  
I32 Minimum_Velocity;  
I32 Velocity_Interval;  
I32 TotalPoints = 32;  
I32 OutputVoltageData[32];  
  
//Configure linear speed  
//1st speed: 10000, 2nd speed: 20000, ...., 32th speed: 320000  
Minimum_Velocity = 10000;  
Velocity_Interval = 10000;  
TotalPoints = 32;  
  
//Configure mapping output voltage  
OutputVoltageData[0] = 500; // 1st voltage: 500 mv  
OutputVoltageData[1] = 600; // 2nd voltage: 600 mv  
.....  
OutputVoltageData[31] = 8600; // 32th voltage: 8600 mv  
  
//Set mapping table of Vao table 0  
Ret = APS_set_vao_table( Board_ID, 0, MinVelocity, VellInterval, TotalPoints, OutputVoltageData );
```

See also:

APS_set_vao_param(); APS_get_vao_param(); APS_switch_vao_table(); APS_start_vao();

APS_set_vao_param_ex	Set parameters via VAO structure
----------------------	----------------------------------

Descriptions:

This function is used to set parameters via VAO structure. This is a extension of APS_set_vao_param() and APS_set_vao_table(). By invoking APS_set_vao_param_ex(), user could set all parameters via VAO structure at once. By invoking APS_set_vao_param(), user could set a specified parameter one by one.

This function is also used to set mapping table to replace APS_set_vao_table(). User could configure related minimum velocity, velocity interval, total points, and mapping output value for laser application. Then, “Velocity to Power” mapping lookup table will be built.

Notice that both functions of APS_set_vao_param() and APS_set_vao_table() could be replaced by APS_set_vao_param_ex(). This is an option between them.

Syntax:

C/C++:

```
I32 APS_set_vao_param_ex( I32 Board_ID, I32 Table_No, VAO_DATA* VaoData );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Table_No: VAO table number. Range is 0 ~ 7.

VAO_DATA *VaoData: Vao structure for setting all parameters.

```
Typedef struct _VAO_DATA
{
    //Parameters perationion
    I32 outputType; //Output type, [0, 3]
    I32 inputType; //Input type, [0, 1]
    I32 config; //PWM configuration according to output type
    I32 inputSrc; //Input source by axis, [0, 0xf]

    //Mapping table perationion
    I32 minVel; //Minimum linear speed, [ positive ]
    I32 VelInterval; //Speed interval, [ positive ]
    I32 totalPoints; //Total points, [1, 32]
    I32 mappingDataArr[32]; //mapping data array
}
```

VAO_DATA, *PVAO_DATA;

VAO_DATA structure definition for setting VAO parameters

Variable name	Description	Value
outputType	Table output type	0: Voltage 1: PWM mode 2: PWM frequency mode with fixed width 3. PWM frequency mode with fixed duty cycle
inputType	Table input type	0: Feedback speed 1: Command speed
config	Configure PWM according to output type.	a. Mode 0 – Don't care b. Mode 1 – set a fixed frequency (3 ~ 50M Hz) c. Mode 2 – set a fixed Pulse Width (20 ~ 335544300 ns) d. Mode 3 – set a fixed duty cycle: N * 0.05 %. (N: 1 ~ 2000)
inputSrc	Specify axisID for VAO table. (linear speed on multi-axes)	Bit0: Axis 0 On Bit1: Axis 1 On Bit2: Axis 2 On Bit3: Axis 3 On

VAO_DATA structure definition for setting VAO mapping table

Variable name	Description	Value
minVel	Minimum linear speed	positive
velInterval	Speed interval	positive
totalPoints	Total points	1 ~ 32
mappingDataArr	mapping data array	Refer to following chart

The mapping data of VAO_DATA structure will be checked according to VAO output type. If the mapping data is invalid, it returns “ERR_ParametersInvalid”.

For example, if output type was set to voltage mode, the mapping output voltage can't be larger than 10000 mV. The range of mapping data is described as below:

Output type (0~3)	Output Range of mapping data
0: Voltage(Reserved)	0 ~ 10000 mv Unit: 1 mv
1: PWM mode	0 ~ 2000 (0.0% ~ 100%) Unit: 0.05%
2: PWM frequency mode with fixed width	3 ~ 50M Hz Unit: 1 Hz
3: PWM frequency mode with fixed duty cycle	3 ~ 50M Hz Unit: 1 Hz

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;  
VAO_DATA VaoData;  
  
VaoData.outputType = 1; // PWM mode  
VaoData.inputType = 0; //Feedback speed  
VaoData.Config = 1000; // set a fixed frequency of PWM mode, 1000hz  
VaoData.inputSrc = 0x03; //axis 0 & axis 1  
VaoData.minVel = 1000; // Minimum linear speed  
VaoData.velInterval = 100; //Speed interval  
VaoData.totalPoints = 2; //Two points  
//10% ~ 15% of PWM mode  
VaoData.mappingDataArr[0] = 200;  
VaoData.mappingDataArr[1] = 300;  
  
//Set parameters to table 0  
ret = APS_set_vao_param_ex(Board_ID, 0, &VaoData);
```

See also:

`APS_get_vao_param_ex(); APS_switch_vao_table(); APS_start_vao();`

APS_get_vao_param_ex	Get parameters via VAO structure
----------------------	----------------------------------

Descriptions:

This function is used to get parameters via VAO structure.

Refer to APS_set_vao_param_ex() for details.

Syntax:

C/C++:

```
I32 APS_get_vao_param_ex( I32 Board_ID, I32 Table_No, VAO_DATA* VaoData );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Table_No: VAO table number. Range is 0 ~ 7.

VAO_DATA *VaoData: Vao structure for setting all parameters. Refer to APS_set_vao_param_ex() for more details.

Typedef struct _VAO_DATA

```
{
    //Parameters perationion
    I32 outputType; //Output type, [0, 3]
    I32 inputType; //Input type, [0, 1]
    I32 config; //PWM configuration according to output type
    I32 inputSrc; //Input source by axis, [0, 0xf]

    //Mapping table perationion
    I32 minVel; //Minimum linear speed, [ positive ]
    I32 VelInterval; //Speed interval, [ positive ]
    I32 totalPoints; //Total points, [1, 32]
    I32 *mappingDataArr; //mapping data array
}
```

VAO_DATA, *PVAO_DATA;

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
VAO_DATA VaoData;
```

```
//Get VAO param structure of VAO table 0
ret = APS_get_vao_param_ex(Board_ID, 0, &VaoData );
```

See also:

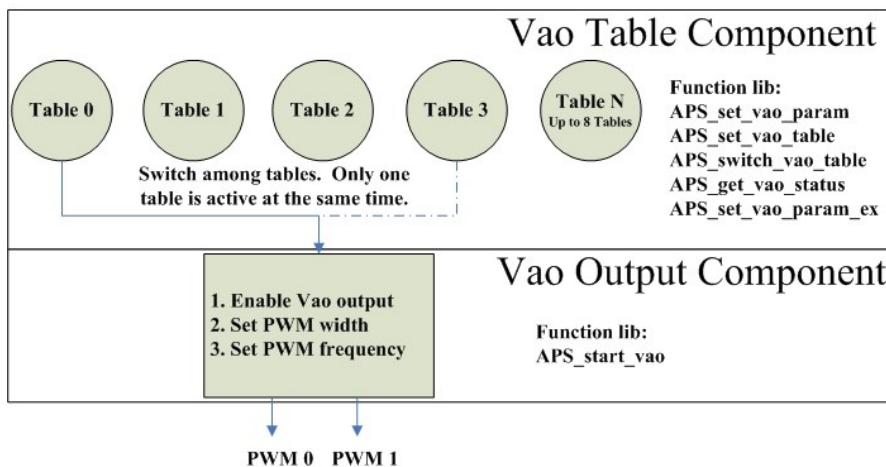
APS_set_vao_param_ex(); APS_start_vao()

APS_switch_vao_table	Switch to specified VAO table
----------------------	-------------------------------

Descriptions:

The VAO module is a laser control application. It provides analog output and PWM signal according to corresponding linear speed.

This function is used to switch to specified VAO table as following figure. There are up to 8 tables to be configurated. User could switch to each table among them. Only one table is active at the same time.



Notice that if point table is running on this point, it will automatically switch to the specified table by setting “opt” variable. Refer to APS_set_point_table(). In the other way, user also could manually switch to specified table by APS_switch_vao_table().

Syntax:

C/C++:

```
I32 APS_switch_vao_table( I32 Board_ID, I32 Table_No );
```

Visual Basic:

```
APS_switch_vao_table(ByVal Board_ID As Long, ByVal Table_No As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Table_No: VAO table number.

0 ~ 7: Table number.

-1: Disable all tables.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
```

```
ret = APS_switch_vao_table( Board_ID, 0 ); //Swtich to table 0
```

See also:

`APS_set_vao_param(); APS_get_vao_param(); APS_set_vao_table(); APS_start_vao()`

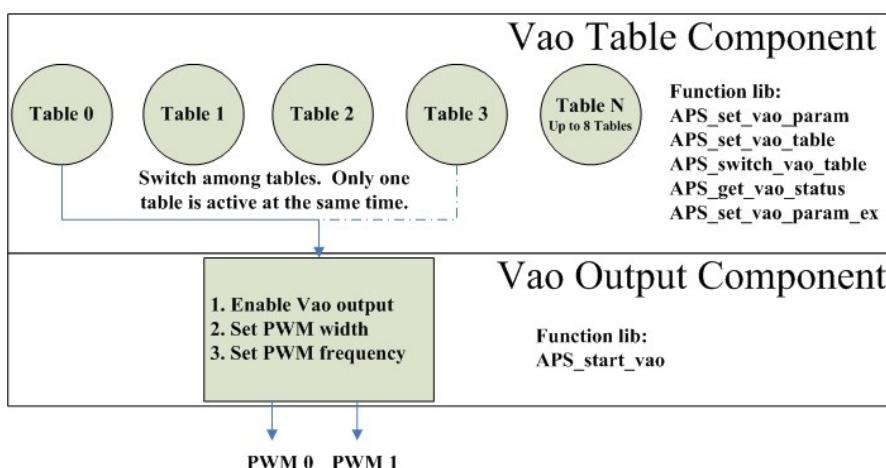
APS_start_vao	Enable VAO output channel.
---------------	----------------------------

Descriptions:

The VAO module is a laser control application. It provides analog output and PWM signal according to corresponding linear speed.

This function is used to enable VAO output channel as following figure. When VAO Output is enabling, analog voltage or PWM signal will output continuously according to corresponding linear speed.

User could also use APS_start_vao() to disable VAO output channel.



Syntax:

C/C++:

```
I32 APS_start_vao( I32 Board_ID, I32 Output_Ch, I32 Enable );
```

Visual Basic:

```
APS_start_vao (ByVal Board_ID As Long, ByVal Output_Ch As Long, ByVal Enable As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Output_Ch: PWM or Analog channel. Range is 0 ~ 5.

- 0: PWM channel 0
- 1: PWM channel 1
- 2: PWM channel 2 (only 8258)
- 3: PWM channel 3 (only 8258)
- 4: Analog output 3 (Pulse mode)
- 5: Analog output 7 (Pulse mode)

I32 Enable: Enable specified channel to output PWM/Voltage.

0: Disable. 1: Enable

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
```

```
ret = APS_start_vao( Board_ID, 0, 1 ); // Enable PWM channel 0 to output
```

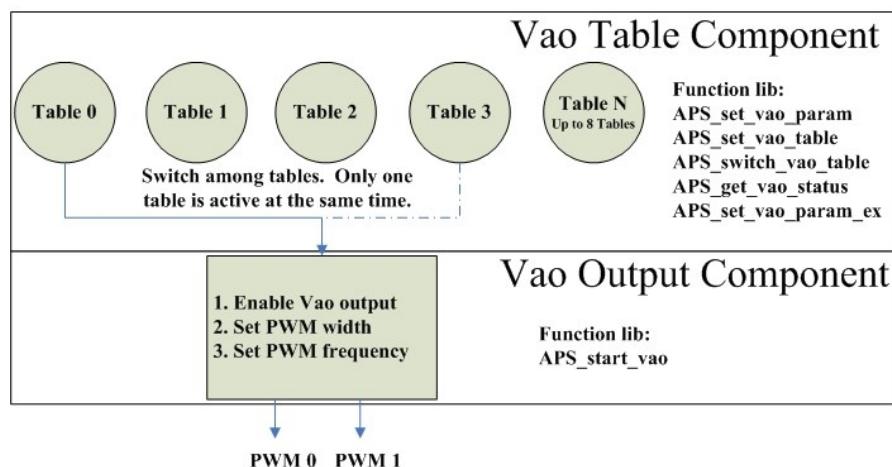
See also:

`APS_set_vao_param();APS_get_vao_param(); APS_set_vao_table (); APS_switch_vao_table()`

APS_get_vao_status	Get VAO status
--------------------	----------------

Descriptions:

This function is used to get VAO status. User could monitor which table is active and which PWM is enabling as following figure.



Syntax:

C/C++:

```
I32 APS_get_vao_status( I32 Board_ID, I32 *Status );
```

Visual Basic:

```
APS_get_vao_status (ByVal Board_ID As Long, Status As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 *Status: Get VAO status by bit.

Bit 0~7: Table 0~7 is active.

Bit 8~15: Reserved

Bit 16: PWM 0 is enabling.

Bit 17: PWM 1 is enabling.

Bit 18~: Reserved

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
```

```
I32 status;
```

```
//Get VAO status.  
Ret = APS_get_vao_status(Board_ID, &status );  
.....
```

See also:

APS_start_vao(); APS_switch_vao_table(); APS_start_vao

APS_check_vao_param	Check parameters setting of specified VAO table
---------------------	---

Descriptions:

This function is used to check table parameters of specified VAO table.

Syntax:

C/C++:

```
I32 APS_check_vao_param( I32 Board_ID, I32 Table_No, I32 *Status );
```

Visual Basic:

```
APS_check_vao_param (ByVal Board_ID As Long, ByVal Table_No As Long, Status As Long) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 Table_No: VAO table number. Range is 0 ~ 7.

I32 *Status: The checking status of parameters. Refer to VAO parameter table definition.

- 0: No any parameters error
- 1: Parameter of table input type is out of range. (VAO_TABLE_INPUT_TYPE)
- 2: Parameter of table output type is out of range. (VAO_TABLE_OUTPUT_TYPE)
- 3: Parameter of table input source is out of range. (VAO_TABLE_SRC)
- 4: Parameter of table pwm perationion is out of range. (VAO_TABLE_PWM_CONFIG)
- 5: Mapping table data is out of range. (Refer to APS_set_vao_table())

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
I32 Sts;

// Check parameters setting of specified VAO table
// Check parameters setting of VAO table 0
ret = APS_check_vao_param (Board_ID, 0, & Sts );
.......
```

See also:

[APS_set_vao_param\(\)](#); [APS_set_vao_table\(\)](#);

APS_set_pwm_on	Start to output PWM signal
----------------	----------------------------

Descriptions:

This function is used to output PWM signal. It is applied to activate laser, trigger, etc. There are two PWM channels which are TG1 and TG2 on main connector for PCI-8254.

Note that the PWM output (TG) is used by two function APIs, that are APS_set_pwm_on() and APS_start_vao(). Don't mix using them at the same time. Be sure that only one of them is enabled, specified PWM channel could rightly work.

Syntax:

C/C++:

I32 APS_set_pwm_on(I32 Board_ID, I32 PWM_Ch, I32 PWM_On);

Visual Basic:

APS_set_pwm_on(ByVal Board_ID As Long , ByVal PWM_Ch As Long , ByVal PWM_On As Long) As Long

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PWM_Ch: PWM output channel (TG) number. Zero based. Range is from 0 to 1.

I32 PWM_On: 0: PWM OFF, 1: PWM ON

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
I32 PWM_Ch = 0; //TG 0 is used
I32 Width = 2000 ns; //Pulse width is 2 us.
I32 Frequency = 10000 Hz;//pulse frequency is 10K Hz.
```

```
// Set pulse width to PWM channel 0
ret = APS_set_pwm_width( Board_ID, PWM_Ch, Width );
// Set pulse frequency to PWM channel 0
ret = APS_set_pwm_frequency( Board_ID, PWM_Ch, Frequency );
// Output PWM signal to activate laser
ret = APS_set_pwm_on ( Board_ID, PWM_Ch, 1 );
```

```
....  
// Stop outputting PWM signal  
Ret = APS_set_pwm_on ( Board_ID, PWM_Ch, 0 );
```

See also:

[APS_set_pwm_width\(\)](#); [APS_set_pwm_frequency\(\)](#); [APS_get_pwm_width\(\)](#);
[APS_get_pwm_frequency\(\)](#)

APS_set_pwm_width	Set pulse width to a PWM channel
-------------------	----------------------------------

Descriptions:

This function is used to set pulse width to specialized PWM channel.

Note that the range of pulse width is form 20 to 335544300. The unit is nano-second. The resolution of pulse width is 20 ns.

Syntax:

C/C++:

```
I32 APS_set_pwm_width( I32 Board_ID, I32 PWM_Ch, I32 Width );
```

Visual Basic:

```
I32 APS_set_pwm_width( ByVal Board_ID As Long , ByVal PWM_Ch As Long , ByVal Width As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PWM_Ch: PWM output channel (TG) number. Zero based. Range is from 0 to 1.

I32 Width: Pulse width. Unit: ns. Range is from 20 to 335544300.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
I32 PWM_Ch = 0; //TG 0 is used.
I32 Width = 2000 ns; //Pulse width is 2 us.
```

```
// Set pulse width to PWM channel 0
ret = APS_set_pwm_width( Board_ID, PWM_Ch, Width );
```

See also:

[APS_set_pwm_on\(\)](#); [APS_set_pwm_frequency\(\)](#); [APS_get_pwm_width\(\)](#); [APS_get_pwm_frequency\(\)](#)

APS_set_pwm_frequency	Set pulse frequency to a PWM channel
-----------------------	--------------------------------------

Descriptions:

This function is used to set pulse frequency to specialized PWM channel.

Note that the range of pulse frequency is form 3 to 50,000,000. The unit is Hz.

It may have slightly offset between actual output frequency and the frequency you set.

The actual frequency is according to following formula:

$$\text{Frequency} = \frac{1,000,000,000}{20 \times N}$$

N: 0 ~ 16777215 (a positive 32 bit value)

For example, User could set the frequency = 10005 Hz to the card by this function.

In side the function, It get the *N* = 5000 from the formula and send it to the controller, and the actual frequency output from the PWM will be 10000 Hz (According above formula).

Syntax:

C/C++:

```
I32 APS_set_pwm_frequency( I32 Board_ID, I32 PWM_Ch, I32 Frequency );
```

Visual Basic:

```
APS_set_pwm_frequency( ByVal Board_ID As Long , ByVal PWM_Ch As Long , ByVal Frequency As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PWM_Ch: PWM output channel (TG) number. Zero based. Range is from 0 to 1.

I32 Frequency: Pulse frequency. Unit: Hz. Range is from 3 to 50000000.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;  
I32 PWM_Ch = 0; //TG 0 is used.  
I32 Frequency = 10000 Hz; //Pulse frequency is 10k Hz.
```

```
// Set pulse frequency to PWM channel 0
ret = APS_set_pwm_frequency( Board_ID, PWM_Ch, Frequency);
```

See also:

`APS_set_pwm_on(); APS_set_pwm_width(); APS_get_pwm_width(); APS_get_pwm_frequency()`

APS_get_pwm_width	Get pulse width from a PWM channel
-------------------	------------------------------------

Descriptions:

This function is used to get pulse width from specialized PWM channel.

Syntax:

C/C++:

```
I32 APS_get_pwm_width( I32 Board_ID, I32 PWM_Ch, I32 *Width );
```

Visual Basic:

```
I32 APS_get_pwm_width( ByVal Board_ID As Long , ByVal PWM_Ch As Long , Width As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PWM_Ch: PWM output channel (TG) number. Zero based. Range is from 0 to 1.

I32 Width: Pulse width. Unit: ns. Range is from 20 to 335544300.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
I32 PWM_Ch = 0; //TG 0 is used.
I32 Width;

// Get pulse width from PWM channel 0
ret = APS_get_pwm_width( Board_ID, PWM_Ch, &Width );
```

See also:

[APS_set_pwm_on\(\)](#); [APS_set_pwm_width\(\)](#); [APS_set_pwm_frequency\(\)](#); [APS_get_pwm_frequency\(\)](#)

APS_get_pwm_frequency	Get pulse frequency from a PWM channel
-----------------------	--

Descriptions:

This function is used to get pulse frequency from specialized PWM channel.

Syntax:

C/C++:

```
I32 APS_get_pwm_frequency( I32 Board_ID, I32 PWM_Ch, I32 *Frequency );
```

Visual Basic:

```
APS_get_pwm_frequency( ByVal Board_ID As Long , ByVal PWM_Ch As Long , Frequency As Long ) As
Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 PWM_Ch: PWM output channel (TG) number. Zero based. Range is from 0 to 1.

I32 Frequency: Pulse frequency. Unit: Hz. Range is from 3 to 50000000.

Return Values:

I32 error code. Refer to error code table.

Example:

```
I32 ret;
I32 PWM_Ch = 0; //TG 0 is used.
I32 Frequency;
```

```
// Get pulse frequency from PWM channel 0
ret = APS_get_pwm_frequency( Board_ID, PWM_Ch, &Frequency);
```

See also:

[APS_set_pwm_on\(\)](#); [APS_set_pwm_frequency\(\)](#); [APS_set_pwm_width\(\)](#); [APS_get_pwm_width\(\)](#)

20. Circular limit functions

APS_set_circular_limit	Set configuration for circular limit
------------------------	--------------------------------------

Descriptions:

Set configuration for circular limit.

Syntax:

C/C++:

```
I32 FNTYPE APS_set_circular_limit (I32 Axis_A, I32 Axis_B, F64 Center_A, F64 Center_B, F64 Radius,  
I32 Stop_Mode, I32 Enable);
```

Parameters:

I32 Axis_A: axis ID 0~7

I32 Axis_B: axis ID 0~7

F64 Center_A: Center position of Axis_A.

F64 Center_B: Center position of Axis_B.

F64 Radius: Distance between circular limit boundary and center.

I32 Stop_Mode: Only Axis_A and Axis_B stop or all axes stop when circular limit is triggered.

I32 Enable: 0: Disable circular limit; 1: Enable circular limit

Return Values:

I32 error code. Refer to error code table.

Example:

```
ret = APS_initial( &BoardID, 0 );
```

```
// Set servo on
```

```
ret = APS_set_servo_on( 0, 1 );
```

```
ret = APS_set_servo_on( 1, 1 );
```

```
// Set command
```

```
ret = APS_set_command(0, 0);
```

```
ret = APS_set_command(1, 0);
```

```
// Enable circular limit
```

```
ret = APS_set_circular_limit( 0, 1, 0, 0, 100, 1, 1 );
```

```
// Set interrupt
Int_No = APS_set_int_factor( 0, 0, 17, 1 ); //Enable the interrupt factor
APS_int_enable( 0, 1 ); //Enable the interrupt main switch
// Start move
ret = APS_relative_move( 0, 500, 1000 );

// Wait interrupt
returnCode = APS_wait_single_int( Int_No, -1 );
if( returnCode == ERR_NoError )
{ //Interrupt occurred
    APS_reset_int( Int_No );
}

// Disable circular limit
ret = APS_set_circular_limit( 0, 1, 0, 0, 100, 1, 0 );
```

See also:

[APS_get_circular_limit](#)

APS_get_circular_limit	Get configuration for circular limit
------------------------	--------------------------------------

Descriptions:

Get configuration for circular limit.

Syntax:

C/C++:

```
I32 FNTYPE APS_get_circular_limit (I32 Axis_A, I32 Axis_B, F64 *Center_A, F64 *Center_B, F64  
*Radius, I32 *Stop_Mode, I32 *Enable);
```

Parameters:

I32 Axis_A: axis ID 0~7

I32 Axis_B: axis ID 0~7

F64 *Center_A: Center position of Axis_A.

F64 *Center_B: Center position of Axis_B.

F64 *Radius: Distance between circular limit boundary and center.

I32 *Stop_Mode: Only Axis_A and Axis_B stop or all axes stop when circular limit is triggered.

I32 *Enable: 0: Disable circular limit; 1: Enable circular limit

Return Values:

I32 error code. Refer to error code table.

Example:

N/A

See also:

[APS_set_circular_limit](#)

21. Manual pulser operation functions

APS_manual_pulser_start	Start manual pulser operation
-------------------------	-------------------------------

Descriptions:

This function is used to start manual pulser operation. It supports one set PA/PB pin to connect handy manual pulse generator and decoder for single axis position control. The decoder allows the input signal type from PA and PB pins being plus and minus pulses (CW/CCW), OUT/DIR, or 90 degrees phase difference signals (AB phase) respectively. User should select correct input signal type based on the specification of handy manual pulser generator carefully. If necessary, inverting PA and PB signals or changing counting direction are also allowed. These setting can be configured by axis parameters. After issuing “enable” command, the manual pulser operation will keep waiting to receive new input signals and then move motor immediately. The bit 29 of motion status is able to indicate the manual pulser operation is enabled or disabled. Before using this function, user should use APS_manual_pulser_velocity_move() to specify which axis and its maximum velocity. For safety consideration, this pulser operation will terminate immediately when

1. The “disable” command is issued by user.
2. The motion IO such that PEL/MEL, ALM, and EMG are triggered.

Syntax:

C/C++:

```
I32 FNTYPE APS_manual_pulser_start (I32 Board_ID, I32 Enable)
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial () .

I32 Enable: Enable pulser operation.

1: enable manual pulser operation

0: disable manual pulser operation

Return Values:

I32 error code. Refer to error code table.

Example:

```
ret = APS_manual_pulser_start( BoardID, 0 ); // Disable pulser process
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_MODE, 2 ); // Set input mode: 0: 1xAB; 2: 4xAB
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_LOGIC, 0 ); //Set logic: 0: InvPA = 0, InvPB = 0
ret = APS_set_axis_param( Axis, PRA_PSR_IPT_DIR, 0 ); // Set direction: 0: InvPA = 0, InvPB = 0

ret = APS_set_axis_param_f( Axis, PRA_PSR_RATIO_VALUE, 1 ); // Set ratio
ret = APS_set_axis_param_f( Axis, PRA_PSR_ACC, 123456 ); // Set acceleration
ret = APS_set_axis_param_f( Axis, PRA_PSR_JERK, 12345678 ); // Set jerk
```

```
ret = APS_manual_pulser_velocity_move( Axis, 12345 ); // Start velocity move
ret = APS_manual_pulser_start( BoardID, 1 ); // Enable pulser
```

See also:

[APS_manual_pulser_velocity_move](#);

APS_manual_pulser_velocity_move	Start velocity move in manual pulser operation
---------------------------------	--

Descriptions:

This function will start velocity move in manual pulser operation.

Syntax:

C/C++:

```
I32 FNTYPE APS_manual_pulser_velocity_move (I32 Axis_ID, F64 MaxVelocity)
```

Parameters:

I32 Axis_ID: Axis number 0~7

F64 MaxVelocity: Maximum speed in manual pulser operation. It's value should be larger than zero.

Return Values:

I32 error code. Refer to error code table.

Example:

See example in APS_manual_pulser_start

See also:

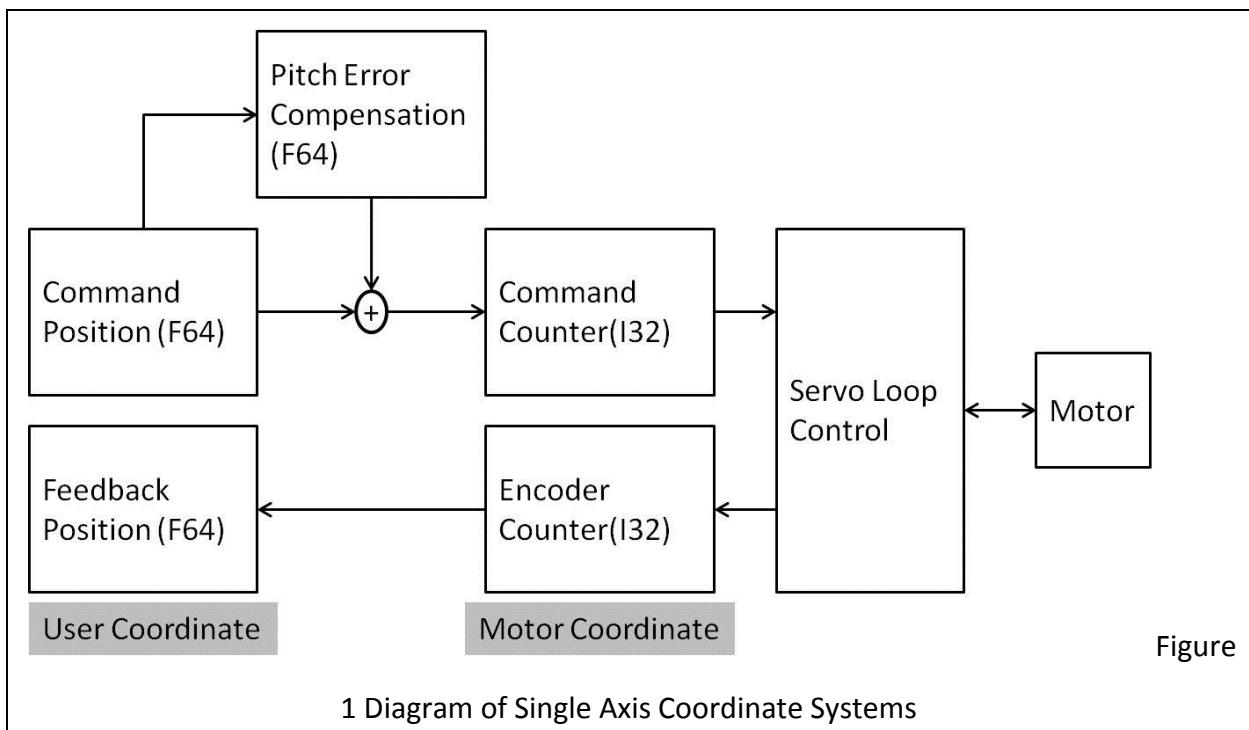
APS_manual_pulser_start;

22. Pitch error compensation functions

APS_set_pitch_table	Set data for pitch error compensation table.
---------------------	--

Description:

Figure 1 introduces two coordinate systems in PCI-8254/8: one is user coordinate, and the other is motor coordinate. Specifying arbitrary values for command position and feedback position in user coordinate are allowed here. Usually, after machine returns to its home position, both of them will be set to zero immediately. On the other hand, user is forbidden to change command counter and encoder counter in motor coordinate since they are used for servo loop control. It is noticed that the data type of these two coordinate are double and integer respectively. The process of pitch error compensation is introduced in Figure 1 now. Its input is current command position, and its output obtained from a look-up table is error compensation. The actual value of command counter will be the resultant of command position and pitch error compensation.



The pitch error compensation data is used for each compensation position at the intervals specified for each axis. The origin of compensation is the home position that the machine is returned. The compensation data is signed value and it is set with respect to home position (usually, compensation data at home position is zero). In order to perform pitch error compensation, it is also necessary to set configurations

such that minimum position, interval of compensation position, and total points. A pitch error compensation table can be built successfully based on these configurations and compensation data. There are two types of compensation methods for user: constant type and linear type. After finishing all settings above, user can enable the pitch error compensation via APS function. It should be noticed that if the machine stroke exceeds the specified range on either the positive direction or the negative direction, the pitch error compensation does not apply beyond the range, so the compensation value will be zero. The unit used in pitch error compensation is pulse (count). Figure 2 is an example of pitch error compensation table. The solid line and dashed line denotes the two compensation types. User specifies minimum position, interval, and total points are 0, 100, and 5. So the maximum position will be 500. And the compensation data at each compensation position are 0, 1, 2, -1, and 1. There is no compensation if command position is outside the range from 0 to 500. After machine returns home position (command position is zero), the error compensation is also zero.

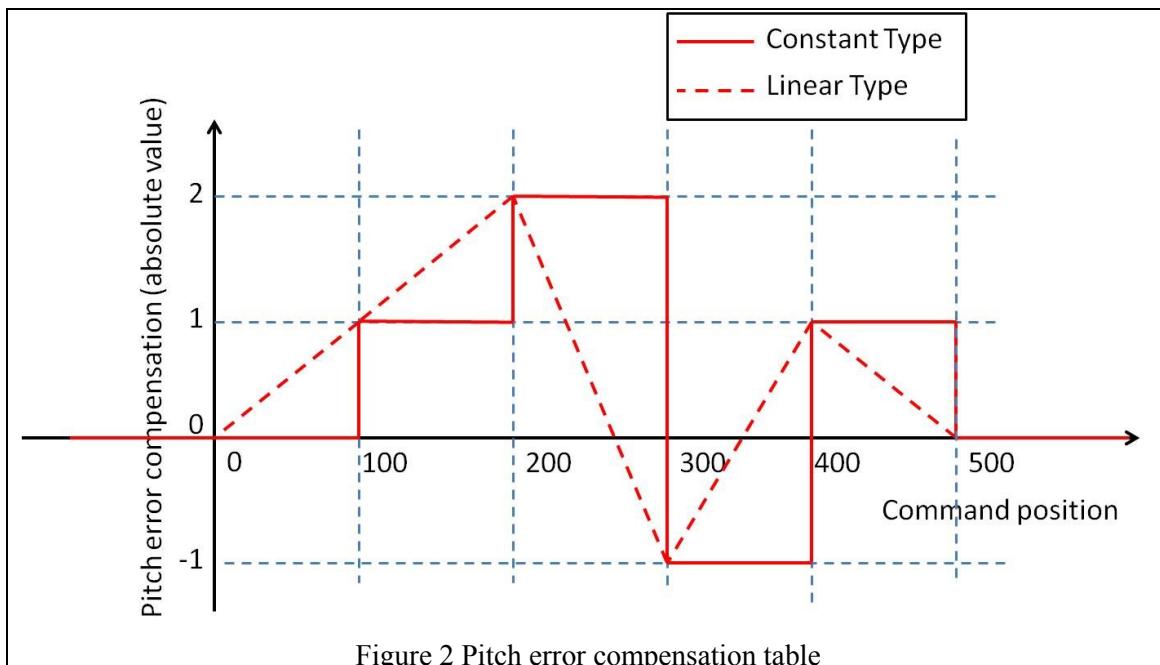


Figure 2 Pitch error compensation table

Table 1 shows the results of using pitch error compensation table in Figure 2. For example, if command position is set to 150.0, for constant compensation type the command counter will be 151; on the contrary, for linear compensation type, the command counter will be 152. This is due to truncation error.

	Command position(F64)	0.0	50.0	100.0	150.0	200.0	250.0	300.0	350.0	400.0	450.0	500.0
Constant compensation	Error Compensation (F64)	0.0	0.0	1.0	1.0	2.0	2.0	-1.0	-1.0	1.0	1.0	0.0
	Command counter(I32)	0	50	101	151	202	252	299	349	401	451	500
Linear compensation	Error Compensation (F64)	0.0	0.5	1.0	1.5	2.0	0.5	-1.0	0.0	1.0	0.5	0.0
	Command counter (I32)	0	51	101	152	202	251	299	350	401	451	500

Table 1 Results using different compensation method

Syntax:

C/C++:

```
I32 FNTYPE APS_set_pitch_table( I32 Axis_ID, I32 Comp_Type, I32 Total_Points, I32
MinPosition, U32 Interval, I32 *Comp_Data);
```

Parameters:

I32 Axis_ID: 0~7

I32 Comp_Type: Compensation type; 0: Constant compensation; 1: Linear compensation

I32 Total_Points: Total amounts of compensation data; Maximum value is 500.

I32 MinPosition: Minimum position in pitch error compensation table

U32 Interval: Interval between two compensation points in pitch error compensation table

I32 *Comp_Data: Compensation data in pitch error compensation table

Return Values:

I32 Error code: Please refer to error code table.

Example:

```
/*
-----*
Program Name: Pitch error compensation demo
Author:      Wei-li Chuang
Date:       2014/09/15
-----*/
void main()
{
    I32 ret; // function return
    I32 BoardID_InBits; // for initialization
    I32 Axis_ID = 0; // axis ID
    I32 MotionStatus; // motion status in bits
    I32 Comp_Data[5] = {0, 1, 2, -1, 1}; // pitch error compensation data
```

```

I32 CommandPosition; // command position
I32 CommandCount; // command counter
I32 Comp_Type = 0; // compensation type
I32 Total_Points = 5; // total points
I32 MinPosition = 0; // minimum command position
U32 Interval = 100; // interval
I32 i;

printf("/* Start pitch error compensation demo */ \n");

// Initialization
ret = APS_initial( &BoardID_InBits, 0 );
if(ret)
{
    printf("APS library initial fail! \n");
    goto END_PROGRAM;
}

// Set servo on
ret = APS_set_servo_on( Axis_ID, 1 );
if(ret)
{
    printf("Set servo on fail! \n");
    goto END_PROGRAM;
}

// Get current command position and command counter
APS_get_command(Axis_ID, &CommandPosition );
APS_get_command_counter(Axis_ID, &CommandCount );
printf("Command position = %d Command count = %d \n",CommandPosition, CommandCount );

// Start home process
printf("Return to home position... \n");
ret = APS_home_move( Axis_ID );
if(ret)
{
    printf("Start home fail! \n");
    goto END_PROGRAM;
}

// Check home is done
do{
    MotionStatus = APS_motion_status( Axis_ID ); //Get Motion status
}while ( ( MotionStatus>>5 & 0x1 ) == 0 );

// Get command position and command counter at home position
APS_get_command(Axis_ID, &CommandPosition );
APS_get_command_counter(Axis_ID, &CommandCount );
printf("Command position = %d Command count = %d \n",CommandPosition, CommandCount );

// Set pitch error compensation table
ret = APS_set_pitch_table( Axis_ID, Comp_Type, Total_Points, MinPosition, Interval,
Comp_Data);
if(ret)
{
    printf("Set pitch error compensation data and configuration fail! \n");
    goto END_PROGRAM;
}

```

```

}

// Start pitch error compensation
ret = APS_start_pitch_comp( Axis_ID, 1 );
if(ret)
{
    printf("Start pitch error compensation fail! \n");
    goto END_PROGRAM;
}

// Start PTP to test pitch error compensation
for( i=0; i<Total_Points; i++ )
{
    ret = APS_absolute_move( Axis_ID, 100+i*100, 10000 );
    if(ret)
    {
        printf("Start PTP fail! \n");
        goto END_PROGRAM;
    }

    // Check PTP is done
    do{
        MotionStatus = APS_motion_status( Axis_ID ); //Get Motion status
    }while ( ( MotionStatus>>5 & 0x1 ) == 0 );

    // Get command position and command counter
    APS_get_command(Axis_ID, &CommandPosition );
    APS_get_command_counter(Axis_ID, &CommandCount );
    printf("Command position = %d Command count = %d \n",CommandPosition, CommandCount );
}

// Set servo off
ret = APS_set_servo_on( Axis_ID, 0 );
if(ret)
{
    printf("Set servo off fail! \n");
    goto END_PROGRAM;
}

// Stop pitch error compensation
ret = APS_start_pitch_comp( Axis_ID, 0 );
if(ret)
{
    printf("Stop pitch error compensation fail! \n");
    goto END_PROGRAM;
}

END_PROGRAM:
    printf("/* Stop pitch error compensation demo */ \n");
    system("pause");

```

See also:

[APS_get_pitch_table](#), [APS_start_pitch_comp](#)

APS_get_pitch_table	Get data from pitch error compensation table.
----------------------------	---

Description:

Get configurations and data of pitch error compensation table

Syntax:

C/C++:

```
I32 FNTYPE APS_get_pitch_table( I32 Axis_ID, I32 *Comp_Type, I32 *Total_Points,
I32 *MinPosition, U32 *Interval, I32 *Comp_Data);
```

Parameters:

I32 Axis_ID: 0~7

I32 *Comp_Type: Compensation type; 0: Constant compensation; 1: Linear compensation

I32 *Total_Points: Amounts of compensation data

I32 *MinPosition: Minimum position in pitch error compensation table

U32 *Interval: Interval between two compensation points in pitch error compensation table

I32 *Comp_Data: Compensation data in pitch error compensation table

Return Values:

I32 Error code: Please refer to error code table.

Example:

See sample program in APS_set_pitch_table

See also:

APS_set_pitch_table, APS_start_pitch_comp

APS_start_pitch_comp	Start pitch error compensation
-----------------------------	--------------------------------

Description:

Start pitch error compensation table

Syntax:

C/C++:

```
I32 FNTYPE APS_start_pitch_comp( I32 Axis_ID, I32 Enable );
```

Parameters:

I32 Axis_ID: 0~7

I32 Enable: 0: Disable error compensation: 1: Enable error compensation

Return Values:

I32 Error code: Please refer to error code table.

Example:

See sample program in APS_set_pitch_table

See also:

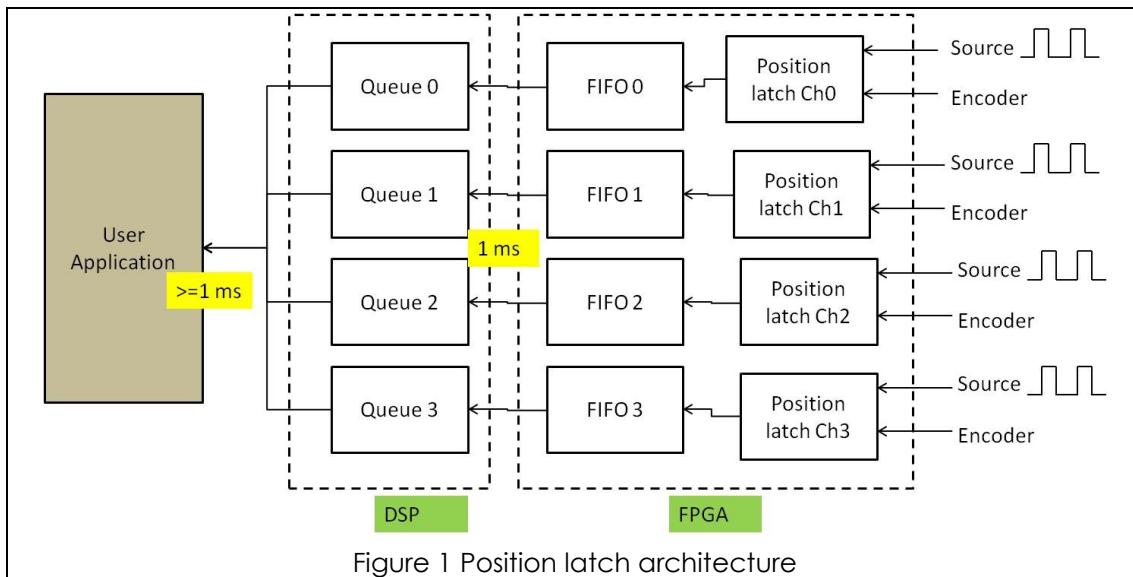
APS_set_pitch_table, APS_get_pitch_table

23. Position latch functions

APS_enable_ltc_fifo	Enable position latch process
---------------------	-------------------------------

Descriptions:

This function is used to enable position latch process. Figure 1 shows the position latch module architecture. There are four position latch channels and each of them has dedicated FIFO and queue. User has to specify the trigger source and encoder no. for position latch channel. The trigger source can be digital input signals or PWM pulse out. The rising, falling or both edge trigger modes are also supported here. These setting can be configured by the Latch parameter table.



Syntax:

C/C++:

```
I32 APS_enable_ltc_fifo( I32 Board_ID, I32 FLtcCh, I32 Enable );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 FLtcCh: The range of latch channel is 0~3

I32 Enable: 0: Disable; 1: Enable

Return Values:

I32 error code. Refer to error code table.

Example:

```
ret = APS_enable_ltc_fifo( BoardID, 0, 0 ); // Disable position latch  
  
ret = APS_reset_ltc_fifo( BoardID, 0 ); // Reset position latch queue  
  
ret = APS_set_ltc_fifo_param(BoardID, 0, LTC_IPT, 0xFFFF ); // Set input source  
ret = APS_set_ltc_fifo_param(BoardID, 0, LTC_ENC, 0 ); // Set EncoderNo  
ret = APS_set_ltc_fifo_param(BoardID, 0, LTC_LOGIC, 0 ); // Set Logic  
  
ret = APS_enable_ltc_fifo( BoardID, 0, 1 ); // Start position latch
```

See also:

[APS_set_ltc_fifo_param](#); [APS_get_ltc_fifo_param](#); [APS_reset_ltc_fifo](#)

APS_get_Ltc_fifo_point	Get latch point array
------------------------	-----------------------

Descriptions:

This function is used to get latch point array. Each latch point will include position in user coordinate and corresponding trigger source. The maximum latch point array size is 16.

Syntax:

C/C++:

```
I32 APS_get_Ltc_fifo_point( I32 Board_ID, I32 FLtcCh, I32 *ArraySize, LATCH_POINT  
*LatchPoint )
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 FLtcCh: The range of latch channel is 0~3

I32 * ArraySize: The size of latch point array; the maximum value of ArraySize is **16**.

LATCH_POINT * LatchPoint: Latched point array. The define of LATCH_POINT is shown below

```
typedef struct  
{  
    F64 position; // Latched position  
    I32 ltcSrcInBit; // Latch source: bit 0~7: DI; bit 8~11: trigger channel  
} LATCH_POINT;
```

Members

position

Latched position

ltcSrcInBit

- (1) bit 0~7: Digital input signal
- (2) bit 8~11: trigger channel

Return Values:

I32 error code. Refer to error code table.

Example:

N/A

See also:

APS_set_Ltc_fifo_param; APS_get_Ltc_fifo_param; APS_enable_Ltc_fifo

APS_set_Itc_fifo_param	Set latch parameter value
------------------------	---------------------------

Descriptions:

This function is used to set latch parameter value into Latch parameter table.

Syntax:

C/C++:

```
I32 APS_set_Itc_fifo_param( I32 Board_ID, I32 FLtcCh, I32 Param_No, I32 Param_Val );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 FLtcCh: The range of latch channel is 0~3

I32 Param_No: Latch parameter number; User can refer the Latch parameter table

I32 Param_Val: Latch parameter value

Return Values:

I32 error code. Refer to error code table.

Example:

N/A

See also:

APS_get_Itc_fifo_param;

APS_get_Itc_fifo_param	Get latch parameter value
------------------------	---------------------------

Descriptions:

This function is used to get latch parameter value from Latch parameter table.

Syntax:

C/C++:

```
I32 APS_get_Itc_fifo_param( I32 Board_ID, I32 FLtcCh, I32 Param_No, I32 *Param_Val )
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 FLtcCh: The range of latch channel is 0~3

I32 Param_No: Latch parameter number; User can refer the Latch parameter table

I32* Param_Val: Latch parameter value

Return Values:

I32 error code. Refer to error code table.

Example:

See also:

APS_set_Itc_fifo_param;

APS_reset_Itc_fifo	Reset latch queue and fifo
--------------------	----------------------------

Descriptions:

This function is used before starting position latch to reset or clear both Queue and FIFO that has been introduced in `APS_enable_Itc_fifo`. It is noticed that the position latch is also cleared simultaneously.

Syntax:

C/C++:

```
I32 APS_reset_Itc_fifo( I32 Board_ID, I32 FLtcCh )
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to `APS_initial()`.

I32 FLtcCh: The range of latch channel is 0~3

Return Values:

I32 error code. Refer to error code table.

Example:

See also:

APS_get_ltc_fifo_usage	Get latch queue used space
------------------------	----------------------------

Descriptions:

This function is used to get the latch queue used space which is introduced in APS_enable_ltc_fifo.

Syntax:

C/C++:

```
I32 APS_get_ltc_fifo_usage( I32 Board_ID, I32 FLtcCh, I32 *Usage )
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 FLtcCh: The range of latch channel is 0~3

I32 *Usage: Queue used space

Return Values:

I32 error code. Refer to error code table.

Example:

See also:

APS_get_ltc_fifo_free_space	Get latch queue free space
-----------------------------	----------------------------

Descriptions:

This function is used to get latch queue used space which is introduced in APS_enable_ltc_fifo.

Syntax:

C/C++:

```
I32 APS_get_ltc_fifo_free_space( I32 Board_ID, I32 FLtcCh, I32 *FreeSpace )
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 FLtcCh: The range of latch channel is 0~3

I32 * FreeSpace: Queue free space

Return Values:

I32 error code. Refer to error code table.

Example:

See also:

APS_get_ltc_fifo_status	Get latch queue and fifo status
-------------------------	---------------------------------

Descriptions:

This function is used to get latch queue and fifo status.

Syntax:

C/C++:

```
I32 APS_get_ltc_fifo_status( I32 Board_ID, I32 FLtcCh, I32 *Status );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 FLtcCh: The range of latch channel is 0~3

I32 * Status: the bit define of status is

Bit0 = 0: FIFO is not empty, 1: FIFO is Empty (cyclic update)

Bit1 = 0: FIFO is not full, 1: FIFO is full (cyclic update)

Bit2 = X

Bit3 = 0: FIFO is not overflow, 1: FIFO is Overflow (clear by reset Queue and FIFO)

Bit4 = 0: Queue is not empty, 1: Queue is empty (cyclic update)

Bit5 = 0: Queue is not full, 1: Queue is full (cyclic update)

Bit6 = 0: Queue is not overflow, 1: Queue is overflow (clear by reset Queue and FIFO)

Return Values:

I32 error code. Refer to error code table.

Example:**See also:**

24. Backlash functions

APS_set_backlash_en	Enable/Disable backlash
---------------------	-------------------------

Descriptions:

This function is used to enable or disable backlash function. The backlash compensation is applied to that commanded axis movement by superimposition as the direction of axis movement is reversed. Users should configure the axis parameters PRA_BKL_DIST and PRA_BKL_CNSP prior to enable backlash. The former denotes the backlash compensation value, and the latter denotes the backlash compensation increment value every cycle. If this function is already enabled, it is not allowed to enable again. If user set *Enable* as 2, the output velocity will be limited by axis parameter PRA_VM (0x12). The motion status bit 30 is used to indicate backlash compensation is in operation or not.

The tables below show two backlash *Enable* mode examples: The backlash axis parameters are both 1000. The *Direction* denotes the motion status *DIR* bit, 1 for positive, and 0 for negative. And the *Command* and *Position* denote the command position and feedback position in user coordinate respectively. The *Encoder* denotes the encoder values of motor coordinate. It is supposed backlash is enabled after setting Servo ON, then run the point-to-point function step by step, and the backlash compensation is effected when the *Direction* is reversed.

Table 1 An Example of setting *Enable* as 1: user coordinate position is NOT aligned

Step	Description	Direction	Command	Position	Encoder	Note
1	Initial condition	Positive	0	0	0	Servo ON
2	Forward 1000 pulse	Positive	1000	1000	1000	
3	Forward 1000 pulse	Positive	2000	2000	2000	
4	Backward 1000 pulse	Negative	1000	0	0	Backlash compensation
5	Backward 1000 pulse	Negative	0	-1000	-1000	
6	Backward 1000 pulse	Negative	-1000	-2000	-2000	
7	Forward 1000 pulse	Positive	0	0	0	Backlash compensation

Table 2 An Example of setting *Enable* as 2: user coordinate position is aligned

Step	Description	Direction	Command	Position	Encoder	Note
1	Initial condition	Positive	0	0	0	Servo ON
2	Forward 1000 pulse	Positive	1000	1000	1000	
3	Forward 1000 pulse	Positive	2000	2000	2000	
4	Backward 1000 pulse	Negative	1000	1000	0	Backlash compensation
5	Backward 1000 pulse	Negative	0	0	-1000	
6	Backward 1000 pulse	Negative	-1000	-1000	-2000	
7	Forward 1000 pulse	Positive	0	0	0	Backlash compensation

Syntax:

C/C++:

I32 FNTYPE APS_set_backlash_en (I32 Axis_ID, I32 Enable);

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Enable: see description below

Enable	Description
0	Disable backlash compensation
1	<ul style="list-style-type: none">a. Enable backlash compensationb. User coordinate position is NOT alignedc. The motion status bit 30 is used to indicate backlash compensation is in operation (=0) or not (=1).
2	<ul style="list-style-type: none">a. Enable backlash compensationb. User coordinate position is alignedc. Output velocity is limited by axis parameter PRA_VM (0x12)d. The motion status bit 30 is used to indicate backlash compensation is in operation (=0) or not (=1).

Return Values:

I32 error code. Refer to error code table.

Example:**See also:**

APS_get_backlash_en	Check backlash is enabled/disabled
---------------------	------------------------------------

Descriptions:

This function is used to check backlash is enabled or disabled.

Syntax:

C/C++:

```
I32 FNTYPE APS_get_backlash_en( I32 Axis_ID, I32 *Enable );
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Enable: see description below

Enable	Description
0	Disable backlash compensation
1	<ul style="list-style-type: none">a. Enable backlash compensationb. User coordinate position is NOT alignedc. The motion status bit 30 is used to indicate backlash compensation is in operation (=0) or not (=1).
2	<ul style="list-style-type: none">a. Enable backlash compensationb. User coordinate position is alignedc. Output velocity is limited by axis parameter PRA_VM (0x12)d. The motion status bit 30 is used to indicate backlash compensation is in operation (=0) or not (=1).

Return Values:

I32 error code. Refer to error code table.

Example:**See also:**

25. Board Parameter Table

PCI-8254/58 Board parameter table

PCI-8254/58 Board parameter table				
NO.	Define	Description	Parameter data meaning	Default
00h	PRB_EMG_LOGIC	EMG Logic	0:Not inverse 1:Inverse	0
14h	PRB_DO_LOGIC	DO logic	0: no invert; 1: invert	0
15h	PRB_DI_LOGIC	DI logic	0: no invert; 1: invert	0
101h	PRS_EMG_MODE	EMG condition mode	0 (EMO): Servo off directly 1 (EMS): Emergency stop without servo off	0
110h	PRB_PWM0_MAP_DO	Enable the mapping between PWM0 & Do. Specify a Do channel to map PWM0. Select its mapping logic between PWM0 & Do. NOTE: When disabling this parameter, the PWM output of VAO table may also be disabled.	-1: Disable mapping Positive number: Enable mapping Bit0~7: Specify a Do channel. Bit8: Select logic. Set to 1: Turning on Do maps enabling PWM0. Turning off Do maps disabling PWM0. Set to 0: Turning on Do maps disabling PWM0. Turning off Do maps enabling PWM0.	-1
111h	PRB_PWM1_MAP_DO	Enable the mapping between PWM1 & Do. Specify a Do channel to map PWM1. Select its mapping logic between PWM1 & Do. NOTE: When disabling this parameter, the PWM output of VAO table may also be disabled.	-1: Disable mapping Positive number: Enable mapping Bit0~7: Specify a Do channel. Bit8: Select logic. Set to 1: Turning on Do maps enabling PWM1. Turning off Do maps disabling PWM1. Set to 0: Turning on Do maps disabling PWM1. Turning off Do maps enabling PWM1.	-1
112h	PRB_PWM2_MAP_DO	Enable the mapping between PWM2 & Do. Specify a Do channel to map PWM2. Select its	-1: Disable mapping Positive number: Enable mapping Bit0~7: Specify a Do channel. Bit8: Select logic. Set to 1: Turning	-1

		<p>mapping logic between PWM2 & Do.</p> <p>NOTE: When disabling this parameter, the PWM output of VAO table may also be disabled.</p>	<p>on Do maps enabling PWM2.</p> <p>Turning off Do maps disabling PWM2. Set to 0: Turning on Do maps disabling PWM2. Turning off Do maps enabling PWM2.</p>	
113h	PRB_PWM3_MAP_DO	<p>Enable the mapping between PWM3 & Do.</p> <p>Specify a Do channel to map PWM3. Select its mapping logic between PWM3 & Do.</p> <p>NOTE: When disabling this parameter, the PWM output of VAO table may also be disabled.</p>	<p>-1: Disable mapping</p> <p>Positive number: Enable mapping</p> <p>Bit0~7: Specify a Do channel.</p> <p>Bit8: Select logic. Set to 1: Turning on Do maps enabling PWM3.</p> <p>Turning off Do maps disabling PWM3. Set to 0: Turning on Do maps disabling PWM3. Turning off Do maps enabling PWM3.</p>	-1

26. Axis Parameter Table

PCI-8254/58 Axis parameter table

PCI-8254/58 axis parameter table					
NO.	Define	Description	Value	Default	Type
00h (0)	PRA_EL_LOGIC	PEL/MEL input logic	0:Not inverse 1:Inverse	0	I32
01h (1)	PRA_ORG_LOGIC	ORG input logic	0:Not inverse 1:Inverse	0	I32
02h (2)	PRA_EL_MODE	Stop mode when EL turns on. Note: Deceleration profile is given according to PRA_SD_DEC.	0: Deceleration stop 1: Stop immediately	0	I32
03h (3)	PRA_MDM_COND1	Motion done condition (Affective with motion stats NSTP bit)	0: Command done; 1: Command done with INP	0	I32
04h (4)	PRA_ALM_LOGIC	Set ALM logic	0: Low active 1: High active	0	I32
06h (6)	PRA_EZ_LOGIC	Set EZ logic	0: Low active 1: High active	0	I32
07h (7)	PRA_SD_DEC	Stop deceleration including EL stop, stop function and multi-stop().	Unit: pulse/sec ² 0.0	10000000 0.0	F64
08h (8)	PRA_SPEL_EN	Soft PEL enable	0: Disable 1: Reserved 2: Soft-Limit (SPEL)	0	I32
09h (9)	PRA_SMEL_EN	Soft MEL enable	0: Disable 1: Reserved 2: Soft-Limit (SMEL)	0	I32
0Ah (10)	PRA_SPEL_POS	Soft-end-limit for positive end [F64]	Unit: pulse	100000.0	F64
0Bh (11)	PRA_SMEL_POS	Soft-end-limit for negative end [F64]	Unit: pulse	-100000.0	F64
10h (16)	PRA_HOME_MODE	Home mode setting	0: home mode 1 (ORG) 1: home mode 2 (EL) 2: home mode 3 (EZ)	0	I32
11h (17)	PRA_HOME_DIR	Homing direction	0: positive direction 1: negative direction	0	I32
12h (18)	PRA_HOME_CURVE	Home move acceleration / deceleration speed pattern	[0.0 ~ 1.0] 0: T curve 1: S curve	0.5	F64
13h (19)	PRA_HOME_ACC	Home move acceleration /	Unit: pulse/sec ²	10000.0	F64

		Deceleration rate			
14h (20)	PRA_HOME_VS	Homing starting velocity	Unit: pulse/sec	0.0	F64
15h (21)	PRA_HOME_VM	Homing maximum velocity	Unit: pulse/sec	1000.0	F64
16h (22)	Reserved				
17h (23)	PRA_HOME_SHIFT	The distance shift from EZ, EL, ORG signal.	Unit: pulse	0.0	F64
18h (24)	PRA_HOME_EZA	EZ alignment enable	0: Not enable 1: Enable	0	I32
19h (25)	PRA_HOME_VO	Homing leave home velocity	Unit: pulse/sec	0.0	F64
1A (26)	Reserved				
1B (27)	PRA_HOME_POS	User defined position after homing.	Unit: pulse	0.0	F64
20h (32)	PRA_SF	move acceleration / deceleration speed pattern	[0.0 ~ 1.0] 0: T curve 1: S curve	0.0	F64
21h (33)	PRA_ACC	Acceleration rate	Unit: pulse/sec ²	10000000. 0	F64
22h (34)	PRA_DEC	Deceleration rate	Unit: pulse/sec ²	10000000. 0	F64
23h (35)	PRA_VS	Start velocity	Unit: pulse/sec	0.0	F64
24h (36)	PRA_VM	Maximum velocity	Unit: pulse/sec	1000.0	F64
25h (37)	PRA_VE	End velocity	Unit: pulse/sec	0.0	F64
2Ah (42)	PRA_PRE_EVENT_DIST	Pre-event distance	Unit: pulse	0.0	F64
2Bh (43)	PRA_POST_EVENT_DIST	Post-event distance	Unit: pulse	0.0	F64
32h(50)	PRA_PT_STP_DO_EN	Enable Do when point table stopping/pausing	0: Disable 1: Enable	0	I32
33h(51)	PRA_PT_STP_DO	Set Do value when Point table stopping	0: Set to 0 1: Set to 1	0	I32
34h(52)	PRA_PWM_OFF	Disable the specified PWM output when ASTP input signal is active.	0: Disable. No action when ASTP is active. 1: PWM_CH0 output will be disabled when ASTP is active. 2: PWM_CH1 output will be disabled when ASTP is active.	0	I32
35h(53)	PRA_DO_OFF	Set Do value when ASTP input	0: Disable. No action	0	I32

		signal is active.	when ASTP is active. Bit0~3: select DO channel. Bit8: Set Do output value when ASTP is active. (*1)		
40h (64)	PRA_JG_MODE	Jog mode	0:Continuous mode, 1:Step mode	0	I32
41h (65)	PRA_JG_DIR	Jog move direction	0: Positive direction 1: Negative direction	0	I32
42h (66)	PRA_JG_SF	Jog move acceleration / deceleration speed pattern	0 ~ 1	0.0	F64
43h (67)	PRA_JG_ACC	Jog move acceleration	value > 0	2000.0	F64
44h (68)	PRA_JG_DEC	Jog move deceleration	value > 0	2000.0	F64
45h (69)	PRA_JG_VM	Jog move max velocity	value > 0	1000.0	F64
46h (70)	PRA_JG_OFFSET	Jog offset, for step mode	value >= 0	1000.0	F64
47h (71)	PRA_JG_DELAY	Jog delay, for step mode, microsecnod	0 ~ 10,000,000 microsecnod	500000	I32
48h (72)	PRA_JG_MAP_DI_EN	(Enable Digital input map to jog command signal	1: Enable 0: Disable Bit 0: Active for PRA_JG_P_JOG_DI Bit 1: Active for PRA_JG_N_JOG_DI Bit 2: Active for PRA_JG_JOG_DI	0	I32
49h (73)	PRA_JG_P_JOG_DI	(I32) Mapping configuration for positive jog and digital input.	DI Channel 0 ~ 23	0	I32
4Ah (74)	PRA_JG_N_JOG_DI	(I32) Mapping configuration for negative jog and digital input.	DI Channel 0 ~ 23	1	I32
4Bh (75)	PRA_JG_JOG_DI	(I32) Mapping configuration for jog and digital input.	DI Channel 0 ~ 23	2	I32
51h (81)	PRA_SINP_WDW	Soft-INP window, unit (pulse count)	0: Disable 1~2147483647: Enable INP window	0	I32
52h (82)	PRA_SINP_STBT	Soft-INP stable time, unit	[0~10000] ms	0	I32

		(mill-second)			
60h (96)	PRA_GEAR_MASTER	<p>Select gearing master: [0h~7h, 20h~27h]</p> <p>0x00~0x07: Axis 0~7 command position deviation</p> <p>0x20~0x27: Axis 0~7 feedback position deviation</p> <p>NOTE:</p> <ol style="list-style-type: none"> 1. Setting master and slave in same axis is not allowed. For example, APS_set_axis_param(0 , 0x60, 0); 2. Relationship between master axis and slave axis can not become a loop. For example, APS_set_axis_param(0 , 0x60, 1); APS_set_axis_param(1 , 0x60, 0); 	0x00~0x07: Axis 0~7 command position deviation 0x20~0x27: Axis 0~7 feedback position deviation	0x00	I32
61h (97)	PRA_GEAR_ENGAGE_R ATE	<p>Gear engage rate unit = 1 / sec, [>= 0.0]</p> <p>(*) When cycle time changed, this parameter must be reset. (Cycle time = 1ms)</p>		1000.0	F64
62h (98)	PRA_GEAR_RATIO	Gear ratio	[-10000.0 ~ 10000.0]	1.0	F64
63h (99)	PRA_GANTRY_PROTECT _1	<p>E-gear gantry mode protection level 1 [>= 0.0]</p> <p> fbk_master – fbk_slave >= value, sd-stop motion</p>	0.0: disable gantry error check. >0.0: Enable gantry error check.	0.0	F64
64h (100)	PRA_GANTRY_PROTECT _2	<p>E-gear gantry mode protection level 2 [>= 0.0]</p> <p> fbk_master – fbk_slave >= value, both servo-off</p>	0.0: disable gantry error check. >0.0: Enable gantry error check,	0.0	F64
65h(101)	PRA_EGEAR_MASTER	Select gearing master axis	Axisid: Specified a existed axis ID from 0 to 65535 virtual axisid: 0x7fffffff	0x7fffffff	I32
66h(102)	PRA_EGEAR_SOURCE	Select gearing master source	0: command position	0	I32

			deviation 1: feedback position deviation		
80h (128)	PRA_PLS_IPT_MODE	Pulse input mode	0 :DEC_OUT_DIR_MO DE0 1 :DEC_CW_CCW_MO DE0 2 :DEC_1XAB 3 :DEC_2XAB 4 :DEC_4XAB 5 :DEC_OUT_DIR_MO DE1 6:DEC_OUT_DIR_MO DE2 7 :DEC_OUT_DIR_MO DE3 8 :DEC_CW_CCW_MO DE1	0	I32
81h (129)	PRA_PLS_OPT_MODE	Pulse output mode	0x00: OUT/DIR 0x01: CW/CCW	0	I32
83h (131)	PRA_ENCODER_FILTER	Encoder filter		0	I32
85h (133)	PRA_ENCODER_DIR	Encoder direction		0	I32
86h (134)	PRA_POS_UNIT_FACTO R			1	F64
88h (136)	PRA_MOVE_RATIO	Move ratio		1.0	F64
90h (144)	PRA_KP_GAIN(*2)	PID controller Kp gain		0	I32
91h (145)	PRA_KI_GAIN(*2)	PID controller Ki gain		0	I32
92h (146)	PRA_KD_GAIN(*2)	PID controller Kd gain		0	I32
93h (147)	PRA_KVFF_GAIN(*2)	Velocity feed-forward gain		0	I32
9Ah (154)	PRA_KAFF_GAIN(*2)	Acceleration feedforward gain		0	I32
9Bh(155)	PRA_KP_SHIFT(*2)	Proportional control result shift	>0: left shift; <0: right shift; =0: no shift; [31 ~ -31]	-5	I32
9Ch(156)	PRA_KI_SHIFT(*2)	Integral control result shift	>0: left shift; <0: right shift; =0: no shift; [31 ~ -31]	-15	I32

9Dh(157)	PRA_KD_SHIFT(*2)	Derivative control result shift	>0: left shift; <0: right shift; =0: no shift; [31 ~ -31]	0	I32
9Eh(158)	PRA_KVFF_SHIFT(*2)	Velocity feed-forward control result shift	>0: left shift; <0: right shift; =0: no shift; [31 ~ -31]	0	I32
9Fh(159)	PRA_KAFF_SHIFT(*2)	Acceleration feed-forward control result shift	>0: left shift; <0: right shift; =0: no shift; [31 ~ -31]	0	I32
A0h(160)	PRA_PID_SHIFT(*2)	PID control result shift	>0: left shift; <0: right shift; =0: no shift; [31 ~ -31]	-5	I32
120h (288)	PRA_SERVO_V_BIAS (*2)	Servo voltage offset		0	F64
123h (291)	PRA_SERVO_V_LIMIT (*2)	Voltage saturation output limit		10.0	F64
124h (292)	PRA_ERR_POS_LEVEL	Error counter check level		90000	F64
125h (293)	PRA_SERVO_V_INVERS E(*2)	Control voltage inverse,	1: inverse. 0:Not inverse	0	
129h(297)	PRA_BKL_DIST	Backlash compensation value		0.0	F64
12Ah(298)	PRA_BKL_CNSP	Backlash compensation increment value every cycle		0.0	F64
12Bh (299)	PRA_INTEGRAL_LIMIT (*2)	Integral limit		21474836 47	I32
132h (306)	PRA_BIQUAD0_A1(*2)	Biquad filter0 coefficient A1		0	F64
133h (307)	PRA_BIQUAD0_A2(*2)	Biquad filter0 coefficient A2		0	F64
134h (308)	PRA_BIQUAD0_B0(*2)	Biquad filter0 coefficient B0		1	F64
135h (309)	PRA_BIQUAD0_B1(*2)	Biquad filter0 coefficient B1		0	F64
136h (310)	PRA_BIQUAD0_B2(*2)	Biquad filter0 coefficient B2		0	F64
137h (311)	PRA_BIQUAD0_DIV(*2)	Biquad filter0 coefficient DIVDER		1	F64
138h (312)	PRA_BIQUAD1_A1(*2)	Biquad filter1 coefficient A1		0	F64
139h (313)	PRA_BIQUAD1_A2(*2)	Biquad filter1 coefficient A2		0	F64

13Ah (314)	PRA_BIQUAD1_B0(*2)	Biquad filter1 coefficient B0		1	F64
13Bh (315)	PRA_BIQUAD1_B1(*2)	Biquad filter1 coefficient B1		0	F64
13Ch (316)	PRA_BIQUAD1_B2(*2)	Biquad filter1 coefficient B2		0	F64
13Dh (317)	PRA_BIQUAD1_DIV(*2)	Biquad filter1 coefficient DIVDER		1	F64
160h(352)	PRA_PSR_IPT_MODE	Pulser input mode	0: 1xAB; 1: 2xAB; 2: 4xAB 3: CW/CCW 4: OUT/DIR mode;	0	I32
161h(353)	PRA_PSR_IPT_LOGIC	Pulser EA and EB logic	0: EAinv = 0, EBinv = 0 1: EAinv = 0, EBinv = 1 2: EAinv = 1, EBinv = 0 3: EAinv = 1, EBinv = 1	0	I32
162h(354)	PRA_PSR_IPT_DIR	Pulser input direction	0: not inverse; 1: inverse	0	I32
163h(355)	PRA_PSR_RATIO_VALUE	Pulser ratio	Non zero value	1	F64
168h(360)	PRA_PSR_ACC	Pulser acceleration	Unit: pulse/sec ²	1000000	F64
169h(361)	PRA_PSR_JERK	Pulser jerk	Unit: pulse/sec ³	10000000 00	F64
211h (529)	PRA_INP_LOGIC	Set inp logic	0: Low active 1: High active	0	I32

*1: Parameter value detail description

7	6	5	4	3	2	1	Bit : 0
-	-	-	-	1~8 :DO_CH0~ DO_CH7			
15	14	13	12	11	10	9	Bit : 8
-	-	-	-	-	-	-	ON/OFF

27. Sampling parameters table

1. Sampling parameter table

Sampling parameter table				
Para NO.	Define	Description	Parameter data value.	Default
00h	SAMP_PA_RATE	Sampling rate(cycle), (depended on motion cycle time) For 8392, 8253/6 and 8254/58	1~ 65535(times of cycle)	1
		Sampling rate(ms), (depended on OS Timer) For MNET-4XMO	1~5	1
02h	SAMP_PA_EDGE	Edge triggered	0:Rising edge, 1:faling edge	0
03h	SAMP_PA_LEVEL	Triggered level	(I32) -2147483648 to 2147483647	0
05h	SAMP_PA_TRIGCH	Trigger channel	0 ~ 3 (Ch0~Ch3)	0
10h	SAMP_PA_SRC_CH 0	Sampling source of Channel 0	Refer to sampling source table (*1)	0
11h	SAMP_PA_SRC_CH 1	Sampling source of Channel 1	Refer to sampling source table (*1)	0
12h	SAMP_PA_SRC_CH 2	Sampling source of Channel 2	Refer to sampling source table (*1)	0
13h	SAMP_PA_SRC_CH 3	Sampling source of Channel 3	Refer to sampling source table (*1)	0

(*1), This parameter must also involve the information of axis id. Four bytes data is needed for this parameter. The first two bytes is the information of referred id including axis id / channel id / Vao id, and the low two bytes is the type of sampling source.

2. Sampling source table

sampling source table					
Source	Symbol Define	Description	Value range	type	Referred id
0x00	SAMP_COM_POS	command position	I32 value	I32	Axis id
0x01	SAMP_FBK_POS	feedback position	I32 value	I32	Axis id
0x02	SAMP_CMD_VEL	command velocity	I32 value	I32	Axis id

0x03	SAMP_FBK_VEL	feedback velocity	I32 value	I32	Axis id
0x04	SAMP_MIO	motion IO	I32 value (bit format)	I32	Axis id
0x05	SAMP_MSTS	motion status	I32 value (bit format)	I32	Axis id
0x06	SAMP_MSTS_ACC	motion status acc	0: Not at acceleration 1: At acceleration	I32	Axis id
0x07	SAMP_MSTS_MV	motion status at max velocity	0: Not at max. velocity 1: At max. velocity	I32	Axis id
0x08	SAMP_MSTS_DEC	motion status at dec	0: Not at deceleration 1: At deceleration	I32	Axis id
0x09	SAMP_MSTS_CSTP	motion status CSTP	0: CSTP status ON 1: CSTP status OFF	I32	Axis id
0x0A	SAMP_MSTS_MDN	motion status MDN	0: NSTP status ON 1: NSTP status OFF	I32	Axis id
0x0B	SAMP_MIO_INP	motion status INP	0: INP status ON 1: INP status OFF	I32	Axis id
0x0D	SAMP_MIO_ORG	motion status OGR	0: OGR status ON 1: OGR status OFF	I32	Axis id
0x20	SAMP_CONTROL_VOL	Control command voltage	I32 value	I32	Axis id
0x21	SAMP_GTY_DEVIATION	Gantry deviation	I32 value	I32	Axis id
0x22	SAMP_ENCODER_RAW	Encoder raw data	I32 value	I32	Axis id

0x23	SAMP_ERROR_POS	Error position	I32 value	I32	Axis id
0x24	SAMP_PTBUFF_RUN_INDEX	Point table running index	I32 value	I32	Table id 0~1
0x10	SAMP_COM_POS_F64	Command position	F64 value	F64	Axis id
0x11	SAMP_FBK_POS_F64	Feedback position	F64 value	F64	Axis id
0x12	SAMP_CMD_VEL_F64	Command velocity	F64 value	F64	Axis id
0x13	SAMP_FBK_VEL_F64	Feedback velocity	F64 value	F64	Axis id
0x14	SAMP_CONTROL_VOL_F64	Control command voltage	F64 value	F64	Axis id
0x15	SAMP_ERR_POS_F64	Error position	F64 value	F64	Axis id
0x18	SAMP_PWM_FREQUENCY_F64	PWM frequency (Hz)	F64 value	F64	PWM CH id
0x19	SAMP_PWM_DUTY_CYCLE_F64	PWM duty cycle (%)	F64 value	F64	PWM CH id
0x1A	SAMP_PWM_WIDTH_F64	PWM width (ns)	F64 value	F64	PWM CH id
0x1B	SAMP_VAO_COMP_VEL_F64	Composed velocity for Laser power control (pps)	F64 value	F64	VAO id
0x1C	SAMP_PTBUFF_COMP_VEL_F64	Composed velocity of point table	F64 value	F64	Table id 0~1
0x1D	SAMP_PTBUFF_COMP_ACC_F64	Composed acceleration of point table	F64 value	F64	Table id 0~1

28. Motion IO status and motion status definitions

1. Motion IO status table

PCI-8254/58 motion IO status table								
Bit No	7	6	5	4	3	2	1	0
	SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM
Bit No	15	14	13	12	11	10	9	8
				SMEL	SPEL	SCL		

Motion IO status description table

Motion IO status description table		
Bit	Define	Description
0	ALM	Servo alarm
1	PEL	Positive end limit
2	MEL	Negative end limit
3	ORG	Original position sensor(home sensor)
4	EMG	EMG sensor
5	EZ	EZ passed
6	INP	In position
7	SVON	Servo ON
8~9	Reserved	Reserved, always be 0
10	SCL	Software circular limit
11	SPEL	Software positive end limit
12	SMEL	Software negative end limit
13~	Reserved	Reserved, always be 0

2. Motion status table

Motion status definition table								
BitNo	7	6	5	4	3	2	1	0
	--	HMV	MDN	DIR	DEC	ACC	VM	CSTP
BitNo	15	14	13	12	11	10	9	8

	JOG	--	--	--	PTB	WAIT	--	--
BitNo	23	22	21	20	19	18	17	16
	--	--	--	--	POSTD	PRED	BLD	ASTP
BitNo	31	30	29	28	27	26	25	24
	--	BACKLASH	--	GER	--	--	--	--

Motion Status Description Table

Bit	Define	Description
0	CSTP	Command stopped (But it could be in motion)
1	VM	In maximum velocity
2	ACC	In acceleration
3	DEC	In deceleration
4	DIR	Move direction. 1:Positive direction, 0:Negative direction
5	MDN	Motion done. 0: In motion, 1: Motion done (It could be abnormal stop)
6	HMV	In homing
...	Reserved	Reserved, always be 0
10	WAIT	Axis is in waiting state. (Wait move trigger)
11	PTB	Axis is in point buffer moving. (When this bit on, MDN and ASTP will be cleared)
...	Reserved	Reserved, always be 0
15	JOG	In jogging
16	ASTP	0: Stop normally, 1: abnormal stop, When axis in motion, this bit will be clear.
17	BLD	Axis (Axes) in blending moving
18	PRED	Pre-distance event, 1: event arrived. The event will be clear when axis start moving
19	POSTD	Post-distance event. 1: event arrived. The event will be clear when axis start moving
...	Reserved	Reserved, always be 0
28	GER	1: In geared (This axis as slave axis and it follow a master specified in axis parameter.)
29	Reserved	Reserved, always be 0
30	BACKLASH	0: In operation; 1: IDLE
31	Reserved	Reserved, always be 0

29. Interrupt factor table

Interrupt Item Definition Table

PCI-8254/58 Interrupt Item Definition Table	
Item No.	Description
0~7	Axes interrupts For PCI-8254, Item is from 0 to 3. (4~7 is reserved.) For PCI-8254, Item is from 0 to 7.
8	System interrupts
9	DI – Rising edge interrupts
10	DI- Falling edge interrupts

PCI-8254 Axes interrupt factors definition of Item 0~3

PCI-8258 Axes interrupt factors definition of Item 0~7

PCI-8254/58 Axes interrupt factors description table			
Factor No.	Define	Interrupt condition description	Note
0	IALM	Servo alarm signal turn ON	
1	IPEL	Positive end limit switch turn ON	
2	IMEL	Minus end limit switch turn ON	
3	IORG	Home switch turn ON	
4	IEZ	EZ passed signal turn ON	
5	IINP	In position	
6	IEMG	EMG signal turn ON	
7	Reserved	Reserved, always be 0	
8	ICSTP	Command stop	(*2)
9	IVM	In maximum velocity	
10	IACC	In acceleration	
11	IDEC	In deceleration	
12	IMDN	Motion done	(*1)
13	IASTP	Abnormal stop	
14	Reserved	Reserved, always be 0	
15	ISPEL	In positive soft limit	
16	ISMEL	In negative soft limit	
17	ISCL	In soft circular limit	
18	IPTB1	P(V)T buffer 1/4 empty (free size > 250)	

19	IPTB2	P(V)T buffer 1/2 empty (free size > 500)	
20	IPTB3	P(V)T buffer empty (free size = 1000)	
21~	Reserved	Reserved, always be 0	

(*1)IMDN: All motion action including home move which can be waited motion done by this interrupt factor.

Users can set normal stop (motion done) condition by set axis parameter function.

(*2)ICSTP: Motion command is stopped, but the axis could be still in motion.

System interrupt factors definition of Item 8

PCI-8254/58 System interrupt factors description table			
Factor No.	Define	Interrupt condition description	Note
0	IEMG	Hardware emergency stop	
1	ILCF0	Hardware linear comparator 0 finish event	
2	ILCF1	Hardware linear comparator 1 finish event	
3	IFCF0	hardware FIFO comparator 0 finish event	
4	IFCF1	hardware FIFO comparator 1 finish event	
5	Reserved	Reserved	

DI-Rising edge interrupt factors definition of Item 9

DI interrupt factors description table			
Factor No.	Define	Interrupt condition description	Note
0 ~ 7	DI0~DI7	DI0 ~ 7 rising edge event	
8 ~ 23	TTL0~TTL15	TTL0 ~ 15 rising edge event	
24	Reserved	Reserved	

DI-Falling edge interrupt factors definition of Item 10

DI interrupt factors description table			
Factor No.	Define	Interrupt condition description	Note
0 ~ 7	DI0~DI7	DI0 ~ 7 falling edge event	
8 ~ 23	TTL0~TTL15	TTL0 ~ 15 falling edge event	
24	Reserved	Reserved	

30. Trigger parameter table

Trigger parameter table				
NO	Define	Description	Value	Default
0x00	TGR_LCMP0_SRC	Linear compare 0 (LCMP0) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x01	TGR_LCMP1_SRC	Linear compare 1 (LCMP1) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x02	TGR_TCMP0_SRC	Table compare 0 (TCMP0) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x03	TGR_TCMP1_SRC	Table compare 1 (TCMP1) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x04	TGR_TCMP0_DIR	Table compare 0 (TCMP0) direction	0: Negative direction 1: Positive direction 2: Bi-direction(No direction)	0
0x05	TGR_TCMP1_DIR	Table compare 1 (TCMP1) direction	0: Negative direction 1: Positive direction 2: Bi-direction(No direction)	0
0x06	TGR_TRG_EN	TRG 0 ~ 3 enable by bit NOTE: This parameter is also controlled by board parameter “PWMx map DO” and “VAO table” functions.	Bit x: (0: disable, 1: enable) Bit 0:TRG0 enable Bit 1:TRG1 enable Bit 2:TRG2 enable Bit 3:TRG3 enable	0
0x10	TGR_TRG0_SRC	Trigger output 0 (TRG0) source Note: OR multi-sources, then output to TRG0)	Bit x:(1: On, 0: Off) Bit 0:Manual0 Bit 1:Reserved Bit 2:FCMP0 Bit 3:FCMP1 Bit 4:LCMP0 Bit 5:LCMP1 Bit 6:MCMP	0

			Bit 7:FCMP2 Bit 8:FCMP3 Bit 9 LCMP2 Bit 10 LCMP3	
0x11	TGR_TRG1_SRC	Trigger output 1 (TRG1) source Note: OR multi-sources, then output to TRG0)	Bit x:(1: On, 0: Off) Bit 0: Manual0 Bit 1: Reserved Bit 2: FCMP0 Bit 3: FCMP1 Bit 4: LCMP0 Bit 5: LCMP1 Bit 6: MCMP Bit 7: FCMP2 Bit 8: FCMP3 Bit 9 LCMP2 Bit 10 LCMP3	0
0x12	TGR_TRG2_SRC	Trigger output 2 (TRG2) source Note: OR multi-sources, then output to TRG0)	Bit x:(1: On, 0: Off) Bit 0: Manual0 Bit 1: Reserved Bit 2: FCMP0 Bit 3: FCMP1 Bit 4: LCMP0 Bit 5: LCMP1 Bit 6: MCMP Bit 7: FCMP2 Bit 8: FCMP3 Bit 9 LCMP2 Bit 10 LCMP3	0
0x13	TGR_TRG3_SRC	Trigger output 3 (TRG3) source Note: OR multi-sources, then output to TRG0)	Bit x:(1: On, 0: Off) Bit 0: Manual0 Bit 1: Reserved Bit 2: FCMP0 Bit 3: FCMP1 Bit 4: LCMP0 Bit 5: LCMP1 Bit 6: MCMP Bit 7: FCMP2 Bit 8: FCMP3	0

			Bit 9 LCMP2 Bit 10 LCMP3	
0x14	TGR_TRG0_PWD	TRG0 pulse width	Pulse Width = (N-1)*20ns N = 2 ~ 0xffffffff	11
0x15	TGR_TRG1_PWD	TRG1 pulse width	Pulse Width = (N-1)*20ns N = 2 ~ 0xffffffff	11
0x16	TGR_TRG2_PWD	TRG2 pulse width	Pulse Width = (N-1)*20ns N = 2 ~ 0xffffffff	11
0x17	TGR_TRG3_PWD	TRG3 pulse width	Pulse Width = (N-1)*20ns N = 2 ~ 0xffffffff	11
0x18	TGR_TRG0_LOGIC	TRG 0 logic	0: Not inverse 1:Inverse	0
0x19	TGR_TRG1_LOGIC	TRG 1 logic	0: Not inverse 1:Inverse	0
0x1A	TGR_TRG2_LOGIC	TRG 2 logic	0: Not inverse 1:Inverse	0
0x1B	TGR_TRG3_LOGIC	TRG 3 logic	0: Not inverse 1:Inverse	0
0x1C	TGR_TRG0_TGL	TRG 0 toggle mode	0: Pulse out 1:Toggle out	0
0x1D	TGR_TRG1_TGL	TRG 1 toggle mode	0: Pulse out 1:Toggle out	0
0x1E	TGR_TRG2_TGL	TRG 2 toggle mode	0: Pulse out 1:Toggle out	0
0x1F	TGR_TRG3_TGL	TRG 3 toggle mode	0: Pulse out 1:Toggle out	0
0x20	TIMR_ITV	Timer Interval	Timer Interval = N * 100 ns N = 1~ 53687091	1(100 ns)
0x21	TIMR_DIR	Timer direction	0: Positive count 1: Negative count	0
0x22	TIMR_RING_EN	Enable timer counter to be as Ring counter	0: Disable (0x7fffffff+1 → 0x80000000) (0x80000001-1 → 0x80000000) 1: Enable (0x7fffffff+1 → 0x00000000) (0x00000000-1 → 0x7fffffff)	0
0x23	TIMR_EN	Timer enable	0: Disable, 1:Enable	0

0x30	TGR_MCMP0_SRC	Multi-axis comparator 0 (MCMP0) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x31	TGR_MCMP1_SRC	Multi-axis comparator 1 (MCMP1) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x32	TGR_MCMP2_SRC	Multi-axis comparator 2 (MCMP2) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x33	TGR_MCMP3_SRC	Multi-axis comparator 3 (MCMP3) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x34	TGR_MCMP_MODE	Mode to decide when multi-axis comparator can trigger PWM	0: original mode: when motor reaches boundary defined by window 1: precision mode: when motor is inside boundary defined by window and the position deviation relative to compared point is shortest	0
0x35	TGR_TRG0_TOGGLE_MODE	Enable this mode to allow user to set the status of the trigger out pin. NOTE: This mode will affect other FPGA modules such that Compare trigger.	0: disable 1: enable	0
0x36	TGR_TRG1_TOGGLE_MODE	Same as above	0: disable 1: enable	0
0x37	TGR_TRG2_TOGGLE_MODE	Same as above	0: disable 1: enable	0
0x38	TGR_TRG3_TOGGLE_MODE	Same as above	0: disable 1: enable	0
0x39	TGR_TRG0_TOGGLE_STATUS	Write and read the Toggle output status	0: output low 1: output high	0
0x3A	TGR_TRG1_TOGGLE_STATUS	Same as above	0: output low 1: output high	0
0x3B	TGR_TRG2_TOGGLE_STATUS	Same as above	0: output low 1: output high	0
0x3C	TGR_TRG3_TOGGLE_STATUS	Same as above	0: output low 1: output high	0
0x40	TGR_TCMP2_SRC	Table compare 2 (TCMP2) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x41	TGR_TCMP3_SRC	Table compare 3 (TCMP3) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x42	TGR_TCMP2_DIR	Table compare 2 (TCMP2) direction	0: Negative direction 1: Positive direction 2: Bi-direction(No direction)	0

0x43	TGR_TCMP3_DIR	Table compare 3 (TCMP3) direction	0: Negative direction 1: Positive direction 2: Bi-direction(No direction)	0
0x44	TGR_LCMP2_SRC	Linear compare 2 (LCMP2) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9
0x45	TGR_LCMP3_SRC	Linear compare 3 (LCMP3) source	0 ~ 7: Encoder counter 0~7 8: Timer 0 counter 9: Disable	9

31. Latch parameter table

Trigger parameter table				
NO	Define	Description	Value	Default:
0x10	LTC_IPT	Latch source	Source to trigger the position latch: bit 0~7: <u>TTL0~TTL7</u> Digital input signals; bit 8~11: PWM pulse out	0
0x11	LTC_ENC	Latch encoder	Determine which encoder is latched when latch source is triggered; The range of encoder no. is 0~7	0
0x12	LTC_LOGIC	Latch logic	Support three kinds of logic modes to trigger the latch process: 0: Only RisingEdge 1: Only FallingEdge 2: Both RisingEdge and FallingEdge	0

32. Device information table

PCI-8254/58 Device information					
InfoNo.	Info	Format	InfoNo.	Info	
0x00	Reserved	--	0x01	Reserved	--
0x10	Driver version	Date	0x11	Reserved	--
0x20	Reserved	--	0x21	FPGA version	32 Bits
0x30	PCB version (Carrier)	PCB	0x31	Reserved	--
0x40	DSP version	Date	0x41	Reserved	--

Format description:

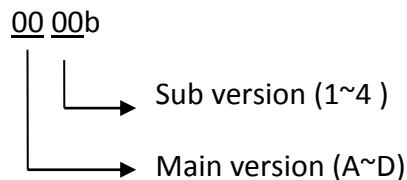
1. Date format: 32 bit value

Value = YYMMDD; Y:year, M:month, D:day

Eg. Driver version = 80212. 2008/2/12 release.

2. PCB format: 4 bits value.

00 00b = PCB A1 version



Dec.	Bin	Version	Dec.	Bin	Version
0	0000b	A1	8	1000b	C1
1	0001b	A2	9	1001b	C2
2	0010b	A3	10	1010b	C3
3	0011b	A4	11	1011b	C4
4	0100b	B1	12	1100b	D1
5	0101b	B2	13	1101b	D2
6	0110b	B3	14	1110b	D3
7	0111b	B4	15	1111b	D4

3. 32 Bits Format: 32 bit value

33. VAO parameter table

PCI-8254/58 VAO parameter table				
NO	Define	Description	Value	Default:
0x00 + (2 * N) Note: N is TableNo, range is 0 ~ 7. (*1)	VAO_TABLE_OU TPUT_TYPE	Table output type (*1)	0: Voltage(reserved) 1: PWM mode 2: PWM frequency mode with fixed width 3. PWM frequency mode with fixed duty cycle	1
0x01 + (2 * N) Note: N is TableNo, range is 0 ~ 7. (*1)	VAO_TABLE_ INPUT_TYPE	Table input type	0: Feedback speed 1: Command speed	0
0x10 + N Note: N is TableNo, range is 0 ~ 7. (*1)	VAO_TABLE_ PWM_Config	Configure PWM according to output type.	a. Mode 0 - Don't care b. Mode 1 - set a fixed frequency (3 ~ 50M Hz) c. Mode 2 - set a fixed Pulse Width (20 ~ 335544300 ns) d. Mode 3 – set a fixed duty cycle: N * 0.05 %. (N: 1 ~ 2000)	100
0x20 + N Note: N is TableNo, range is 0 ~ 7. (*1)	VAO_TABLE_ SRC	Specify axisID for VAO table. (linear speed on multi- axes)	Bit0: Axis 0 On Bit1: Axis 1 On Bit2: Axis 2 On Bit3: Axis 3 On	0x01
0x30	Reserved	Reserved	Reserved	

(*1): Vao supports 8 tables. Each table has own parameter setting. For example, user could use 0x00 to set table output type to table0 and use 0x02 to set output type to table1. For another example, user could use 0x20 to specify axis id for table 0 and use 0x21 to specify axis id for table 1.

APS Functions Return Code

The following table provides a list of possible return value in APS library. If the return value is a negative value, it means there are some errors or warning occurred.

We provide C/C++ standard header file, “ErrorCodeDef.h”, which define all errors return value.

APS Error Code Table

Code	Define	Error descriptions and items to check
0	ERR_NoError	Success, No error
-1	ERR_OsVersion	Operating system version error. The current operating system you used are not supported by this function.
-2	ERR_OpenDriverFailed	Open driver failed. Create driver interface failed. Check device driver is installed correctly. Check devices are installed correctly in your system.
-3	ERR_InsufficientMemory	System memory insufficiently. There is not enough memory in your system.
-4	ERR_DeviceNotInitial	The Device or the card is not be initialized. Check the card ID The device has been closed The device is not be initialized.
-5	ERR_NoDeviceFound	Devices not found Check device driver is installed correctly. Check devices are installed correctly in your system.
-6	ERR_CardIdDuplicate	Card ID duplicated. Check the card ID settings (SW jump) Check the parameter of initial function is correctly.
-7	ERR_DeviceAlreadyIntialed	The devices have already been initialed. 1. Check the close card function is work correctly.
-8	ERR_InterruptNotEnable	Interrupt events not be enabled. 1. Enable the hardware interrupt. 2. Check the interrupt factor is set correctly.
-9	ERR_TimeOut	Function timeout.

-10	ERR_ParametersInvalid	The value of the parameters is incorrect. Check the setting range of parameters. Compare the setting value of parameters with user manual.
-11	ERR_SetEEPROM	Hardware memory write error.
-12	ERR_GetEEPROM	Hardware memory read error.
-13	ERR_FunctionNotAvailable	The function is not available in current stage. The device is not support this function. System is in error state. 1. Check the function library. 2. Check the hardware connection (servo drive connection) 3. Reinitial(Reboot) the system.
-14	ERR_FirmwareError	Firmware process error. 1. Check the firmware version.
-15	ERR_CommandInProcess	The previous command is in process.
-16	ERR_AxisIdDuplicate	Axes' ID is duplicated.
-17	ERR_ModuleNotFound	Slave module not found.
-18	ERR_InsufficientModuleNo	System ModuleNo insufficiently
-19	ERR_HandShakeFailed	HandSake with the DSP out of time.
-20	ERR_FILE_FORMAT	Config file format error.(cannot be parsed)
-21	ERR_ParametersReadOnly	Function parameters read only.
-22	ERR_DistantNotEnough	Distant is not enough for motion.
-23	ERR_FunctionNotEnable	Function not yet enabled
-24	ERR_ServerAlreadyClose	Server already closed
-25	ERR_DllNotFound	Could't find virtual DLL
-32	ERR_DllFuncFailed	Could't find specified function on virtual DLL
-33	ERR_FeederAbnormalStop	Feeder abnormally stop
-40	ERR_DoubleOverflow	Double format parameter is overflow
-1000	ERR_Win32Error	No such event number, or WIN32_API error, contact with ADLINK's FAE staff.

DSP motion kernel error code

Code	Define	Error descriptions and items to check
-2001	MKERR_AXIS_INDEX	Axis range error
-2002	MKERR_CHANNEL_INDEX	Channel range error
-2003	MKERR_PARA_UNDEFINE	Parameter number undefined

-2004	MKERR_PARA_FAULT	Parameter data is wrong
-2005	MKERR_STATE_UNAVAILABLE	This state cannot do this thing. (command position only can be set when axis is in idle)
-2006	MKERR_CHECKSUM	Checksum error (Internal error),(Check code error)
-2007	MKERR_TIME_OUT	Timeout error
-2008	MKERR_MEM_TEST	Memory test error
-2009	MKERR_CTRL_CMD	Unknown command or this state cannot accept this command
-2010	MKERR_AXES_DIMENSION	The dimension of the axes is invalid.
-2011	MKERR_MBUF_FULL	Motion buffer is full
-2012	MKERR_NO_AVAILABLE_SPG	This function cannot be accept when axes are in blending, (all spgs are busy)
-2013	MKERR_BLEND_PERCENT	The transition parameter " percent " is out of range.
-2014	MKERR_TRANSITION_MODE	Transition mode is undefined or not support.
-2015	MKERR_COORD_TRANS_INDEX	Transform matrix column index > MAX_AXES (invalid)
-2016	MKERR_SINP_WIDTH	Soft-inp width invalid (≥ 0)
-2017	MKERR_SINP_STABLE_TIME	Soft-inp stable time invalid (< 65535)
-2018	MKERR_GANTRY_MASTER	Gantry master axis invalid
-2019	MKERR_FCMP_SIZE	The size of compare date is over range.
-2020	MKERR_VS_INVALID	Start velocity invalid, (vs ≥ 0)
-2021	MKERR_VE_INVALID	End velocity invalid. (ve ≥ 0)
-2022	MKERR_VM_INVALID	Maximum velocity invalid. (vm > 0)
-2023	MKERR_ACC_INVALID	Acceleration invalid. (acc > 0)
-2024	MKERR_DEC_INVALID	Deceleration invalid. (dec > 0)
-2025	MKERR_S_INVALID	S invalid. ($0 \leq S \leq 1$)
-2026	MKERR_SD_DEC_INVALID	SD Dec invalid. (>0)
	MKERR_AXES_OVERLAPPING	Axes number cannot overlapping.
	MKERR_BLEND_DISTANCE	The transition parameter "ResidueDistance" cannot < 0.0.
	MKERR_ERROR_POS_LEVEL	Error position check level must ≥ 0.0
-2030	MKERR_WAIT_MOVE	Wait mode not accept when axis in moving
-2031	MKERR_AX_DISABLE	Axis is disable (Servo off)
-2032	MKERR_AX_ERROR	Axis is in error state (AX_ERR_STOPPING/AX_ERR_STOPPED). you should reset the error state.
-2033	MKERR_AX_MOVING	Axis is in moving and command cannot accept (cannot overwrite), axes must same dimension, same axes

-2034	MKERR_PRE_EVENT_DIST	pre-event distance must >= 0
-2035	MKERR_POST_EVENT_DIST	post-event distance must >= 0
-2040	MKERR_ARC_PARA	This function cannot accept half arc or full arc parameter
-2041	MKERR_ARC_FINAL_R	The finalR cannot be negative value.
-2042	MKERR_ARC_NORMAL	Normal vector invalid. or arc parameters invalid.
-2050	MKERR_GANTRY_DEV_PROTECT	Gantry deviation protect value must >= 0
-2051	MKERR_INVALID_IN_GANTRY	This command is not allow in gantry mode.
-2052	MKERR_INVALID_GEAR_MASTER	Gear master is not define
-2053	MKERR_ENGAGE_RATE	
-2054	MKERR_GEAR_RATIO	
-2055	MKERR_GEAR_ENABLE_MODE	
-2056	MKERR_GEAR_LOOP	Cannot be a gear loop.
-2057	MKERR_JOG_OFFSET	The jog offset parameter is wrong. must > 0
-2062	MKERR_DI_GROUP	DI group number is wrong.
-2063	MKERR_DI_CH	DI channel number is wrong.
-2070	MKERR_FILTER_COEFFICIENT	Filter coefficient is wrong
-2071	MKERR_FBK_VEL_COEFFICIENT	
-2080	MKERR_PTBUFF_AXIS_NOT_IDLE	Axis (Axes) are now in moving state, cannot start point buffer
-2081	MKERR_PTBUFF_DIMENSION	Invalid axes dimension setting
-2082	MKERR_PTBUFF_AXIS_IN_USE	Axes number are already in use by another PTB
-2083	MKERR_PTBUFF_NOT_ENABLE	PTBUFF not enable
-2084	MKERR_PTBUFF_FULL	PTBUFF is full.
-2085	MKERR_PTBUFF_CURVE_TYPE	Invalid motion curve type
-2086	MKERR_PTBUFF_DIMENSION_MISS	Miss mach line move dimension and PTBUFF dimension
-2087	MKERR_PTBUFF_CURVE_DIMENSION	Curve type and it's dimension miss match
-2088	MKERR_PTBUFF_ABNORMAL_STOP	Axes abnormal stop, please check axis stop code.
-2089	MKERR_PTBUFF_M_QUEUE_EXCEPTION	Axes motion queue exception.
-2090	MKERR_PTBUFF_AXIS_INDEX_INVALID	Target axes index is over PTBUFF dimension
-2091	MKERR_PTBUFF_ID	Invalid PTBUFF ID. Check the input parameter.
-2092	MKERR_PTBUFF_CTRL_COMMAND	Invalid PTBUFF control command. Check the input parameter.
-2093	MKERR_PTBUFF_AXES_IN_MOTION	PTBUFF detected axes are in motion when start PTBUFF.
-2094	MKERR_PTBUFF_EXT_COMMANDS_NUM	PTBUFF extra-commands must < = 7
-2095	MKERR_PTBUFF_EXT_COMMAND_EMPTY	Execute extra command but command queue is empty.

-2096	MKERR_PTBUFF_EXT_COMMAND_FULL	Push extra command but command queue is full.
-2097	MKERR_PTBUFF_EXT_COMMNAD	Invalid extra command code.
-2098	MKERR_PTBUFF_NOT_STOPPED	Invalid command when point buffer is running.
-2099	MKERR_PTBUFF_DWELL_TIME	Dwell time must >= 0
-2100	MKERR_HS_UNKNOWN_CMD	Handshake data error. command unknown.
-2101	MKERR_HS_SIZE	Handshake data error. size or dimension invalid.
-2102	MKERR_HS_SUB_ERRORS	Sub functions have errors. please check sub return code
-2201	MKERR_DSP_SYSTEM_CONFIG	DSP initialize error. (Call adlink R&D staff)
-2208	MKERR_LOAD_CALIBRATION_DATA	Load calibration data failed.
-2209	MKERR_DPRAM_TEST_FAILED	DPRAM hardware test failed. (Call adlink staff)
-2210	MKERR_FPGA_VERSION	FPGA version outdated. (Call adlink staff)
-2211	MKERR_PSC_TIME_OUT	
-2212	MKERR_PLL_TIME_OUT	
-2213	MKERR_PSC_PARAM_ERR	
-2300	MKERR_MOTION_LOOP_TIMING	The timing of motion loop is out of range.
-2401	MKERR_WRITE_ROM	
-2402	MKERR_READ_ROM	
-2403	MKERR_REF_5V	
-2404	MKERR_SET_AI_OFFSET	
-2405	MKERR_SET_AI_GAIN	
-2406	MKERR_SET_AO_OFFSET	
-2407	MKERR_SET_AO_GAIN	
-2408	MKERR_NO_CALIB	No calibration data in EEPROM. (Not all gain/offset of AO/AI are calibrated)
-2409	MKERR_DATA_SIZE	
-2410	MKERR_BACKDOOR_PWD	Backdoor password wrong.
-2411	MKERR_ROM_PROG_SIZE	(Flash) Byte count of data to much or Offset is over range.
-2500	MKERR_OVER_QUEUE_SIZE	Queue size is less than input array
-2600	MKERR_FRP_STATE	State of frequency response process is invalid
-2601	MKERR_RCP_STATE	State of relay control process is invalid
-2700	MKERR_TASK_NUM	Task number is wrong
-2701	MKERR_UNKNOWN_PROGRAM_REGISTER	Unknow program register.
-2702	MKERR_CANNOT_SET_REG _WHEN_PG_NOT_STOP	Program is running, you cannot issue this command

-2703	MKERR_OVER_STACK_SIZE	Over stack size range
-2800	MKERR_ACCESS_UNDEFINE	Access type is undefined
-2801	MKERR_ACCESS_DENIED	Parameter access is denied since it is in save/load process
-2802	MKERR_ERASE_SECTION	Erase section is failed due to time out
-2803	MKERR_LAST_MARK_INVALID	Load data error; ; last mark in flash is error
-2804	MKERR_CHK_PARITY	Load data error; parity bit is error
-2805	MKERR_OVER_DATA_TYPE	Load data error; over max data type define
-2806	MKERR_OVER_PA_TYPE	Load data error; over max parameter type define
-2807	MKERR_OVER_MAX_BOARD	Load data error; over max board parameter define
-2808	MKERR_OVER_MAX_AXIS	Load data error; over max axis parameter define
-2900	MKERR_WDT1_STATE	Watchdog timer 1 has been enabled
-2901	MKERR_WDT1_PERIOD	Max reset period of watchdog timer 1 is out of range