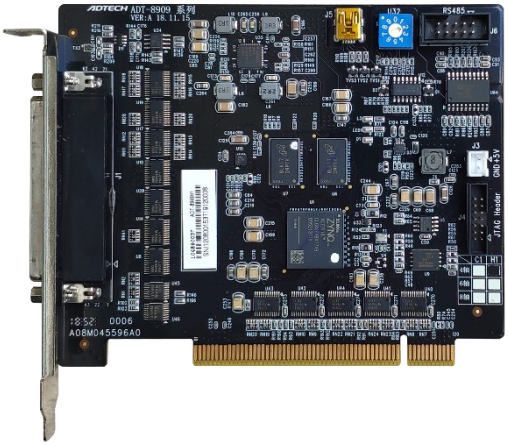


ADT-8909 Series Motion Control Card Programming Manual

- ADT-8949C1/H1
- ADT-8969C1/H1
- ADT-8989C1/H1

Basic Information

This Manual is written by Adtech (Shenzhen) Technology Co., Ltd. This Manual is mainly written by: Ai Xiaoyun.

Copyright

Adtech (Shenzhen) Technology Co., Ltd. (Adtech hereafter) is in possession of the copyright of this manual. Without the permission of Adtech, the imitation, copy, transcription and translation by any organization or individual are prohibited. This manual doesn't contain any assurance, stance or implication in any form. Adtech and the employees are not responsible for any direct or indirect data disclosure, profits loss or cause termination caused by this manual or any information about mentioned products in this manual. In addition, the products and data in this manual are subject to changes without prior notice.

All rights reserved.

Adtech (Shenzhen) Technology Co., Ltd.

Tip	The blue underlined words in this manual, such as adt_initial, are all linkable logos, and the links will give you more detailed information.

Contents

CHAPTER 1 INSTRUCTION LIST 1
CHAPTER 2 ERROR CODE 9
2.1. ERROR CODE 9
2.2. EXAMPLE OF ERROR CODE PARSING 17
CHAPTER 3 SDK CALLING METHOD 20
3.1. VC CALLING METHOD 20
3.1. C# CALLING METHOD 23
CHAPTER 4 BASIC CONTROL 26
4.1. FUNCTION INTRODUCTION 26
4.1.1. Initialize control card 26
4.1.2. Close control card 26
4.1.3. Reset control card 27
4.1.4. Control card soft restart 27
4.1.5. Get index of available control cards 27
4.1.6. Get the number of available solid axes of the control card 28
4.1.7. Get control card version information 28
4.2. ROUTINE 4-1 BASIC CONTROL 29
CHAPTER 5 RESOURCE ALLOCATION 31
5.1. FUNCTION INTRODUCTION 31
5.1.1. Hardware stop signal (EMGN) 31
5.1.2. Hardware limit signal (LMT+/LMT-) 31
5.1.3. Machine home signal (STOP0) 32
5.1.4. Encoder Z-phase signal (STOP1) 32
5.1.5. Pulse mode 34
5.1.6. Programming mode 35
5.1.7. Pulse equivalent 36

5.1.8. Software limit 36
5.2. ROUTINE 5-1 CONFIGURATION METHOD ROUTINES 37
CHAPTER 6 SPEED PLANNING 40
6.1. BASIC SPEED PLANNING 40
6.1.1. Function Introduction 40
6.1.2. Routine 6-1 Basic Speed Planning and Quantitative Driving 41
6.2. SPEED LOOK-AHEAD 44
6.2.1. Function Introduction 44
6.2.2. Routine 6-2 Speed Look-ahead 45
6.3. ARC INTERPOLATION SPEED CONSTRAINT 48
6.3.1. Function Introduction 48
6.3.2. Routine 6-3 Arc Interpolation Speed Constraint 48
6.4. CORNER ACCELERATION SMOOTHING LEVEL 52
6.4.1. Function Introduction 52
6.5. SPEED RATIO 53
6.5.1. Function Introduction 53
6.5.2. Routine 6-4 Speed Ratio 53
6.6. GET CURRENT DRIVE SPEED 55
CHAPTER 7 DRIVE 56
7.1. QUANTITATIVE DRIVE 56
7.1.1. Function Introduction 56
7.1.2. Routine 7-1 Quantitative Drive 57
7.2. CONTINUOUS DRIVE 57
7.2.1. Function Introduction 57
7.2.2. Routine 7-2 Continuous Drive 58
7.3. INTERPOLATION DRIVE 60
7.3.1. Function Introduction 60
7.3.2. Linear interpolation 61
7.3.3. Routine 7-3 Linear Interpolation 61
7.3.4. Plane Arc Interpolation 64
7.3.5. Routine 7-4 Plane Arc Interpolation 64

7.3.6. Spherical Arc Interpolation 66
7.3.7. Cache interpolation 67
7.3.8. Routine 7-5 Cache Interpolation 67
7.3.9. Gantry Double Drive 72
7.3.9.1. Function Introduction 72
7.3.9.2. Routine 7-6 Gantry Double Drive 72
7.3.10. Drive Status Detection, Drive Stop and Stop Information 75
7.3.10.1. Function Introduction 75
7.3.10.2. Routine 7-7 Drive Status and Stop Information 75
CHAPTER 8 HOMING 77
8.1. FUNCTION INTRODUCTION 77
8.2. ROUTINE 8-1 HOMING 80
CHAPTER 9 POSITION CONTROL 82
9.1. FUNCTION INTRODUCTION 82
9.2. ROUTINE 9-1 POSITION CONTROL 83
CHAPTER 10 INPUT/OUTPUT CONTROL 87
10.1. OUTPUT CONTROL 87
10.1.1. Function Introduction 87
10.1.2. Routine 10-1 Output Port Control 87
10.2. INPUT CONTROL 89
10.2.1. Function Introduction 89
10.2.2. Routine 10-2 Input Port Control 89
10.3. INPUT FILTER 91
10.3.1. Function Introduction 91
10.3.2. Routine 10-3 Input Filter Control 91
CHAPTER 11 CACHE EVENT 92
11.1. CACHE OUTPUT CONTROL 92
11.1.1. Function Introduction 92
11.1.2. Routine 11-1 Cache Output Control 92
11.2. CACHE PWM CONTROL 96
11.2.1. Function Introduction 96

11.2.2. Routine 11-2 Cache PWM Control 96
11.3. CACHE DELAY CONTROL 98
11.3.1. Function Introduction 98
CHAPTER 12 POSITION COMPARISON 99
12.1.1. Function Introduction 99
12.1.2. Routine 12-1 Position Comparator 100
CHAPTER 13 POSITION LATCHING 103
13.1.1. Function Introduction 103
13.1.2. Routine 13-1 Position Latch Control 103
CHAPTER 14 DA CONTROL 106
14.1.1. Function Introduction 106
14.1.2. Routine 14-1 DA Control 106
CHAPTER 15 USER CONTROL 107
15.1.1. Function Introduction 107
15.1.2. Routine 15-1 User Control 107
CHAPTER 16 HANDWHEEL CONTROL 109
16.1.1. Function Introduction 109
16.1.2. Routine 16-1 Handwheel Control 109
CHAPTER 17 DETAILS OF STANDARD INSTRUCTIONS 111
17.1. BASIC CONTROL CLASS 111
adt_initial 111
adt_close_card 111
adt_reset_card 112
adt_soft_reboot 112
adt_get_card_index 113
adt_get_total_axis 114
adt_get_motion_err 114
adt_get_communication_err 115
17.2. VERSION INFORMATION CLASS 116
adt_get_lib_ver 116
adt_get_motion_ver 116
adt_get_firmware_ver 117

adt_get_board_ver 117
adt_get_output_alarm 118
17.3. RESOURCE CONFIGURATION CLASS 119
adt_set_emergency_stop 119
adt_get_emergency_stop 119
adt_set_pulse_mode 120
adt_get_pulse_mode 121
adt_set_actual_count_mode 122
adt_get_actual_count_mode 123
adt_set_unit_mode 123
adt_get_unit_mode 124
adt_set_pulse_equiv 124
adt_get_pulse_equiv 125
adt_set_stop0_mode 126
adt_get_stop0_mode 126
adt_set_stop1_mode 128
adt_get_stop1_mode 128
adt_set_limit_mode 129
adt_get_limit_mode 130
adt_set_axis_io_map 131
adt_get_axis_io_map 132
adt_set_softlimit_mode 133
adt_get_softlimit_mode 134
adt_set_pos_variable_loop 135
adt_get_pos_variable_loop 135
adt_get_axis_io_status 136
adt_set_axis_alarm_mode 137
adt_set_limit_lock 137
17.4. SPEED PLANNING CLASS 139
adt_set_startv 139
adt_get_startv 140
adt_set_endv 140

adt_get_endv 141
adt_set_maxv 142
adt_get_maxv 143
adt_set_acc 143
adt_get_acc 144
adt_set_dec 145
adt_get_dec 146
adt_set_admode 146
adt_get_admode 147
adt_set_rate 147
adt_get_rate 148
adt_get_speed 149
adt_set_corner_speed_smooth_level 149
adt_set_arc_speed_clamp_unit 150
adt_set_speed_pretreat_mode 151
17.5. POSITION CONTROL CLASS 152
adt_set_command_pos 152
adt_get_command_pos 152
adt_set_actual_pos 153
adt_get_actual_pos 153
adt_get_target_pos_unit 154
adt_get_target_pos_pulse 154
17.6. DRIVE CONTROL CLASS 156
adt_pmove_unit 156
adt_pmove_pulse 156
adt_continue_move 157
adt_inp_move_unit 157
adt_inp_move_pulse 158
adt_inp_arc2_unit 159
adt_inp_arc3_unit 160
adt_get_axis_status 161
adt_get_all_axis_move_status 162

adt_get_all_axis_reach_status 163
adt_get_stopdata 163
adt_set_axis_stop 164
adt_set_follow_axis 165
adt_get_inp_fifo_len 165
adt_get_inp_index 166
adt_change_pmove_pos_unit 166
17.7. INPUT/OUTPUT CONTROL CLASS 168
adt_write_outport 168
adt_read_outport 168
adt_write_outbit 169
adt_read_outbit 169
adt_read_inport 170
adt_read_inbit 171
adt_set_input_filter 171
17.8. CACHE EVENT CONTROL CLASS 173
adt_set_fifo_outbit 173
adt_set_fifo_pwm 173
adt_set_pwm_output 174
adt_get_pwm_output 175
adt_set_fifo_delay 176
adt_clear_fifo_event 176
adt_get_fifo_event_len 177
adt_get_fifo_event_index 177
adt_set_fifo_multi_io 178
adt_set_fifo_pwm_count 179
17.9. POSITION COMPARISON CONTROL CLASS 181
adt_get_compare_len 181
adt_get_compare_status 181
adt_clear_compare_point 182
adt_set_compare_mode 182
adt_get_compare_mode 183

adt_add_compare_point 184
adt_add_compare_table 185
adt_add_compare_linear 186
adt_set_compare_xd_mode 186
adt_get_compare_xd_mode 187
adt_set_compare_xd_diffout 188
adt_set_compare_xd_area 189
adt_add_compare_xd_point 190
adt_add_compare_2d_linear 191
17.10. POSITION LATCH CONTROL CLASS 193
adt_set_latch_mode 193
adt_get_latch_status 193
adt_get_latch_command_pos 194
adt_get_latch_actual_pos 195
adt_clear_latch 196
adt_close_latch 196
17.11. DA CONTROL CLASS 198
adt_set_daout 198
adt_set_da_adjust 198
17.12. USER CONTROL CLASS 200
adt_set_user_password 200
adt_check_password 200
17.13. HOMING CONTROL CLASS 202
adt_set_home_mode 202
adt_set_home_speed 203
adt_set_home_process 204
adt_get_home_status 205
17.14. HANDWHEEL CONTROL CLASS 206
adt_set_handwheel_move 206
adt_stop_handwheel 206
CHAPTER 18 APPENDIX 208
18.1. ROUTINE INDEX 208

Routine 4-1 Basic Control 208
Routine 5-1 Standard Configuration Method Routines 208
Routine 6-1 Basic Speed Planning and Quantitative Driving..208
Routine 6-2 Speed Look-ahead 208
Routine 6-3 Arc Interpolation Speed Constraint 208
Routine 6-4 Speed Ratio.....208
Routine 7-1 Quantitative Drive 208
Routine 7-2 Continuous Drive 208
Routine 7-3 Linear Interpolation.....208
Routine 7-4 Plane Arc Interpolation 208
Routine 7-5 Cache Interpolation 208
Routine 7-6 Gantry Double Drive.....208
Routine 7-7 Drive Status and Stop Information 208
Routine 8-1 Homing 208
Routine 9-1 Position Control 208
Routine 10-1 Output Port Control 208
Routine 10-2 Input Port Control 208
Routine 10-3 Input Filter Control.....208
Routine 11-1 Cache Output Control.....208
Routine 11-2 Cache PWM Control.....208
Routine 12-1 Position Comparator.....208
Routine 13-1 Position Latch Control 208
Routine 14-1 DA Control 208
Routine 15-1 User Control 208
Routine 16-1 Handwheel Control.....208
18.2. AUXILIARY INTERFACES AND DEFINITIONS.....209
DecodeErrorCode -- Example of Error Code Parsing.....209
DecodeStopData -- Example of Drive Stop Information Parsing 209
DoEvent – Transfer of Control of the Operating System 209
VERIFY_RETURN 209
VERIFY_RETURN_VALUE.....209

VERIFY_RETURN_MSG 209
CONFIRM_DRIVE 210

Chapter 1 Instruction List

Basic control class	
adt_initial	Initialize control card
adt_close_card	Close control card
adt_reset_card	Reset control card
adt_soft_reboot	Control card software restart
adt_get_card_index	Get the index of available control cards of the system
adt_get_total_axis	Get the number of available solid axes of the current control card
adt_get_motion_error	Error in getting motion library of current control card
adt_get_communication_err	Error in getting motion library of current control card

Version information class	
adt_get_lib_ver	Get the LIB library version number of current control card
adt_get_motion_ver	Get the MOTION library version number of current control card
adt_get_firmware_ver	Get the firmware version number of current control card
adt_get_board_ver	Get the terminal board version number of current control card
adt_get_output_alarm	Get the output port overload alarm

Resource configuration class	
adt_set_emergency_stop	Set the hardware stop signal mode of current control card
adt_get_emergency_stop	Get the hardware stop signal mode of current

	control card
adt_set_pulse_mode	Set the current axis pulse mode of the control card
adt_get_pulse_mode	Get the current axis pulse mode of the control card
adt_set_actual_count_mode	Set the current axis encoder count mode of the control card
adt_get_actual_count_mode	Get the current axis encoder count mode of the control card
adt_set_unit_mode	Set the current axis programming mode of the control card
adt_get_unit_mode	Get the current axis programming mode of the control card
adt_set_pulse_equiv	Set the current axis pulse equivalent of the control card
adt_get_pulse_equiv	Get the current axis pulse equivalent of the control card
adt_set_stop0_mode	Set the current axis machine home signal (STOP0) mode of the control card
adt_get_stop0_mode	Get the current axis machine home signal (STOP0) mode of the control card
adt_set_stop1_mode	Set the current axis encoder Z-phase signal (STOP1) mode of the control card
adt_get_stop1_mode	Get the current axis encoder Z-phase signal (STOP1) mode of the control card
adt_set_limit_mode	Set the current axis hardware limit signal mode of the control card
adt_get_limit_mode	Get the current axis hardware limit signal mode of the control card
adt_set_axis_io_map	Custom available axis hardware signal mapping of the control card

adt_get_axis_io_map	Get available axis hardware signal mapping of the control card
adt_set_softlimit_mode	Set the current axis software limit mode of the control card
adt_get_softlimit_mode	Get the current axis software limit mode of the control card
adt_set_pos_variable_loop	Set the current axis logic position variable loop function of the control card
adt_get_pos_variable_loop	Get the current axis logic position variable loop function of the control card
adt_get_axis_io_status	Gets the status of axis binding IO signal
adt_set_axis_alarm_mode	Set axis alarm mode
adt_set_limit_lock	Hardware limit lock enable

Speed planning class	
adt_set_startv	Set the current axis drive start speed of the control card
adt_get_startv	Get the current axis drive start speed of the control card
adt_set_endv	Set the current axis drive end speed of the control card
adt_get_endv	Get the current axis drive end speed of the control card
adt_set_maxv	Set the maximum drive speed of the current axis of the control card
adt_get_maxv	Get the maximum drive speed of the current axis of the control card
adt_set_acc	Set the current axis drive acceleration of the control card
adt_get_acc	Get the current axis drive acceleration of the control card

adt_set_dec	Set the current axis drive deceleration of the control card
adt_get_dec	Get the current axis drive deceleration of the control card
adt_set_admode	Set the current axis drive acceleration/deceleration mode of the control card
adt_get_admode	Get the current axis drive acceleration/deceleration mode of the control card
adt_set_rate	Set the current axis drive speed ratio of the control card
adt_get_rate	Get the current axis drive speed ratio of the control card
adt_get_speed	Get the current axis drive speed of the control card
adt_set_corner_speed_smooth_level	Set the current axis corner acceleration smoothing level of the control card
adt_set_arc_speed_clamp_unit	Set the interpolation axis corner arc speed constraint
adt_set_speed_pretreat_mode	Set the interpolation axis speed look-ahead function mode

Position control class	
adt_set_command_pos	Set the logic position of the specified axis
adt_get_command_pos	Get the logic position of the specified axis
adt_set_actual_pos	Set the actual position of the specified axis
adt_get_actual_pos	Get the actual position of the specified axis
adt_get_target_pos_unit	Get the target position of the specified axis based on the pulse equivalent programming mode

adt_get_target_pos_pulse	Get the target position of the specified axis based on the pulse programming mode

Drive control class	
adt_pmove_unit	Single-axis quantitative drive based on pulse-equivalent programming mode
adt_pmove_pulse	Single-axis quantitative drive based on pulse programming mode
adt_continue_move	Single axis continuous drive
adt_inp_move_unit	Cacheable multi-axis linear interpolation based on pulse equivalent programming mode
adt_inp_move_pulse	Cacheable multi-axis linear interpolation based on pulse programming mode
adt_inp_arc2_unit	Cacheable planar circular interpolation based on pulse equivalent programming mode
adt_inp_arc3_unit	Cacheable spherical circular interpolation based on pulse equivalent programming mode
adt_get_axis_status	Get the drive status of specified axis of the control card
adt_get_all_axis_move_status	
adt_get_all_axis_reach_status	
adt_get_stopdata	Get the drive stop information of specified axis of the control card
adt_set_axis_stop	Stop the drive of the specified axis
adt_set_follow_axis	Set the axis following drive mode (open loop gantry double drive)
adt_get_inp_fifo_len	Get the cache interpolation margin of specified interpolation axis
adt_get_inp_index	Get the index of latest triggered cache

	interpolation instruction
adt_change_pmove_pos_unit	

Input and output control class	
adt_write_outport	Control output port by group
adt_read_outport	Read status of control output port by group
adt_write_outbit	Control a single output port
adt_read_outbit	Read status of a single output port
adt_read_inport	Read status of control input port by group
adt_read_inbit	Read status of a single input port
adt_set_input_filter	Set input filtering function

Cache event control class	
adt_set_fifo_outbit	Cache output control
adt_set_fifo_pwm	Cache high precision PWM output control
adt_set_pwm_output	Output PWM pulse with OUT16, 17 (non-cached, output immediately after setting)
adt_get_pwm_output	Output PWM pulse with OUT16, 17 (non-cached, output immediately after setting)
adt_set_fifo_delay	Cache delay control
adt_clear_fifo_event	Clear cache event
adt_get_fifo_event_len	Get the available cache event margin
adt_get_fifo_event_index	Get the index of latest triggered cache event instruction
adt_set_fifo_multi_io	Get the index of latest triggered cache event instruction
adt_set_fifo_pwm_count	Get the index of latest triggered cache event instruction

Position comparison control class

adt_get_compare_len	Get the one-dimensional position comparator margin
adt_get_compare_status	Get the number of comparison points that have been completed in one-dimensional position
adt_clear_compare_point	Clear the comparison point of one-dimensional position comparator and reset the comparator
adt_set_compare_mode	Set one-dimensional position comparator mode
adt_get_compare_mode	Get one-dimensional position comparator mode
adt_add_compare_point	Add comparison position point of the one-dimensional position comparator
adt_add_compare_table	Add comparison position table of one-dimensional position comparator
adt_add_compare_linear	Add a set of comparison position points of one-dimensional position linearly
adt_set_compare_xd_mode	Set position comparator mode
adt_get_compare_xd_mode	Get position comparator mode
adt_set_compare_xd_diffout	Set position comparator，Pulse differential output parameter
adt_set_compare_xd_area	Set the 2/3D comparison area range
adt_add_compare_xd_ponit	Add comparison position point of the position comparator
adt_add_compare_2d_linear	Add comparison position points of two-dimensional position linearly

Position latch control class	
adt_set_latch_mode	Set single axis position latch mode

adt_get_latch_status	Get single axis position latch status
adt_get_latch_command_pos	Get latch logic position
adt_get_latch_actual_pos	Get latch encoder position
adt_clear_latch	Clear latch data
adt_close_latch	Clear latch data and close Position latch

DA control class	
adt_set_daout	Set DA output function
adt_set_da_adjust	Set the DA correction table

User control class	
adt_set_user_password	Modify/set user password
adt_check_password	Verify user password

Homing control class	
adt_set_home_mode	Homing mode setting
adt_set_home_speed	Homing speed setting
adt_set_home_process	Single axis drive homing
adt_get_home_status	Single axis homing status query

Handwheel Control Class	
adt_set_handwheel_move	Handwheel mode setting
adt_stop_handwhell	Handwheel stop

Chapter 2 Error Code

The error code is the actual feedback of the execution of the library function. Each interface of the library function returns the result after the execution ends. The user needs to judge whether the interface call is successful according to the return value and execute the corresponding control and processing measures.

2.1. Error Code

Error type	Return value	Meaning	Processing method
LIB library error	0	Success	None
	1	Parameter error	Check whether the parameters input by the current instruction are correct
	2	Semaphore creation error	
	3	Arc axis selection error	Check whether the axis setting of the current arc is correct
	4	Arc does not exist	Check the current arc coordinates
	5	Motion library response timeout	The control card firmware does not match the current motion library version. It is recommended to update to the latest firmware
	6	Points on the arc are parallel	Check whether the current arc coordinates are set correctly
	7	PCI return data error	Check whether the control card is plugged in properly
	8	Abnormal communication data amount	The control card firmware does not match the current LIB library version. It is recommended to update to the latest firmware and LIB library

	9	WinIo initialization failed	The control card firmware does not match the current LIB library version. It is recommended to update to the latest firmware and LIB library
	10	PCI bridge has fault	Check whether the control card is plugged in properly
	11	Failed to create a mutex	
	12	Failed to open mutex	
	13	Card number repeats when using multiple cards	Check the card number DIP switch of the control card
	14	No control card is recognized, the card is installed incorrectly or the driver installation fails	Check whether the control card is plugged in properly; Please install the control card properly according to the installation procedure of control card in section 2.4 of 09 Series Motion Control Card User Manual
	15	Communication CRC check failed	The control card firmware does not match the current LIB library version. It is recommended to update to the latest firmware and LIB library
Program update class error	20	PCI driver restart failed, need to shut down and restart	Shut down the computer and restart manually
	21	Program update failed	Turn off the computer and try again; The firmware does not match the

			hardware version of the control card. Please use the correct firmware.
	22	Program update timeout	Turn off the computer and try again; The firmware does not match the hardware version of the control card. Please use the correct firmware.
	23	DSP program verification failed	The firmware does not match the hardware version of the control card. Please use the correct firmware.
	24	Firmware program version does not match dll	Firmware program version and DLL do not match, or firmware exception, cannot identify, need to send factory upgrade, or use the old version DLL
	25	The motion library does not match the DLL	The control card firmware does not match the current DLL library version. It is recommended to update to the latest firmware and DLL library
MOTION library error	101	Parameter error	Check whether the parameters input by the current instruction are correct
	102	Abnormal communication data amount	Check whether the driver is correctly installed
	103	Corresponding axis is in limit stop state	The current axis is in mechanical limit effective state

	104	Motion conflict	Check axis motion status; Multiple motion settings can't be performed simultaneously on the same axis at the same time
	105	Current state does not allow modifying parameters	Confirm the status of the control card or axis
	106	Input data mode error	Check whether the parameters input by the current instruction are correct
	107	Acceleration/deceleration mode setting error	Check whether the parameters input by the current instruction are correct
	108	Calculation error during S-type speed planning	Check whether the parameters input by the current instruction are correct
	109	External emergency stop signal active	External emergency stop input signal is valid.
	110	Bottom motion target position data is abnormal	Check whether the parameters input by the current instruction are correct
	111	Null instruction or invalid instruction	Check whether the parameters input by the current instruction are correct
	112	FPGA reading/writing error	Check whether the parameters input by the current instruction are correct
	113	Interpolation motion instruction cache is full	Issue the interpolation motion instruction when the interpolation motion instruction cache has space
	114	Point motion	Issue the next instruction when the

		instruction cache is full	current point motion is completed
	115	There is a positional deviation when the magnification is restored	The magnification is increased (decreased) too fast. It is recommended to increase (decrease) gradually
	116	Current firmware doesn't support	Current firmware does not support this instruction. It is recommended to update to the latest firmware and DLL library
	117	Axis type error	Check whether the parameters input by the current instruction are correct
	118	Bus communication failure	The control card firmware does not match the current DLL library version. It is recommended to update to the latest firmware and DLL library
	119	Axis alarm	Check axis status
	120	E-mail command communication error	The control card firmware does not match the current DLL library version. It is recommended to update to the latest firmware and DLL library
	121	Data transmission CRC check error	The control card firmware does not match the current DLL library version. It is recommended to update to the latest firmware and DLL library
Homing error	-1	Homing interface parameter 1 error	Check the axis number in homing mode setting
	-2	Homing interface	Check the homing direction and

		parameter 2 error	homing type in homing mode setting
	-3	Homing interface parameter 3 error	Check the home position active level in homing mode setting
	-4	Homing interface parameter 4 error	Check the positive and negative limit setting in homing mode setting
	-5	Homing interface parameter 5 error	Check the servo Z phase setting in homing mode setting
	-6	Homing interface parameter 6 error	Check the reverse search distance in homing mode setting
	-7	Homing interface parameter 7 error	Check the Z phase search distance in homing mode setting
	-8	Homing interface parameter 8 error	Check the home position offset in homing mode setting
	-1000	Homing step 0 is abnormal, failed to issue parameters	Check whether the homing related input parameters are correct
	-1001	Homing step 1 is abnormal, failed to fast search mechanical home signal start	Limit signal and emergency stop signal are valid; Homing speed setting is incorrect
	-1002	Homing step 2 is abnormal, failed to search mechanical home signal	Limit signal and emergency stop signal are valid Mechanical home switch is abnormal
	-1003	Homing step 3 is abnormal, failed to exit mechanical home signal start	Limit signal and emergency stop signal are valid; Mechanical home switch is abnormal
	-1004	Homing step 4 is abnormal, failed to exit	Limit signal and emergency stop signal are valid;

		mechanical home signal	Mechanical home switch is abnormal; Reverse search distance setting is too small
	-1005	Homing step 5 is abnormal, failed to approach mechanical home signal at low speed	Limit signal and emergency stop signal are valid; Homing speed setting is incorrect
	-1006	Homing step 6 is abnormal, failed to search mechanical home signal at low speed	Limit signal and emergency stop signal are valid; Mechanical home switch is abnormal
	-1007	Homing step 7 is abnormal, failed to approach encoder Z phase signal at low speed	Limit signal and emergency stop signal are valid; Homing speed setting is incorrect
	-1008	Homing step 8 is abnormal, failed to search encoder Z phase signal at low speed	Limit signal and emergency stop signal are valid; Z-phase signal is abnormal (check the connection line between the servo Z-phase signal and the control card)
	-1009	Homing step 9 is abnormal, home offset drive failed	Limit signal and emergency stop signal are valid
	-1010	Homing step 10 is abnormal, home offset in-place or position	Limit signal and emergency stop signal are valid

		clearing failed	
	-1020	External termination of homing	Axis stop instruction received
	-1030	Other homing errors	

2.2. Example of Error Code Parsing

The auxiliary operator is designed to parse the abnormal return of the standard operator or the extended operator into a readable string error message, that is, error code parsing.

```
void DecodeErrorCode(char *msg, int len, int retn, const char* itf)
{
    if(NULL==msg || 0>=len)
        return ;
    ZeroMemory(msg, len);
    sprintf(msg, "Operator: %s\nError Code: %d\nError Description:", itf, retn);
    switch(retn)
    {
    case 0:strcat(msg, "execution is successful!");break;
    case 1:strcat(msg, "Parameter error!");break;
    case 2:strcat(msg, "Semaphore creation error!");break;
    case 3:strcat(msg, "Arc axis selection error!");break;
    case 4:strcat(msg, "Arc does not exist!");break;
    case 5:strcat(msg, "Motion library response timeout!");break;
    case 6:strcat(msg, "Points on the arc are parallel!");break;
    case 7:strcat(msg, "PCI return data error!");break;
    case 8:strcat(msg, "Abnormal communication data amount!");break;
    case 9:strcat(msg, "WinIo initialization failed!");break;
    case 10:strcat(msg, "PCI bridge has fault!");break;
    case 11:strcat(msg, "Failed to create a mutex!");break;
    case 12:strcat(msg, "Failed to open mutex!");break;
    case 13:strcat(msg, "Card number repeats when using multiple cards!");break;
    case 14:strcat(msg, "No control card is recognized, the card is installed incorrectly or
the driver installation fails!");break;
    case 15:strcat(msg, "Communication CRC check failed!");break;
    case 20:strcat(msg, "PCI driver restart failed, need to shut down and restart!");break;
    case 21:strcat(msg, "Program update failed!");break;
    case 22:strcat(msg, "Timeout!");break;
```

```
        case 23:strcat(msg, "DSP program verification failed!");break;
        case 24:strcat(msg, " Firmware program version does not match dll, or the firmware is abnormal and can't be recognized, need to be sent to the factory to upgrade, or use old version dll!!");break;
        case 25:strcat(msg, " The motion library does not match the DLL!");break;
        case 101:strcat(msg, "Parameter error!");break;
        case 102:strcat(msg, "Abnormal communication data amount!");break;
        case 103:strcat(msg, "Corresponding axis is in limit stop state!");break;
        case 104:strcat(msg, "Motion conflict!");break;
        case 105:strcat(msg, "Current state does not allow modifying parameters!");break;
        case 106:strcat(msg, "Input data mode error!");break;
        case 107:strcat(msg, "Acceleration/deceleration mode setting error!");break;
        case 108:strcat(msg, "Calculation error during S-type speed planning!");break;
        case 109:strcat(msg, "External emergency stop signal active!");break;
        case 110:strcat(msg, "Bottom motion target position data is abnormal!");break;
        case 111:strcat(msg, "Null instruction or invalid instruction!");break;
        case 112:strcat(msg, "FPGA reading/writing error!");break;
        case 113:strcat(msg, "Interpolation motion instruction cache is full!");break;
        case 114:strcat(msg, "Point motion instruction cache is full!");break;
        case 115:strcat(msg, "There is a positional deviation when the magnification is restored!");break;
        case 116:strcat(msg, "Current firmware doesn't support!");break;
        case 117:strcat(msg, " Axis type error!");break;
        case 118:strcat(msg, "Bus communication failure!");break;
        case 119:strcat(msg, "Axis alarm!");break;
        case 120:strcat(msg, "E-mail command communication error!");break;
        case 121:strcat(msg, "Data transmission CRC check error!");break;
                // Homing error
        case -1:
        case -2:
        case -3:
        case -4:
```

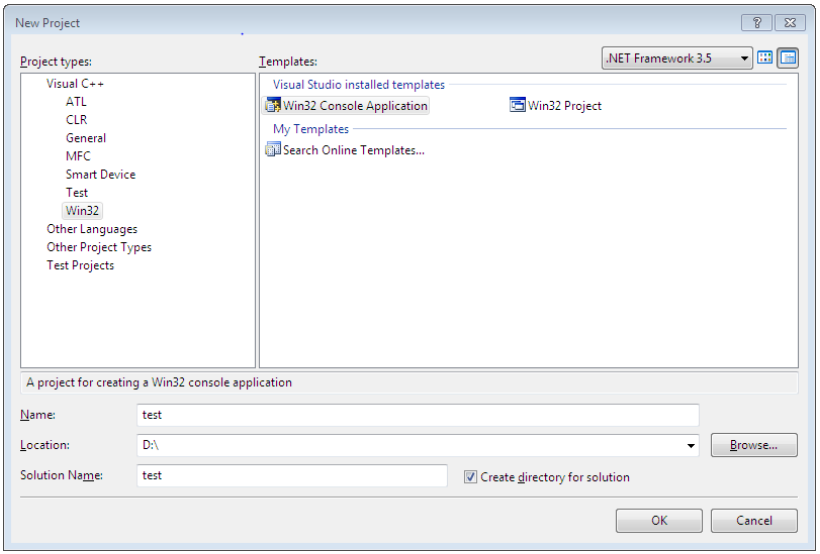
```
        case -5:
        case -6:
        case -7:
        case -8:strcat(msg, "Homing parameter setting error!");break;
        case -1001:
        case -1002:
        case -1003:
        case -1004:
        case -1005:
        case -1006:
        case -1007:
        case -1008:
        case -1009:
        case -1010:strcat(msg, "Homing step is abnormal!");break;
        case -1020:strcat(msg, "Homing is terminated!");break;
        case -1030:strcat(msg, "Other homing errors!");break;
        default:strcat(msg, "Unknown error code!");break;
        }
        strcat(msg, "\n");
}
```

Chapter 3 SDK Calling Method

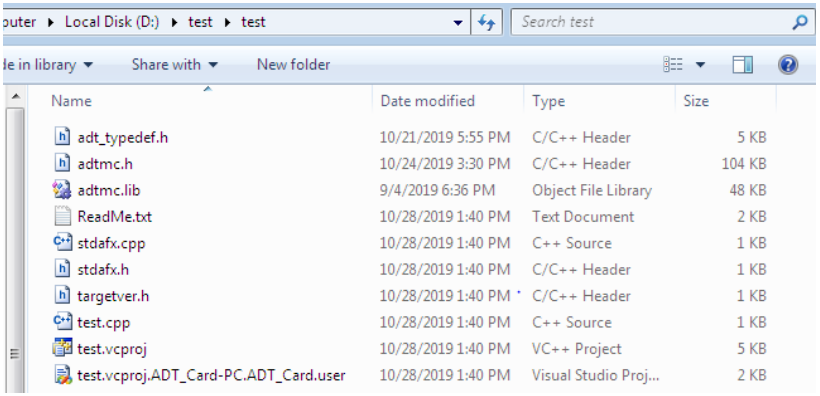
3.1. VC Calling Method

The example of the development platform is VS2008 under the WIN7 x86 operating system. Other platforms under the other operating systems are similar.

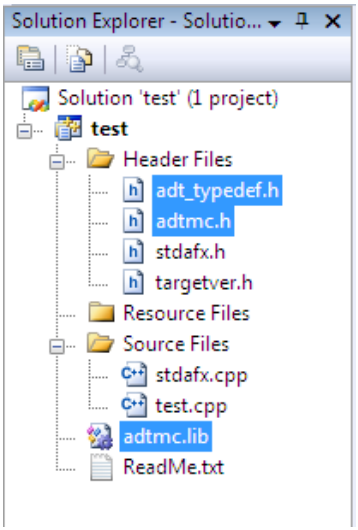
1. File -> New -> Project. Take Win32 console application as an example. Other project types are similar.



2. Copy the adt_typedef.h, adtmc.h, and adtmc.lib files in the development package into the newly created project file path folder. If you develop 32-bit applications under the x64 operating system, copy the files in the x86 development package; if you develop 64-bit applications under the x64 operating system, copy the files in the x64 development package



3. Select your solution, right click -> Add -> Existing Item, and add the development package files copied under the project path to the solution.



4. Add #include "adtmc.h" #include "adt_typedef.h" in the declaration section of the corresponding program source file, header file or global header file "stdafx.h"

```
stdafx.h*  test.cpp  Start Page
(Global Scope)
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#include "targetver.h"

#include <stdio.h>
#include <tchar.h>



// TODO: reference additional headers your program requires here
#include "adtmc.h"
#include "adt_typedef.h"
```

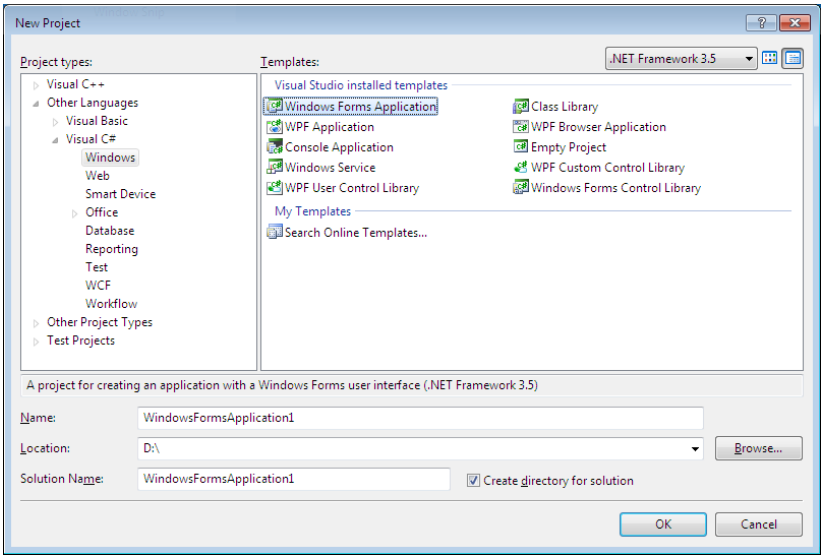
5. Use the control interface provided by the dynamic link library to implement your control program. The following program implements simple initialization basic control, and then drives after the speed parameters are set.

```
stdafx.h  test.cpp*
test.cpp                    d:\test\test\test.cpp
(Global Scope)
// test.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#define VERIFY_RETURN(x)    {int retn = x; if(retn) return retn;}

int _tmain(int argc, _TCHAR* argv[])
{
    int crd_count = 0, index[10] = {0};
    VERIFY_RETURN(adt_initial(&crd_count));//Initialize the control card
    VERIFY_RETURN(adt_get_card_index(&crd_count, index));//Get the number of cards
    VERIFY_RETURN(adt_set_unit_mode(index[0], 1, 0));//Set the current axis programming mode of the control card
    VERIFY_RETURN(adt_set_pulse_equiv(index[0], 1, 1000));//Set the current axis pulse equivalent of the control card
    VERIFY_RETURN(adt_set_startv(index[0], 1, 10));//Set the current axis drive start speed of the control card
    VERIFY_RETURN(adt_set_maxv(index[0], 1, 50));//Set the maximum drive speed of the current axis of the control card
    VERIFY_RETURN(adt_set_acc(index[0], 1, 200));//Set the current axis drive acceleration of the control card
    VERIFY_RETURN(adt_pmove_unit(index[0]), 1, 100, 0);//Single-axis quantitative drive based on pulse-equivalent programming mode
    return 0;
}
```

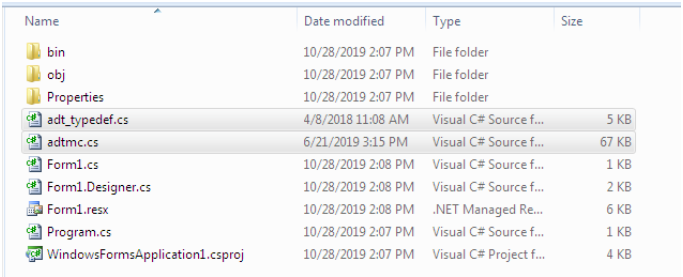
3.1. C# Calling Method

The example of the development platform is VS2008 under the WIN7 x86 operating system. Other platforms under the other operating systems are similar.

1. File -> New -> Project. Take Windows Forms application as an example. Other project types are similar.

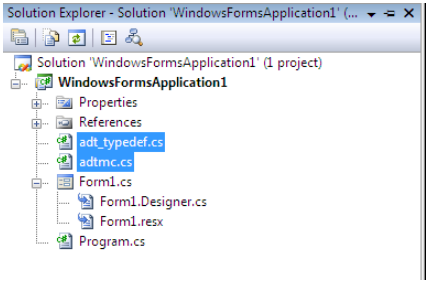


2. We provide users with the standard development kit files adt_typedef.cs and adtmc.cs required for C#. You can copy them into the newly created project file path. We also provide Global.cs aid, which includes some global constant definitions and error code parsing routines to aid user programming. You can have a better programming aid. Global.cs is optional.

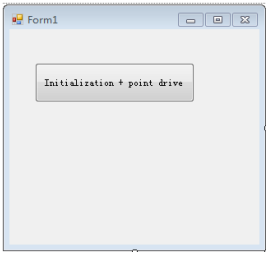


3. Select your solution, right click -> Add -> Existing, and add the development

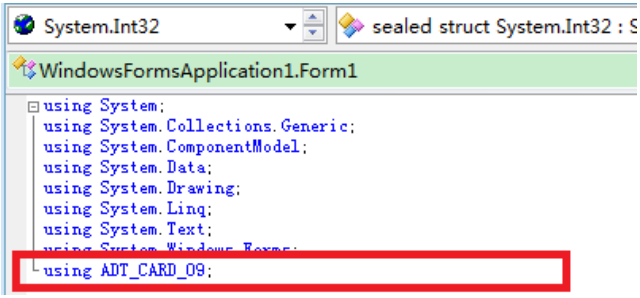
package file to the solution



4. Add a button on the Form



Add a namespace in Form1.cs



The button response event code can be edited as follows to achieve simple point drive

```
private void button1_Click(object sender, EventArgs e)
{
    Int32 crd_count = 0;
    Int32[] crd_index = new Int32[Global.MAX_CARD_COUNT];
    //Initialize
    Int32 retn = adt_card_09.adt_initial(out crd_count, 0);
    if (0 != retn)
    {
        MessageBox.Show(Global.DecodeErrCode(retn));//Error code parsing
        return;
    }
    retn = adt_card_09.adt_get_card_index(out crd_count, crd_index);
    if (0 != retn)
    {
        MessageBox.Show(Global.DecodeErrCode(retn));//Error code parsing
        return;
    }
    //Basic drive
    retn = adt_card_09.adt_set_startv(crd_index[0], 1, 10); //Initial speed setting
    retn += adt_card_09.adt_set_maxv(crd_index[0], 1, 50);//Maximum speed setting
    retn += adt_card_09.adt_set_acc(crd_index[0], 1, 200);//Acceleration setting
    if (0 != retn)
    {
        MessageBox.Show("Speed parameter setting failed!");
        return;
    }
    retn = adt_card_09.adt_pmove_unit(crd_index[0], 1, 100, 0);//Relative drive 100mm
    if (0 != retn)
    {
        MessageBox.Show(Global.DecodeErrCode(retn));//Error code parsing
        return;
    }
}
```

Chapter 4 Basic Control

4.1. Function Introduction

Basic control is the preparation work required by the control card before development and the acquisition of some basic information.

4.1.1. Initialize control card

Control card initialization is the “portal” to the control card function. It only makes sense to call other operators when the motion control card is successfully initialized.

After the control card is successfully initialized, the default resource configuration will be made for the current control card, including but not limited to the following.

1. Assign the default hardware signal port according to the junction box silkscreen
2. Assign the default speed parameter. The acceleration mode is T type by default, and the speed multiplier is 1
3. Use programming mode based pulse equivalent, which is 1000 pulses/mm by default
4. Pulse output mode is PLS+DIR, positive logic pulse, positive logic direction signal
5. Default A/B phase pulse input of the encoder, direction signal positive logic
6. Clear the logic position but do not clear the actual position of the encoder
7. Set input port filter level to 10ms
8. Close all output ports
9. DA output setting 0V

Refer to adt_initial for the control card initialization operator.

4.1.2. Close control card

After the control card is used, the control card resources should be released reasonably.

The resource release operation performed after the control card is closed including but not limited to the following.

1. Reset control card
2. Close all output ports
3. Set DA output voltage to 0V

Refer to adt_close_card for the control card close operator.

4.1.3. Reset control card

Control card reset is often used to clear the hardware emergency stop signal, clear the abnormal stop information, clear the invalid cache data, and clear the motion library stop information.

When the hardware emergency stop function is enabled and the hardware emergency stop signal is triggered, the control card must be reset to clear the hardware emergency stop lock before the axis can be driven normally.

When the system has a motion library exception (adt_get_motion_ver), the control card must be reset to clear the motion library exception information before the system can be driven normally when the problem has been resolved.

Resetting the control card will stop the currently executing drive control and clear the stop information; in case of cache control, it also stops the drive and clears the cache data that haven't been executed.

Resetting the control card does not reset the basic configuration parameters such as the pulse mode and limit mode that the user has executed, nor does it reset the axis speed parameter and the current axis position that have been set successfully.

Refer to adt_reset_card for the control card reset operator.

4.1.4. Control card soft restart

The control card soft restart will first decelerate and stop all the drivers of the current control card and clear the cache, and then reset all data of the control card to the initial power-on state.

The control card soft restart will not power down the control card.

After the soft restart, the application software must be restarted and re-initialized in order to continue to use the control card normally.

Refer to adt_soft_reboot for the control card soft restart operator

4.1.5. Get index of available control cards

The control card index, which is the control card number, is the card number parameter value in the SDK operator.

The 09 series motion control cards all use DIP switch to distinguish the card number index of the multi-card system.

When install the control card, you can arrange the installation sequence at will, just ensure that the DIP switches correspond correctly. The value range of the DIP

switch is 0~9. The DIP value of different control cards can't be repeated.

Refer to adt_get_card_index for the operator of getting index of available control cards.

4.1.6. Get the number of available solid axes of the control card

The 09 series motion control cards provide the operator adt_get_total_axis to get the number of available solid axes of the currently available control card. This operator can be used to distinguish the currently available control card from the N-axis card of the 09 series control cards.

4.1.7. Get control card version information

Version information is a valid identifier for control card upgrade, update, and maintenance.

The version information of the 09 series motion control cards includes LIB library version information, MOTION library version information, firmware version information, and terminal board version information.

The terminal board version information is the alternate version information.

Operator reference

adt_get_lib_ver

adt_get_motion_ver

adt_get_firmware_ver

adt_get_board_ver

Generally, the version information format is "LIB library version information. MOTION library version information. Firmware version information. Terminal board version information". For example

It indicates LIB library version 8020, MOTION library version 4067, firmware version 8010, and terminal board version 0.

4.2. Routine 4-1 Basic Control

```
int main(int argc, char* argv[])
{
    char msg[256] = {0};
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    int retn = adt_initial (&card_count);//Initialize control card
    //Parse error message
    VERIFY_RETURN_MSG (retn, "adt_initial", 1);
    cout<<"Control card initialization completed!";
    //Get available control card index and the number of available axes
    int lib = 0, motion = 0, fmw = 0, board = 0;
    retn =adt_get_card_index (&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    cout<<"Detected a total of"<<card_count<<"09 series motion control cards!\n";
    for(int i=0; i<card_count; ++i)
    {
        retn = adt_get_total_axis(index[i], &axis_count);
        VERIFY_RETURN_MSG(retn, "adt_get_total_axis", 1);
        retn = adt_get_lib_ver(index[i], &lib);
        VERIFY_RETURN_MSG(retn, "adt_get_lib_ver", 1);
        retn = adt_get_motion_ver(index[i], &motion);
        VERIFY_RETURN_MSG(retn, "adt_get_motion_ver", 1);
        retn = adt_get_firmware_ver(index[i], &fmw);
        VERIFY_RETURN_MSG(retn, "adt_get_firmware_ver", 1);
        retn = adt_get_board_ver(index[i], &board);
        VERIFY_RETURN_MSG(retn, "adt_get_board_ver", 1);
        sprintf(msg, "Card Index: %d  Number of available axes: %d  Version
Information: %d.%d.%d.%d\n", index[i], axis_count, lib, motion, fmw, board);
        printf(msg);
        //Reset motion control card
```

```
            retn =adt_reset_card (index[i]);
            VERIFY_RETURN_MSG(retn, "adt_reset_card", 1);
            cout<<"Card Index"<<index[i]<<"has been reset!\n";
        }
        VERIFY_RETURN_MSG(retn, "adt_reset_card", 1);
        //Close motion control card
        retn = adt_close_card();
        VERIFY_RETURN_MSG(retn, "adt_close_card", 1);
        cout<<"09 series motion control card is off!\n";
        system("pause");
        return 0;
}
```

Chapter 5 Resource Allocation

5.1. Function Introduction

The motion control card requires resource allocation before implementing control operations. The so-called resource allocation is to allocate or integrate the software and hardware resources of the control card reasonably according to the process characteristics of the equipment.

The basic resource allocation supported by the 09 series motion control cards includes hardware stop signal configuration, hardware limit mode setting, machine home signal mode setting, encoder Z-phase signal mode setting, pulse mode configuration, encoder working mode setting, programming mode setting, pulse equivalent setting, software limit mode setting, logic position variable loop mode setting, etc.

5.1.1. Hardware stop signal (EMGN)

The hardware stop signal (EMGN) provides the signal detection function for the emergency stop of the equipment. Once the hardware stop signal is triggered, the control card will immediately stop the execution of all current drive and control instructions, clear the cache information, and no longer issue any drive and control instruction until the hardware stop signal restores the state before triggering and the motion control card is reset.

The hardware stop signal can be set to invalid. When it is invalid, its terminal can be used as a general-purpose input.

For the setting interface of the hardware stop signal mode, refer to adt_set_emergency_stop. To get the setting interface of the hardware stop signal mode, refer to adt_get_emergency_stop.

5.1.2. Hardware limit signal (LMT+/LMT-)

The hardware limit signal (LMT+/LMT-) provides the hardware limit security mechanism for each axis of the equipment. When the hardware limit signal in the specified drive direction is triggered, the current axis drive will stop immediately, clear the cache information, no longer accept the drive instruction in the same direction, and reflect in the stop information. For example, if the hardware positive limit is triggered when the X-axis is driven in the positive direction, the X-axis immediately stops and

clears the cache information, and rejects all drive instructions in the positive direction, but the driving in the negative direction is normal. At this moment, the stop information is stopdata&0x1=1; if the hardware negative limit is triggered when the X-axis is driven in the negative direction, the X axis immediately stops and clears the cache information, and rejects all drive instructions in the negative direction, but the driving in the positive direction is normal. At this moment, the stop information is stopdata&0x10=1.

The hardware positive and negative limit signals can be independently set to valid or invalid. When invalid, the corresponding terminal can be used as a general-purpose input.

For the setting interface of the hardware limit signal mode, refer to adt_set_limit_mode. To get the setting interface of the hardware limit signal mode, refer to adt_get_limit_mode.

5.1.3. Machine home signal (STOP0)

The hardware home signal (STOP0) provides a machine home signal for each axis of the equipment. When the drive axis triggers the hardware home signal of the current axis from any direction, the current axis will stop driving in the set stop mode, clear the cache information, no longer accept any drive instruction and show stopdata&0x100=1 in the stop information.

Under normal circumstances, the hardware home signal is not required to be enabled if there is no special multiplexing requirement. The control card disables the signal by default. The hardware home signal will continue to be used during the homing process. The control card will automatically implement its switching during the homing process without user control.

The stop mode of the hardware home signal can be set to stop immediately or decelerate to stop.

The hardware home signal can be set to invalid (default). When invalid, it can be used as a general purpose input.

For the setting interface of the hardware home signal mode, refer to adt_set_stop0_mode. To get the setting interface of the hardware home signal mode, refer to adt_get_stop0_mode.

5.1.4. Encoder Z-phase signal (STOP1)

The encoder Z-phase signal (STOP1) provides the encoder Z-phase signal

connection for each axis of the equipment for more precise encoder homing. When the drive axis triggers the encoder Z phase signal of the current axis from any direction, the current axis will stop driving in the set stop mode, clear the cache information, no longer accept any drive instruction and show stopdata&0x1000=1 in the stop information.

Under normal circumstances, the encoder Z-phase signal is not required to be enabled. The control card disables the signal by default. The encoder Z-phase signal will continue to be used during the homing process. The control card will automatically implement its switching during the homing process without user control.

The stop mode of the encoder Z-phase signal can be set to stop immediately or decelerate to stop.

The encoder Z-phase signal can be set to invalid (default). When invalid, it can be used as a general-purpose input.

For the setting interface of encoder Z-phase signal mode, refer to adt_set_stop1_mode. To get the setting interface of encoder Z-phase signal mode, refer to adt_get_stop1_mode.

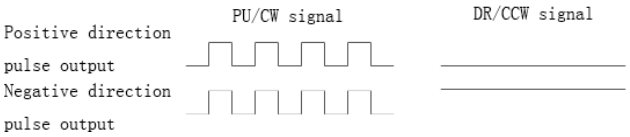
5.1.5. Pulse mode

There are three types of pulse output:

1. Pulse + direction

When the pulse + direction output mode is used, the control card will output drive pulse from the PU/CW, and the output pulse direction will be controlled by DR/CCW.

Its pulse output waveform is as follows

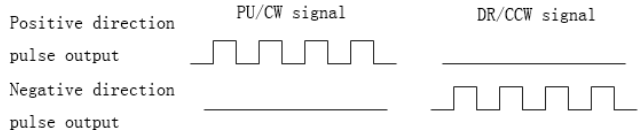


Pulse + direction is one of the pulse output modes commonly used by drivers.

2. CW/CCW

When CW/CCW is used, that is, the double pulse output mode, the control card will output positive direction pulse from PU/CW and output negative direction pulse from DR/CCW.

Its pulse output waveform is as follows



3. A/B phase

A/B phase pulse output refers to counting or encoding by a phase difference between two mutually independent sine waves or square waves.

The setting of the pulse output mode is generally determined by the pulse output parameter setting of the motor or driver.

For the interface of the control card for pulse mode setting, refer to adt_set_pulse_mode.

For the getting interface of pulse mode setting information, refer to adt_get_pulse_mode.

Encoder working mode

The setting of the encoder working mode depends on the actual operating parameters of the peripheral encoder.

For the setting interface of encoder working mode, refer to adt_set_actual_count_mode.

To get the encoder working mode setting interface, refer to adt_get_actual_count_mode.

5.1.6. Programming mode

09 series motion control card supports two axis programming modes: advanced programming mode based on pulse equivalents and standard programming mode based on pulse.

In the different programming modes, the following interface speed or position related parameter data types are different.

adt_set_startv/adt_get_startv

adt_set_endv/adt_get_endv

adt_set_maxv/adt_get_maxv

adt_set_acc/adt_get_acc

adt_set_dec/adt_get_dec

adt_set_softlimit_mode/adt_get_softlimit_mode

In the advanced programming mode based on pulse equivalent, the speed or position related parameter data type is double, and the unit is mm, mm/s and mm/s2. The quantitative drive interface should use adt_pmove_unit, and the linear interpolation drive interface should use adt_inp_move_unit.

In the standard programming mode based on pulse, the speed or position-related parameter data type is double, and the unit is pulse, pulse/s, and pulse/s2. The quantitative drive interface should use adt_pmove_pulse, and the linear interpolation drive interface should use adt_inp_move_pulse.

The axis in advanced programming mode based on pulse equivalent can realize functions such as two-dimensional arc interpolation adt_inp_arc2_unit, three-dimensional arc interpolation adt_inp_arc3_unit, and arc velocity constraint adt_set_arc_speed_clamp_unit.

The axis in standard programming mode based on the pulse does not support functions such as two-dimensional arc interpolation adt_inp_arc2_unit, three-dimensional arc interpolation adt_inp_arc3_unit, and arc velocity constraint adt_set_arc_speed_clamp_unit.

The axis in advanced programming mode based on pulse equivalent needs to set the pulse equivalent of the current axis in units of pulse/mm. By default, the pulse equivalent of the axis is 1000 pulses/mm.

The axis is set to the programming mode based on pulse equivalent by default. The

programming mode parameters successful set by adt_set_unit_mode can be obtained by adt_get_unit_mode.

Axes in different programming modes can't participate in the same set of interpolation drives.

The interpolation axis also needs to set the programming mode in advance before the speed parameters can be set and various interpolation drive interfaces in different programming modes can be called successfully. Changing the programming mode after setting the drive speed parameter may result in scaling of the actual drive speed. The scaling factor is the value of the pulse equivalent. Assigning an interpolated axis for a mismatched programming mode to the interpolating drive interface will cause the drive interface to return an error code 106, i.e., input data mode error.

By default, all axes, including solid and interpolation axes, are programmed based on pulse-equivalent mode.

5.1.7. Pulse equivalent

When the programming mode of the axis is based on the mm unit of pulse equivalent, this parameter determines 1mm=x pulse. By default, 1mm = 1000 pulse. This parameter depends on the pulse subdivision parameter settings of the peripheral. A matching pulse equivalent setting is required to ensure accurate drive position.

For axes in pulse programming mode, setting this parameter has no effect.

By default, refer to adt_set_pulse_equiv for the pulse equivalent setting interface of axes in advanced programming mode based on pulse equivalent. The default is 1000pulse/mm. Pulse equivalent set successfully can be obtained by adt_get_pulse_equiv.

5.1.8. Software limit

09 series motion control cards provide a software limit security mechanism for the equipment in addition to the hardware limit signal security mechanism.

The software limit will set a safe position range for each axis of the equipment. When the axis drive position exceeds the set software limit range, the current axis will stop driving in the set stop mode, clear the cache information, the drive in the same direction of the limit is stopped, and the reverse drive can be executed normally. The stop information is stopdata&0x10000=1 (software positive limit trigger) or stopdata&0x100000=1 (software negative limit trigger).

The positive and negative software limits can be set to be valid or invalid independently. By default, the software limit function is not enabled. For the setting of the software limit function mode, refer to the interface adt_set_softlimit_mode. For setting information acquisition interface, refer to adt_get_softlimit_mode.

Logic variable loop mode

The logic position variable loop function is to re-count from 0 when the logic position of current axis reaches the specified value. This function is mostly used for unidirectional driving occasions such as conveyor belts.

For the logic variable loop mode setting interface, refer to adt_set_pos_variable_loop; to get the logic variable loop mode setting interface, refer to adt_get_pos_variable_loop.

5.2. Routine 5-1 Configuration Method Routines

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    int retn = adt_initial(&card_count);//Initialize control card
    //Parse error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    retn = adt_get_card_index(&card_count, index);//Get available control card index
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    for(int i=0; i<card_count; ++i)
    {
        //Hardware stop function enabled, active low
        retn =adt_set_emergency_stop (index[i], 1, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_emergency_stop", 1);
        //Get the number of available axes of the control card
        retn = adt_get_total_axis(index[i], &axis_count);
        VERIFY_RETURN_MSG(retn, "adt_get_total_axis", 1);
        for(int j=0; j<axis_count; ++j)
        {
            //Set pulse mode, pulse + direction, positive logic pulse, direction output
```

```
signal positive logic
                    retn = adt_set_pulse_mode(index[i], j+1, 1, 0, 0);
                    VERIFY_RETURN_MSG(retn, "adt_set_pulse_mode", 1);
                    //Set encoder working mode, A/B phase pulse input, direction signal
positive logic
                    retn =adt_set_actual_count_mode(index[i], j+1, 0, 0);
                    VERIFY_RETURN_MSG(retn, "adt_set_actual_count_mode", 1);
                    //Set hardware limit mode, positive and negative limit active low
                    retn = adt_set_limit_mode(index[i], j+1, 1, 1, 0);
                    VERIFY_RETURN_MSG(retn, "adt_set_limit_mode", 1);
                    //Setting home signal (STOP0) invalid
                    retn = adt_set_stop0_mode(index[0], j+1, 0, 0, 0);
                    VERIFY_RETURN_MSG(retn, "adt_set_stop0_mode", 1);
                    //Setting encoder Z phase signal (STOP1) invalid
                    retn =adt_set_stop1_mode(index[0], j+1, 0, 0, 0);
                    VERIFY_RETURN_MSG(retn, "adt_set_stop1_mode", 1);
                    //Set axis programming mode to pulse equivalent based
                    retn = adt_set_unit_mode(index[i], j+1, 0);
                    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
                    //Set axis pulse equivalent to pulse/mm
                    retn = adt_set_pulse_equiv(index[i], j+1, 1000);
                    VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
                    //Setting software limit invalid
                    retn = adt_set_softlimit_mode(index[i],j+1,0,9999999,-9999999,0);
                    VERIFY_RETURN_MSG(retn, "adt_set_softlimit_mode", 1);
                    //Setting logic variable loop function invalid
                    retn = adt_set_pos_variable_loop(index[i], j+1, 0, 10000);
                    VERIFY_RETURN_MSG(retn, "adt_set_pos_variable_loop", 1);
            }
        }
        retn = adt_close_card();//Close motion control card
        VERIFY_RETURN_MSG(retn, "adt_close_card", 1);
```

```
    system("pause");
    return 0;
}
```

Chapter 6 Speed Planning

6.1. Basic Speed Planning

6.1.1. Function Introduction

09 series motion control cards support a variety of acceleration and deceleration modes, including T-symmetric/asymmetric acceleration/deceleration, S-symmetric/asymmetric acceleration/deceleration, exponential (EXP) acceleration/deceleration, and trigonometric (COS) acceleration/deceleration.

The basic speed planning involves the following operators:

adt_set_startv/adt_get_startv – Setting and getting the start speed

adt_set_endv/adt_get_endv – Setting and getting the end speed

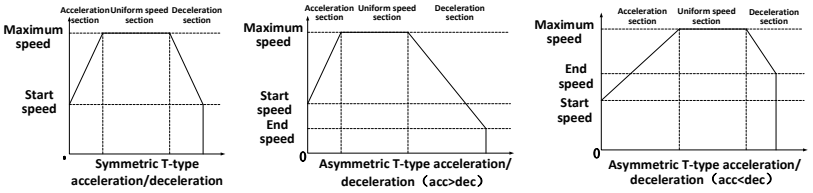
adt_set_maxv/adt_get_maxv – Setting and getting the maximum speed

adt_set_acc/adt_get_acc – Setting and getting acceleration

adt_set_dec/adt_get_dec – Setting and getting deceleration

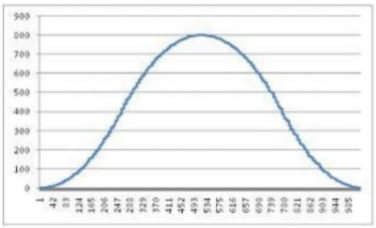
adt_set_admode /adt_get_admode – Setting and getting acceleration/deceleration mode

The influence of the start speed, the end speed, the maximum speed, the acceleration and the deceleration parameters on the speed curve can be understood by referring to the following figures.

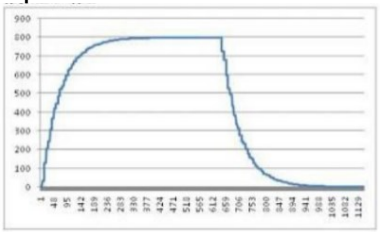


The acceleration/deceleration mode determines the smoothness of the speed curve. In addition to the above-mentioned T-type acceleration/deceleration mode, the 09 series motion control cards also support the following acceleration models.

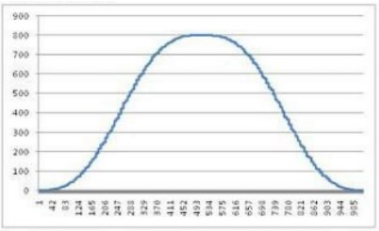
S type



Exponential acceleration/deceleration



Trigonometric function

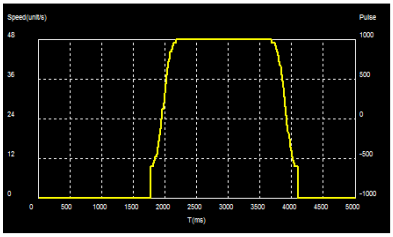


6.1.2. Routine 6-1 Basic Speed Planning and Quantitative Driving

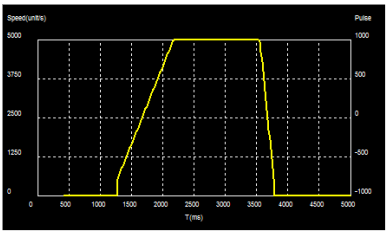
```
int main(int argc, char* argv[])
{
    const int MSG_LENGTH = 100;
    char msg[MSG_LENGTH] = {0};//Error message memory
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse error message
    VERIFY_RETURN_MSG (retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //The first available control card
    //Axis 1 setting based on pulse equivalent programming mode,
```

```
        //Axis 2 setting based on pulse programming mode
        retn = adt_set_unit_mode(index[0], 1, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        retn = adt_set_unit_mode(index[0], 2, 1);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        //Axis 1 pulse equivalent set to mm=1000pulse
        retn = adt_set_pulse_equiv(index[0], 1, 1000);
        VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
        //Axis 1, startv=10mm/s, maxv=50mm/s, acc=100mm/s2, symmetric S-type
acceleration/deceleration
        retn = adt_set_startv(index[0], 1, 10);
        VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
        retn = adt_set_maxv(index[0], 1, 50);
        VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
        retn = adt_set_acc (index[0], 1, 100);
        VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
        retn = adt_set_admode (index[0], 1, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
        //Axis 2, startv=1000pulse/s, endv=500pulse/s
        //maxv=5000pulse/s,acc=10000pulse/s2,dec=20000pulse/s2
        //Asymmetric T-type acceleration/deceleration
        retn = adt_set_startv(index[0], 2, 1000);
        VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
        retn = adt_set_endv(index[0], 2, 500);
        VERIFY_RETURN_MSG(retn, "adt_set_endv", 1);
        retn = adt_set_maxv(index[0], 2, 5000);
        VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
        retn = adt_set_acc(index[0], 2, 10000);
        VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
        retn = adt_set_dec(index[0], 2, 20000);
        VERIFY_RETURN_MSG(retn, "adt_set_dec", 1);
        retn = adt_set_admode(index[0], 2, 1);
```

```
    VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
    //Confirm execution
    CONFIRM_DRIVE(0);
    //Axis 1, relative drive mm
    retn = adt_pmove_unit(index[0], 1, 100, 0);
    VERIFY_RETURN_MSG(retn, "adt_pmove_unit", 1);
    //Axis 2, relative drive pulse
    retn = adt_pmove_pulse(index[0], 2, 50000, 0);
    VERIFY_RETURN_MSG(retn, "adt_pmove_pulse", 1);
    system("pause");
    return 0;
}
```



Axis 1 speed curve



Axis 2 speed curve

6.2. Speed Look-ahead

6.2.1. Function Introduction

09 series motion control card has a large capacity multi-axis cache area, which makes CAM discrete data well restored to the processing model.

The speed look-ahead function makes the multi-segment interpolation drive continuous, and the control card will automatically calculate and optimize the inflection point speed to ensure the smooth transition of the speed while ensuring the accuracy of the machining path.

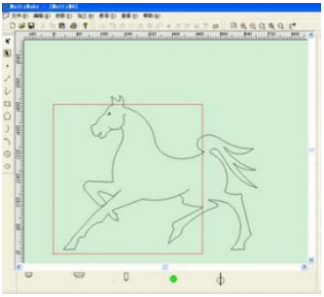
Speed look-ahead and cache interpolation complement each other. Cache interpolation, that is, ultra-large capacity cache, combined with speed look-ahead function, can achieve automatic speed optimization and automatic track adjustment under high-precision high-speed, smooth speed transition, weak mechanical vibration, high-quality machining accuracy, and high-speed and efficient machining process.

The speed look-ahead function is only effective in the execution of the interpolation drive, and the quantitative drive and continuous drive functions are independent of the speed look-ahead.

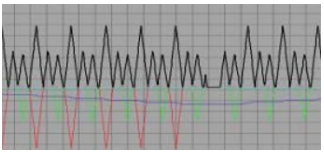
When the speed look-ahead function is enabled, the control card will automatically enable the cache mode for interpolation. For speed look-ahead function interface, refer to adt_set_speed_pretreat_mode.

For the introduction to the interpolation function, please refer to

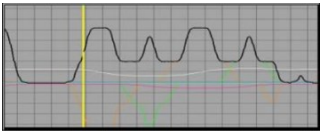
For example, when processing the following trajectory



If the speed look-ahead function is not enabled, that is, when the complex trajectory curve is fitted with multiple small line segments or small arc interpolation, the speed curve will be very sharp, and the processing equipment will also vibrate severely if the cache interpolation function is not used.



If the speed look-ahead function is enabled, that is, use the cache interpolation function to fit multiple small line segments or small arc interpolation, the speed curve will be smooth and the equipment vibration will be greatly reduced.



6.2.2. Routine 6-2 Speed Look-ahead

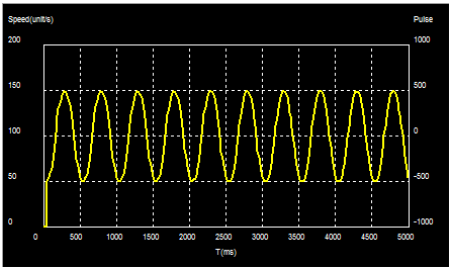
In this case, linear interpolation of 10 small line segments is performed in the standard interpolation mode with speed look-ahead disabled and in the cache interpolation mode with speed look-ahead enabled respecitively.

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //控制卡初始化
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //The first available control card, axis setting based on pulse equivalent programming mode
    retn = adt_set_unit_mode(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
```

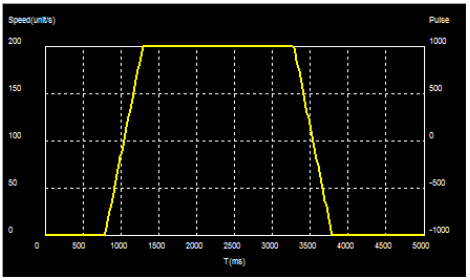
```
        //Corresponding interpolation axis is set to programming mode based on pulse
equivalent
        retn = adt_set_unit_mode(index[0], INPA_AXIS, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        //Axis 1 pulse equivalent set to mm=1000pulse
        retn = adt_set_pulse_equiv(index[0], 1, 1000);
        VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
        //Interpolation axis INPA_AXIS = 63, startv = 50mm/s, maxv = 200mm/s, acc =
400mm/s2
        retn = adt_set_startv(index[0], INPA_AXIS, 50);
        VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
        retn = adt_set_maxv(index[0], INPA_AXIS, 200);
        VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
        retn = adt_set_acc(index[0], INPA_AXIS, 400);
        VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
        retn = adt_set_admode(index[0], INPA_AXIS, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
        //Confirm execution
        int flag = 0;
        printf("Speed parameter has been set. Whether to enable speed look-ahead!\n Please
enter: 0 - No, 1 - Yes, Other - Do not execute the drive and exit\n");
        cin>>flag;
        if(0!=flag && 1!=flag)
                return 0;
        //Enable speed look-ahead
        retn = adt_set_speed_pretreat_mode(index[0], INPA_AXIS, flag);
        VERIFY_RETURN_MSG(retn, "adt_set_speed_pretreat_mode", 1);
        //Single axis linear interpolation of exeution segment
        int AxisList[1] = {1};
        double PosList[1] = {50};
        for(int i=0; i<10; ++i)
        {
```

```
            retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, AxisList, PosList, 0);
            VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
        }
        system("pause");
        return 0;
}
```

When speed look-ahead is disabled and standard interpolation is used, the axis 1 speed curve involved in interpolation is as follows. The speed is limited by the interpolation line segments and can't reach the maximum speed set, resulting in 10 peaks.



When speed look-ahead is enabled and cache interpolation is used, the axis 1 speed curve involved in interpolation is as follows. The speed can reach the set maximum speed under the length of the bus segment, and the smoothness and efficiency are better.



6.3. Arc Interpolation Speed Constraint

6.3.1. Function Introduction

09 series motion control card supports setting the interpolation axis to limit its driving speed according to the arc radius coefficient during arc interpolation.

In the plane arc or space arc interpolation, if the actual radius is smaller than the radius coefficient, the smaller the actual radius is, the less the arc speed is constrained; if the actual radius is larger than the radius coefficient, the larger the actual radius is, the more the arc speed is constrained. When the actual radius is equal to the radius coefficient, the maximum speed of the arc is equal to the speed coefficient.

The relationship among the actual speed constraint Va, the actual arc radius Ra, the speed constraint setting Vs, and the arc radius coefficient setting value Rs is as follows: Va = sqrt(Ra/Rs)*Vs。

Refer to adt_set_arc_speed_clamp_unit for the interface.

Arc interpolation speed constraints can only be used for programming mode based on pulse equivalent.

6.3.2. Routine 6-3 Arc Interpolation Speed Constraint

This case involves a plane arc interpolation function.

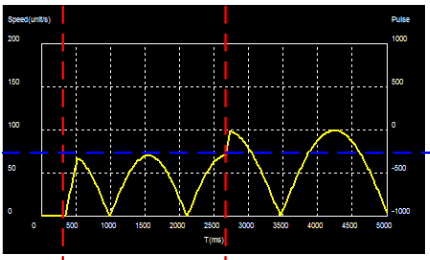
```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //The first available control card
    //Axes 1/2 setting based on pulse equivalent programming mode
    //Arc interpolation drive isn't available for axes not in pulse equivalent programming
```

```
mode
        retn = adt_set_unit_mode(index[0], 1, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        retn = adt_set_unit_mode(index[0], 2, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        //Corresponding interpolation axis is set to programming mode based on pulse
equivalent
        retn = adt_set_unit_mode(index[0], INPA_AXIS, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        //Axis 1/2 pulse equivalent set to mm=1000 pulse
        retn = adt_set_pulse_equiv(index[0], 1, 1000);
        VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
        retn += adt_set_pulse_equiv(index[0], 2, 1000);
        VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
        //Interpolation axis INPA_AXIS = 63, startv = 50mm/s, maxv = 200mm/s, acc =
400mm/s2
        retn = adt_set_startv(index[0], INPA_AXIS, 50);
        VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
        retn = adt_set_maxv(index[0], INPA_AXIS, 200);
        VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
        retn = adt_set_acc(index[0], INPA_AXIS, 400);
        VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
        retn = adt_set_admode(index[0], INPA_AXIS, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
        //Enable speed look-ahead
        retn = adt_set_speed_pretreat_mode(index[0], INPA_AXIS, 1);
        VERIFY_RETURN_MSG(retn, "adt_set_speed_pretreat_mode", 1);
        //Arc interpolation radius coefficient mm, speed constraint coefficient mm/s
        retn = adt_set_arc_speed_clamp_unit (index[0], INPA_AXIS, 50, 100);
        VERIFY_RETURN_MSG(retn, "adt_set_arc_speed_clamp_unit", 1);
        //Confirm execution
        CONFIRM_DRIVE(0);
```

```
    int AxisList[2] = {1, 2};
    double PosList[2] = {0, 0};
    //25 mm radius counterclockwise full arc interpolation
    //Actual speed constraint Va=sqrt(Ra/Rs)*Vs=sqrt(25/50)*100≈.710mm/s
    double ctr_list[2] = {0, 25};
    retn = adt_inp_arc2_unit(index[0], INPA_AXIS, 0, AxisList, PosList, ctr_list,
        0/* counterclockwise*/, 0/* relative position drive mode*/);
    VERIFY_RETURN_MSG(retn, "adt_inp_arc2_unit", 1);
    //50 mm radius counterclockwise full arc interpolation
    //Actual speed constraint Va = sqrt (Ra/Rs) * Vs = sqrt (50/50) * 100 = 100mm/s
    ctr_list[0] = 0, ctr_list[1] = 50;
    retn = adt_inp_arc2_unit(index[0], INPA_AXIS, 0, AxisList, PosList, ctr_list,
        0/* counterclockwise*/, 0/* relative position drive mode*/);
    VERIFY_RETURN_MSG(retn, "adt_inp_arc2_unit", 1);
    //100 mm radius counterclockwise full arc interpolation
    //Actual speed constraint Va=sqrt(Ra/Rs)*Vs=sqrt(100/50)*100=141.421mm/s
    ctr_list[0] = 0, ctr_list[1] = 100;
    retn = adt_inp_arc2_unit(index[0], INPA_AXIS, 0, AxisList, PosList, ctr_list,
        0/* counterclockwise*/, 0/* relative position drive mode*/);
    VERIFY_RETURN_MSG(retn, "adt_inp_arc2_unit", 1);
    system("pause");
    return 0;
}
```

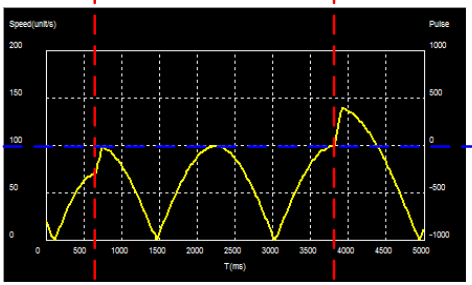
For the axis 1 involved in the interpolation, the maximum actual speed is set to 200 mm/s, the radius coefficient of the arc interpolation speed constraint is 50 mm, and the speed coefficient is 100 mm. After the arc speed is constrained, the speed curve is as shown in the figure below.

25mm radius arc segment

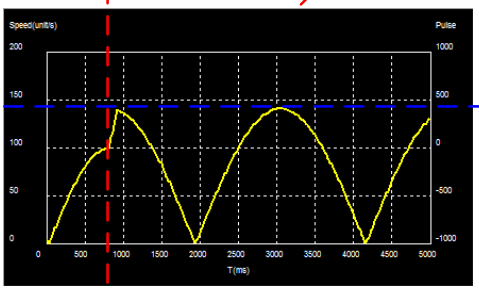


Maximum speed
70.710mm/s

50mm radius arc segment



100mm radius arc segment



Maximum speed
141.421mm/s

6.4. Corner Acceleration Smoothing Level

6.4.1. Function Introduction

The corner acceleration smoothing is used to limit the speed change rate at the corner of the small line segments during the cache interpolation process. During debugging, the smoothing coefficient is gradually increased without affecting the driving precision and efficiency, which is beneficial to reduce the mechanical jitter during interpolation drive.

The acceleration constraint values corresponding to the smoothing coefficients 1 ~ 100 are as follows, in mm/s2.

```
{//1-10
100,258,416, 574, 732, 890, 1048, 1206, 1364,1522,
//11-20
1680, 1838, 1996, 2154, 2312, 2470, 2628, 2786, 2944, 3102,
//21-30
3260, 3418, 3576, 3734, 3892, 4050, 4208, 4366, 4524, 4682,
//31-40
4840, 4998, 5156, 5314, 5472, 5630, 5788, 5946, 6104, 6262,
//41-50
6420, 6578, 6736, 6894, 7052, 7210, 7368, 7526, 7684, 7842,
//51-60
8000,9878,11756,13634,15512,17390,19268,21146,23024,24902,
//61-70
26780, 28658, 30536, 32414, 34292, 36170, 38048, 39926, 41804,43682,
//71-80
45560, 47438, 49316, 51194, 53072, 54950, 56828, 58706, 60584,62462,
//81-90
64340, 66218, 68096, 69974, 71852, 73730, 75608, 77486, 79364,81242,
//91-100
83120,84998,86876,88754,90632, 92510, 94388, 96266, 98144,100000
}
```

Refer to adt_set_corner_speed_smooth_level for the interface. The default smoothing factor is 50.

6.5. Speed Ratio

6.5.1. Function Introduction

09 series motion control card allows setting the speed ratio for axis drive. The axis for setting the ratio can be either the drive axis or the interpolation axis.

Setting the axis ratio immediately refreshes the current speed of the axis, so it is recommended to set the change rate appropriately to prevent speed steps. The ideal way is to generate acceleration and deceleration effects through timed stepwise setting. When the speed ratio is set to 0, the current axis will be in the pause state, and the issued drive data will not be cleared; when the speed ratio restores, the current axis will continue to drive according to the preset track.

Refer to adt_set_rate/adt_get_rate to set and get the speed ratio.

To check whether the current axis is paused when it is in idle state, use adt_get_axis_status to get the state of current axis. State 2 indicates pause.

During interpolation drive, setting the speed ratio of any axis participating in the interpolation is invalid. To set the interpolation speed ratio, set the speed ratio of the interpolation axis.

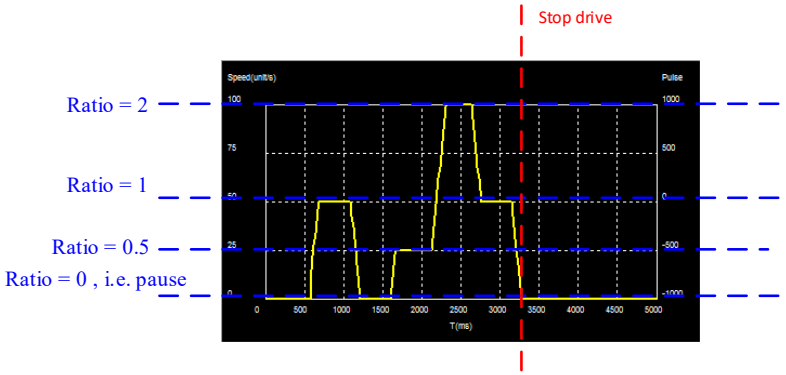
6.5.2. Routine 6-4 Speed Ratio

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //The first available control card
    //Axis 1 setting based on pulse equivalent programming mode
    retn = adt_set_unit_mode(index[0], 1, 0);
```

```
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
//Axis 1 pulse equivalent set to mm=1000pulse
retn = adt_set_pulse_equiv(index[0], 1, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
//轴1,startv=10mm/s,maxv=50mm/s,acc=400mm/s2
//Axis 1, startv=10mm/s, maxv=50mm/s, acc=400mm/s2
retn = adt_set_startv(index[0], 1, 10);
VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
retn = adt_set_maxv(index[0], 1, 50);
VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
retn = adt_set_acc(index[0], 1, 400);
VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
retn = adt_set_admode(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
//Recover ratio
retn = adt_set_rate(index[0], 1, 1);
VERIFY_RETURN_MSG(retn, "adt_set_rate", 1);
//Confirm execution
CONFIRM_DRIVE (0);
//Forward continuous drive
retn = adt_continue_move(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_continue_move", 1);
//Ratio, pause
Sleep(500);
retn = adt_set_rate(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_set_rate", 1);
//Ratio 0.5
Sleep(500);
retn = adt_set_rate(index[0], 1, 0.5);
VERIFY_RETURN_MSG(retn, "adt_set_rate", 1);
//Ratio 2
Sleep(500);
```

```
    retn = adt_set_rate(index[0], 1, 2);
    VERIFY_RETURN_MSG(retn, "adt_set_rate", 1);
    //Ratio 1
    Sleep(500);
    retn = adt_set_rate(index[0], 1, 1);
    VERIFY_RETURN_MSG(retn, "adt_set_rate", 1);
    //Stop
    Sleep(500);
    retn = adt_set_axis_stop(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_axis_stop", 1);
    system("pause");
    return 0;
}
```

The axis 1 speed curve is shown below.



6.6. Get Current Drive Speed

09 series motion control card provides adt_get_speed to get the current driving speed of the axis. The axis to which the driving speed is gotten can be either a solid axis or an interpolation axis. The axis driving speed based on the pulse equivalent programming mode or pulse programming mode can be gotten.

Chapter 7 Drive

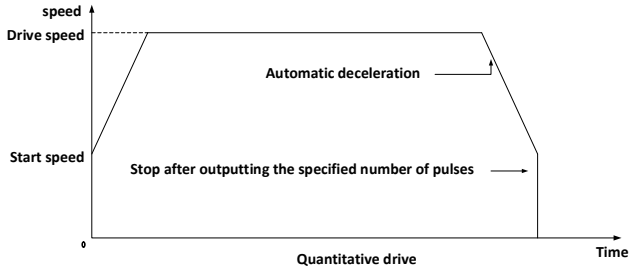
7.1. Quantitative Drive

7.1.1. Function Introduction

Quantitative drive refers to output specified number of pulses at a fixed speed or acceleration/deceleration, and stop accurately after being driven from the current position to the specified position in the set speed mode.

Quantitative drive only focuses on the target position, and has no requirement for the accuracy of the motion trajectory. The driving distance is determined by the number of pulses, and the driving speed is determined by the pulse frequency.

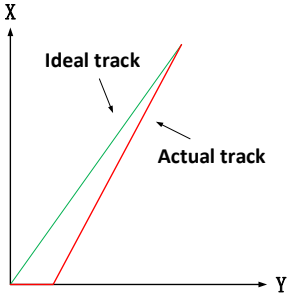
The quantitative driving speed/time curve of the T-type acceleration/deceleration is as shown in the figure below. When the remaining number of output pulses is less than the deceleration cumulative pulses, the drive starts to decelerate. When decelerating to the set initial speed, the axis position will stop at the specified pulse position accurately.



The speed-related parameters must be set reasonably before the quantitative driving. Refer to adt_pmove_unit for the quantitative driving interface based on the pulse equivalent programming mode. Refer to adt_pmove_pulse for the quantitative driving interface based on the pulse programming mode.

Multiple axes in quantitative drive simultaneously are called multi-axis linkage. Multiple quantitative drive instructions are issued, and the instruction cycle interval is only in microseconds. For the machine, it can be considered as simultaneous start. However, if multiple axes are started at the same time and its trajectory must be a spatial straight line, multi-axis linkage is not inadvisable. Instead, the linear interpolation function must be used. Taking the two-axis linkage as an example, the position curve is

a broken line under strict monitoring of time, as shown below.



7.1.2. Routine 7-1 Quantitative Drive

Refer to Routine 6-1 Basic Speed Planning and Quantitative Drive.

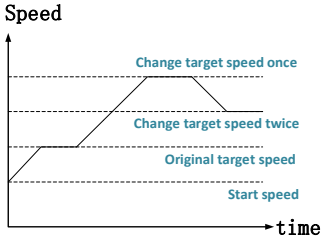
7.2. Continuous Drive

7.2.1. Function Introduction

Continuous drive means that the motor runs continuously from the initial speed to the maximum speed according to the preset parameters until it receives the active stop command or the hardware/software stop signal, and then stops immediately or decelerates to stop.

Continuous drive is often used for speed control applications such as conveyor belt loading and unloading.

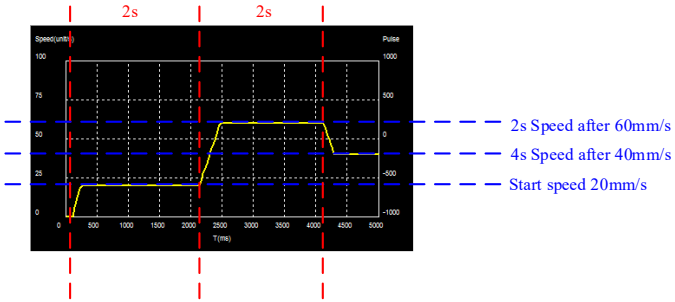
The drive speed related parameters must be set reasonably before continuous drive. Refer to adt_continue_move for the continuous drive interface, and the target speed (adt_set_maxv) can be changed in real time during the drive process. Taking T-type acceleration/deceleration as an example, the case of changing the target speed curve in real time during continuous drive is as follows.



7.2.2. Routine 7-2 Continuous Drive

```
int main(int argc, char* argv[])
{
    const int MSG_LENGTH = 100;
    char msg[MSG_LENGTH] = {0};//Error message memory
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //The first available control card
    //Axis 1 setting based on pulse equivalent programming mode
    retn = adt_set_unit_mode(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
    //Axis 1 pulse equivalent set to mm=1000pulse
    retn = adt_set_pulse_equiv(index[0], 1, 1000);
    VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
    //轴1,startv=5mm/s,maxv=20mm/s,acc=120mm/s2
    //Axis 1, startv=5mm/s, maxv=20mm/s,acc=120mm/s2
    retn = adt_set_startv(index[0], 1, 5);
    VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
    retn = adt_set_maxv(index[0], 1, 20);
    VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
    retn = adt_set_acc(index[0], 1, 120);
    VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
    retn = adt_set_admode(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
    //Confirm execution
```

```
    CONFIRM_DRIVE(0);
    //Forward continuous drive axis
    retn = adt_continue_move(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_continue_move", 1);
    //Change speed after2s
    Sleep(2000);
    retn += adt_set_maxv(index[0], 1, 60);
    VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
    //Change speed again after2s
    Sleep(2000);
    retn += adt_set_maxv(index[0], 1, 40);
    VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
    system("pause");
    return 0;
}
```

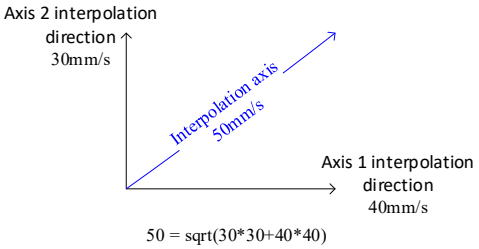


7.3. Interpolation Drive

7.3.1. Function Introduction

Interpolation drive is widely used in various CNC machine tools such as dispensing, cutting, and laser. The interpolation drive realizes multi-axis coordinated drive and densifies the trajectory data to process desired contour.

The interpolation axis is the virtual drive axis of the 09 series motion control card during the interpolation process. The driving direction and speed are the direction and velocity vector sum of the spatial motion coordinate system composed of the N solid axes participating in the interpolation. For example, a simple planar linear interpolation is shown in the following figure. The vector velocity and direction satisfy the following operational relationship. For spatial linear or more complex interpolation, the operation of the velocity and direction of the interpolation axis is geometrically analogous.

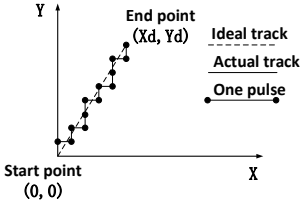


09 series motion control card provides two sets of interpolation axes, INPA_AXIS and INPB_AXIS, which enable simultaneous two independent interpolation drives.

Common types of interpolation drive are: linear interpolation, plane arc interpolation, spherical arc interpolation, spiral interpolation, and spline interpolation.

7.3.2. Linear interpolation

The so-called linear interpolation is an interpolation method that the theoretical contour is a straight line or small line segments can be fitted into a straight line. First assume that a small segment (one pulse equivalent) goes along the x direction from the start point of the actual contour, and the end point is found below the actual contour. Then the next line segment goes a short distance in the y direction. If the end point of the line segment is still below the actual contour, then continue to go a short distance in the y direction until the actual contour is above; then go a small segment in the x direction, and cycle through until the end of the contour is reached. The actual contour is made up of small line segments and polylines within the allowable range of precision.



09 series motion control card supports 2-6 axes linear interpolation. For the linear interpolation interface based on pulse equivalent programming mode, refer to adt_inp_move_unit. For the linear interpolation interface based on pulse programming mode, refer to adt_inp_move_pulse.

For interpolation in different programming modes, be sure to configure the same programming mode for the corresponding interpolation axis before the speed configuration and interpolation drive instructions.

7.3.3. Routine 7-3 Linear Interpolation

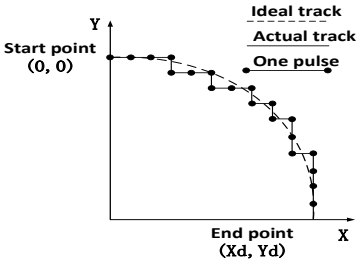
```
int main(int argc, char* argv[])
{
    const int MSG_LENGTH = 100;
    char msg[MSG_LENGTH] = {0};//Error message memory
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
```

```
int retn = adt_initial(&card_count);
//Parse and print error message
VERIFY_RETURN_MSG(retn, "adt_initial", 1);
//Get available control card index
retn = adt_get_card_index(&card_count, index);
VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
//The first available control card
//Axes 1/2/3 setting based on pulse equivalent programming mode
//Axis 4 setting based on pulse programming mode
retn = adt_set_unit_mode(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
retn = adt_set_unit_mode(index[0], 2, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
retn = adt_set_unit_mode(index[0], 3, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
retn = adt_set_unit_mode(index[0], 4, 1);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
//Corresponding to interpolation axis programming mode settings
retn = adt_set_unit_mode(index[0], INPA_AXIS, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
retn = adt_set_unit_mode(index[0], INPB_AXIS, 1);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
//Axis 1/2/3 pulse equivalent set to mm=1000pulse
retn = adt_set_pulse_equiv(index[0], 1, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
retn = adt_set_pulse_equiv(index[0], 2, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
retn = adt_set_pulse_equiv(index[0], 3, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
//Axis INPA_AXIS, startv=5mm/s, maxv=20mm/s, acc=120mm/s2
retn = adt_set_startv(index[0], INPA_AXIS, 5);
VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
```

```
    retn = adt_set_maxv(index[0], INPA_AXIS, 20);
    VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
    retn = adt_set_acc(index[0], INPA_AXIS, 120);
    VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
    retn = adt_set_admode(index[0], INPA_AXIS, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
    //Axis INPB_AXIS, startv=500pulse/s, maxv=2000pulse/s, acc=12000pulse/s2
    retn = adt_set_startv(index[0], INPB_AXIS, 500);
    VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
    retn = adt_set_maxv(index[0], INPB_AXIS, 2000);
    VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
    retn = adt_set_acc(index[0], INPB_AXIS, 12000);
    VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
    retn = adt_set_admode(index[0], INPB_AXIS, 1);
    VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
    //Confirm execution
    CONFIRM_DRIVE(0);
    //09 series motion control card supports group interpolation and independent drive
    int AxisList1[3] = {1,2,3}, AxisList2[1] = {4};
    double PosList1[3] = {100,100,100};
    long PosList2[1] = {10000};
    //Axis 1/2/3 linear interpolation
    retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 3, AxisList1, PosList1, 0);
    VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
    //Axis 4 linear interpolation
    retn = adt_inp_move_pulse(index[0], INPB_AXIS, 0, 1, AxisList2, PosList2, 0);
    VERIFY_RETURN_MSG(retn, "adt_inp_move_pulse", 1);
    system("pause");
    return 0;
}
```

7.3.4. Plane Arc Interpolation

Two-dimensional arc interpolation calculates the point group that approximates the actual arc according to the interpolation digital information between the two ends, and controls the tool to move along these points to process the arc curve. The basic principle is to jointly determine a point with axes X and Y in linear motion, and draw a circle by controlling the coordinates of Y with X in linear motion.



For two-dimensional arc interpolation interface of the 09 series motion control card, refer to adt_inp_arc2_unit.

The plane arc interpolation function is an advanced interpolation function in the programming mode based on pulse equivalent, and the axis based on pulse programming mode can't participate in the plane arc interpolation.

7.3.5. Routine 7-4 Plane Arc Interpolation

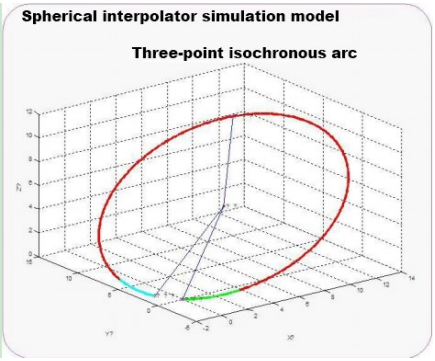
```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //The first available control card
```

```
        //Axes 1/2 setting based on pulse equivalent programming mode
        retn = adt_set_unit_mode(index[0], 1, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        retn = adt_set_unit_mode(index[0], 2, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        //Corresponding interpolation axis is set to programming mode based on pulse
equivalent
        retn = adt_set_unit_mode(index[0], INPA_AXIS, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
        //Axis 1/2 pulse equivalent set to mm=1000pulse
        retn = adt_set_pulse_equiv(index[0], 1, 1000);
        VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
        retn = adt_set_pulse_equiv(index[0], 2, 1000);
        VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
        //Axis INPA_AXIS, startv=5mm/s, maxv=20mm/s, acc=120mm/s2
        retn = adt_set_startv(index[0], INPA_AXIS, 5);
        VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
        retn = adt_set_maxv(index[0], INPA_AXIS, 20);
        VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
        retn = adt_set_acc(index[0], INPA_AXIS, 120);
        VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
        retn = adt_set_admode(index[0], INPA_AXIS, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
        //Confirm execution
        CONFIRM_DRIVE(0);
        int AxisList[2] = {1,2};
        double PosList[2] = {0,0}, ctr_list[2] = {50, 0};
        //Axis/2 plane arc interpolation
        //The current position is the start point, relative position drive, the current position is
the end point, then the relative position of the target position is (0, 0)
        //Circle center coordinates (50, 0), plane full circule arc interpolation
        retn = adt_inp_arc2_unit(index[0], INPA_AXIS, 0, AxisList, PosList, ctr_list, 0, 0);
```

```
    VERIFY_RETURN_MSG(retn, "adt_inp_arc2_unit", 1);
    system("pause");
    return 0;
}
```

7.3.6. Spherical Arc Interpolation

Spherical arc interpolation is suitable for simplifying the teaching operation of complex graphics. Its accuracy is dynamically determined by the interpolation speed, which avoids the contradiction between the discrete error and the speed of the small line segment fitting method in the spatial pattern. Both helical interpolation and planar arc interpolation can be easily achieved based on spherical arc interpolation.



For two-dimensional arc interpolation interface of the 09 series motion control card, refer to adt_inp_arc3_unit.

The spherical arc interpolation function is an advanced interpolation function in the programming mode based on pulse equivalent, and the axis based on pulse programming mode can't participate in the spherical arc interpolation.

7.3.7. Cache interpolation

In the actual machining trajectory, the combination of the above trajectories or the process requirement of continuous interpolation of large capacity small line segments may occur.

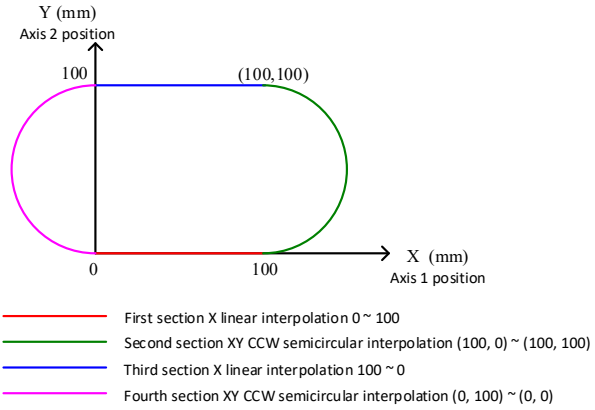
09 series motion control card has a large capacity multi-axis cache area, which makes CAM discrete data well restored to the processing model.

The cache interpolation function and the speed look-ahead function are complementary. The cache interpolation function is automatically turned on when the speed look-ahead function is turned on. The function is also introduced together with the speed look-ahead.

09 series motion control card provides 10,000-segment cache interpolation margin, which means that 10,000-segment interpolation instructions can be buffered at the same time. When the cache area is full, it must wait for the vacancy in the cache area to continue to push the interpolation instruction. For the query of cache margin, refer to adt_get_inp_fifo_len.

7.3.8. Routine 7-5 Cache Interpolation

This case implements the following traces under the standard interpolation drive and cache interpolation drive.



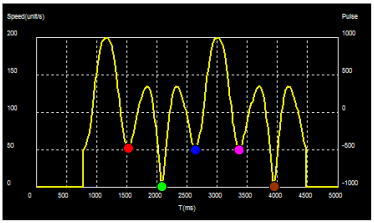

```
int main(int argc, char* argv[])
{
```

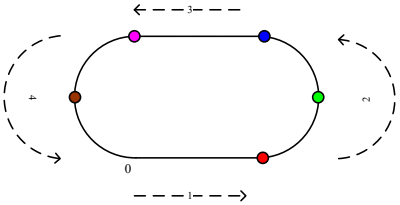
```
int card_count = 0;//Number of system control cards
int index[10] = {0};//Index of available control cards of the system
int axis_count = 0;//Number of available axes of the control card
//Initialize control card
int retn = adt_initial(&card_count);
//Parse and print error message
VERIFY_RETURN_MSG(retn, "adt_initial", 1);
//Get available control card index
retn = adt_get_card_index(&card_count, index);
VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
//The first available control card
//Axes 1/2 setting based on pulse equivalent programming mode
retn = adt_set_unit_mode(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
retn = adt_set_unit_mode(index[0], 2, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
//Corresponding interpolation axis is set to programming mode based on pulse
equivalent
retn = adt_set_unit_mode(index[0], INPA_AXIS, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
//Axis 1/2 pulse equivalent set to mm=1000pulse
retn = adt_set_pulse_equiv(index[0], 1, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
retn = adt_set_pulse_equiv(index[0], 2, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
//Axis INPA_AXIS, startv=50mm/s, maxv=200mm/s, acc=400mm/s2
retn = adt_set_startv(index[0], INPA_AXIS, 50);
VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
retn = adt_set_maxv(index[0], INPA_AXIS, 200);
VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
retn = adt_set_acc(index[0], INPA_AXIS, 400);
VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
```

```
        retn = adt_set_admode(index[0], INPA_AXIS, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
        //Determine whether to enable speed look-ahead according to user input
        int mode = 0;
        printf("Whether to enable speed look-ahead? 1 - Yes, 0 - No\n");
        cin>>mode;
        retn = adt_set_speed_pretreat_mode(index[0], INPA_AXIS, mode);
        VERIFY_RETURN_MSG(retn, "adt_set_speed_pretreat_mode", 1);
        //Query the cache margin, and continue to perform cache interpolation when the
cache margin is exceeded
        int fifo_len = 0;
        while(true)
        {
                retn = adt_get_inp_fifo_len(index[0], INPA_AXIS, &fifo_len);
                VERIFY_RETURN_MSG(retn, "adt_get_inp_fifo_len", 1);
                if(100 <= fifo_len)
                        break;
        }
        //Confirm execution
        CONFIRM_DRIVE(0);
        //---------------The first segment, X linear interpolation 0~100---------------//
        int axis1[1] = {1};
        double pos1[1] = {100};
        retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, axis1, pos1, 0);
        VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
      //----The second segment, XY counterclockwise semi-arc interpolation (100, 0) ~ (100,
100)----//
        //Arc parameter setting method in relative position mode
        //Target position of X is 100, current position is 100, relative position parameter =
target position - current position = 0
        //Target position of Y is 100, current position is 0, relative position parameter = target
position - current position = 100
```

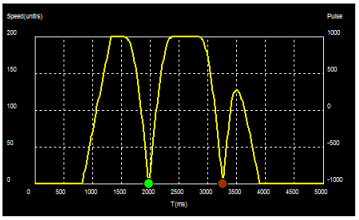
```
        //Target position of X circle center is 100, current position is 100, relative position
parameter = target position - current position = 0
        //Target position of Y circle center is 50, current position is 0, relative position
parameter = target position - current position = 50
        int AxisList[2] = {1, 2};
        double PosList[2] = {0, 100};
        double ctr_list[2] = {0, 50};
        retn = adt_inp_arc2_unit(index[0], INPA_AXIS, 0, AxisList, PosList, ctr_list,
                0/* counterclockwise*/, 0/* relative position drive mode*/);
        VERIFY_RETURN_MSG(retn, "adt_inp_arc2_unit", 1);
      //-------------The third segment, X linear interpolation 100 ~ 0---------------//
        pos1[0] = 0;//Absolute position mode
        retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, axis1, pos1, 1);
        VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
        //-----The fourth segment, XY counterclockwise semi-arc interpolation (0, 100) ~ (0, 0)-
---//
        //Arc parameter setting method in absolute position mode
        //Target position of X is 0, absolute position parameter = target position = 0
        //Target position of Y is 0, absolute position parameter = target position = 0
        //Target position of X circle center is 0, absolute position parameter = target position =
0
        //Target position of Y circle center is 50, absolute position parameter = target position
= 50
        PosList[0] = 0, PosList[1] = 0;
        ctr_list[0] = 0, ctr_list[1] = 50;
        retn = adt_inp_arc2_unit(index[0], INPA_AXIS, 0, AxisList, PosList, ctr_list,
                0/*Counterclockwise*/, 1/*absolute position drive mode*/);
        VERIFY_RETURN_MSG(retn, "adt_inp_arc2_unit", 1);
        //-------------------------------------------------//
        system("pause");
        return 0;
}
```

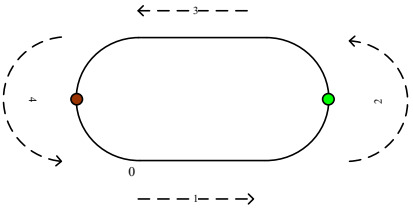
When the speed look-ahead function is not turned on, the 4-segment interpolation is the standard interpolation, and the axis 1 speed curve participating in the interpolation is as shown in the figure below. The five speed turning points in the figure correspond to the five position points in the interpolation trajectory by color respectively.





When the speed look-ahead function is turned on, the 4-segment interpolation is cache interpolation, and the axis 1 speed curve participating in the interpolation is as shown in the figure below. The two speed turning points in the figure correspond to the two position points in the interpolation trajectory by color.





It can be seen from the actual effect that the two speed turning points of green and brown are the speed reversal points that the axis 1 must undergo in arc interpolation, and the other three speed turning points can be transited smoothly after cache interpolation is turned on by the speed look-ahead mode.

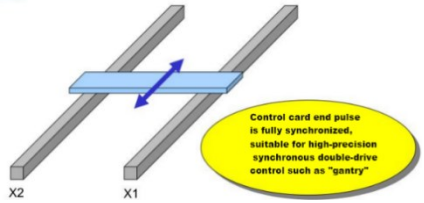
7.3.9. Gantry Double Drive

7.3.9.1. Function Introduction

The gantry double drive can maintain strict synchronization of the pulse between one principal axis and N slave axes, and its synchronization accuracy is much higher than that of the interpolation drive. It is suitable for synchronous double drive control of the gantry structure.

Before gantry double drive control, the basic parameters of the slave axis should be set, such as programming mode, pulse equivalent, etc. After the principal-slave axis relationship of the gantry double drive is set, some basic parameters of the slave axis can't be changed in real time.

Refer to adt_set_follow_axis for the gantry double drive control interface.



7.3.9.2. Routine 7-6 Gantry Double Drive

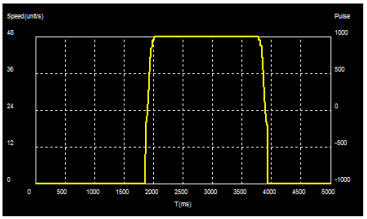
The case involves the drive status and stop information check functions, as well as the auxiliary operator DecodeStopdata for stopping the information check.

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //The first available control card
```

```
//Axes 1/2 setting based on pulse equivalent programming mode
retn = adt_set_unit_mode(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
retn = adt_set_unit_mode(index[0], 2, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
//Axis 1/2 pulse equivalent set to mm=1000 pulse
retn = adt_set_pulse_equiv(index[0], 1, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
retn = adt_set_pulse_equiv(index[0], 2, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
//Axis 1, startv=20mm/s, maxv=50mm/s, acc=200mm/s2
retn = adt_set_startv(index[0], 1, 20);
VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
retn = adt_set_maxv(index[0], 1, 50);
VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
retn = adt_set_acc(index[0], 1, 200);
VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
retn = adt_set_admode(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_set_admode", 1);
//Determine whether to synchronize and follow according to user input
int axis = 0;
printf("Whether to make the axis synchronize and follow drive?\n1 - Yes, 0 - No\n");
cin>>axis;
retn = adt_set_follow_axis(index[0], 2, axis);
VERIFY_RETURN_MSG(retn, "adt_set_follow_axis", 1);
//Confirm execution
CONFIRM_DRIVE(0);
//Axis quantitative drive
retn = adt_pmove_unit(index[0], 1, 100, 0);
VERIFY_RETURN_MSG(retn, "adt_pmove_unit", 1);
int stt = 0, stpdata = 0;
//Drive state and stop information check
```

```
    while(true)
    {
        DoEvent();
        retn = adt_get_axis_status(index[0], 1, &stt);
        VERIFY_RETURN_MSG(retn, "adt_get_axis_status", 1);
        if(0 == stt)
        {
            retn = adt_get_stopdata(index[0], 1, &stpdata);
            VERIFY_RETURN_MSG(retn, "adt_get_stopdata", 1);
            if(0 != stpdata)
            {
                char msg[256] = {0};
                DecodeStopdata(msg, 256, index[0], 1, stpdata);
                printf(msg);
            }
            break;
        }
        Sleep(1);
    }
    retn = adt_set_follow_axis(index[0], 2, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_follow_axis", 1);
     printf("Drive end, sync released!\n");
    system("pause");
    return 0;
}
```

When the synchronous follow drive is started, the speed curve of axis 2 is as shown below.



7.3.10. Drive Status Detection, Drive Stop and Stop Information

7.3.10.1. Function Introduction

The interface adt_get_axis_status can get the drive state of the specified axis, that is, the idle state (0), motion state (1) or pause state (2). The interface is also applicable to the virtual interpolation axis.

When the drive state is stopped, the stop information of specified axis can be obtained by agt_get_stopdata, that is, whether the axis is stopped normally (the pulse has been fully driven), or stopped abnormally due to other factors (limit trigger, emergency stop, etc.).

Call ad_set_axis_stop to stop the drive for the specified axis.

When the axis ratio is 0 during point drive or continuous drive, that is, the axis stop instruction is called when the pause is made, the drive data of the axis will be cleared, and the drive will not be restored even if the ratio is restored. When the axis ratio is 0 during interpolation drive, that is, the axis stop instruction is called when the pause is made, the drive data of the axis will be retained and the drive will be restored when the ratio is restored. This process is usually used when interpolation tool setting is supported. To completely clear the interpolation drive data when the axis drive is stopped, adt_reset_card must be called.

The normal stop after drive execution and the drive stop caused by calling adt_set_axis_stop will not generate abnormal stop information, that is, the stop information of normal stop after drive execution and the axis stopped by the adt_set_axis_stop is 0.

7.3.10.2. Routine 7-7 Drive Status and Stop Information

Refer to the drive status and stop information check section of Routine 7-6 Gantry Double Drive.

Refer to the drive stop routine of Routine 6-4 Speed Ratio.

The following auxiliary operator is intended to parse abnormal drive stop information into readable string information, that is, stop information parsing.

```
void DecodeStopdata(char* msg, int len, int card, int axis, int stp_data)
{
    if(0==msg || 0>=len)
        return ;
```

```
        ZeroMemory(msg, len);
        if(stp_data&0x1)
                sprintf(msg, "card index '%d' axis '%d' hardware positive limit trigger!", card,
axis);
        else if(stp_data&0x10)
                sprintf(msg, "card index '%d' axis '%d' hardware negative limit trigger!", card,
axis);
        else if(stp_data&0x100)
                sprintf(msg, "card index '%d' axis '%d' home signal (STOP0) trigger!", card,
axis);
        else if(stp_data&0x1000)
                sprintf(msg, "card index '%d' axis '%d' encoder Z phase signal (STOP1)
trigger!", card, axis);
        else if(stp_data&0x10000)
                sprintf(msg, "hardware stop signal trigger!", card, axis);
        else if(stp_data&0x100000)
                sprintf(msg, "card index '%d' axis '%d' software positive limit trigger!", card,
axis);
        else if(stp_data&0x1000000)
                sprintf(msg, "card index '%d' axis '%d' software negative limit trigger!", card,
axis);
        else if(0 != stp_data)
                sprintf(msg, "card index '%d' axis '%d' unknown stop information!", card, axis);
        else
                sprintf(msg, "card index '%d' axis '%d' drive completed!", card, axis);
        strcat(msg, "\n");
}
```

Chapter 8 Homing

8.1. Function Introduction

09 series motion control card provides a secondary homing function with repeatability <1 pulse.

Before performing precise motion control, you need to set the home of each drive axis. Most motion platforms have a home sensor or a switch signal that can be used as a home sensor (such as positive/negative limit sensor and home sensor multiplexing). Homing is the process to find the position of the home sensor, and set the position to the coordinate home of the current drive axis.

The homing function provided by the 09 series motion control card is suitable for the homing of various modules such as linear module and circumferential module. It can control the activation and deactivation of homing direction and homing signal (home sensor STOP0/encoder Z phase signal STOP1), which side of the sensor the home stops, activation and deactivation of each hardware signal during homing, and the parameters such as active level, homing process speed, and home offset.

When the position of the home signal is searched at high speed during the homing process, the hardware positive/negative limit (LMT+/LMT-) is touched for the first time, and the homing axis automatically reverses the search for the home signal position. If the hardware positive/negative limit (LMT+/LMT-) is touched again or the hardware stop signal (EMGN) is touched at any time, or an external deceleration or emergency stop occurs at any time, the homing process will terminate and return the corresponding error code.

For the homing interface, refer to

Considering the cost or other external factors in some processes, you may do the following assembly for the module:

① Assemble hardware positive/negative limit sensor, no hardware home sensor

The homing process must have an available home signal. If the encoder Z phase signal is properly wired, it is enabled as home signal and the homing mode is properly set, the equipment can normally home;

If the encoder Z-phase signal is not enabled, the home signal STOP0 at the terminal block and the hardware negative limit (or hardware positive limit) signal must

be shorted to ensure that the machine home signal is available. When short-circuiting with the hardware negative limit, the home should be properly set to stop on the side of the hardware negative limit sensor close to the hardware positive limit sensor; when short-circuiting with the hardware positive limit, the home should be properly set to stop on the side of the hardware positive limit sensor close to the hardware negative limit sensor.

② Assemble hardware positive limit sensor and hardware home sensor, no hardware negative limit sensor

The purpose of most such assembly is to use the hardware home as hardware negative limit multiplexing after homing is completed. In this case, the hardware home stop signal (adt_set_stop0_mode) must be enabled after the homing is completed to ensure that the current axis drive is stopped when the hardware home signal is triggered. However, the hardware home signal is different from the hardware negative limit signal after all, that is, when the hardware negative limit signal is triggered, the current axis can be driven in the positive direction; when the hardware home signal is triggered, the current axis can't be driven in any way since the driving direction of the current axis at the time of triggering can't be determined. The hardware home signal must be disabled to drive the axis leaving the trigger position and then enabled again, or turn off the axis enable, and manually push the axis away from the hardware home signal trigger position.

To completely use the hardware negative limit function when the hardware negative limit sensor is not installed, the hardware negative limit and the hardware home signal can be shorted and set to the same active level on the terminal board, so the hardware home signal can be multiplexed as a hardware negative limit signal after homing.

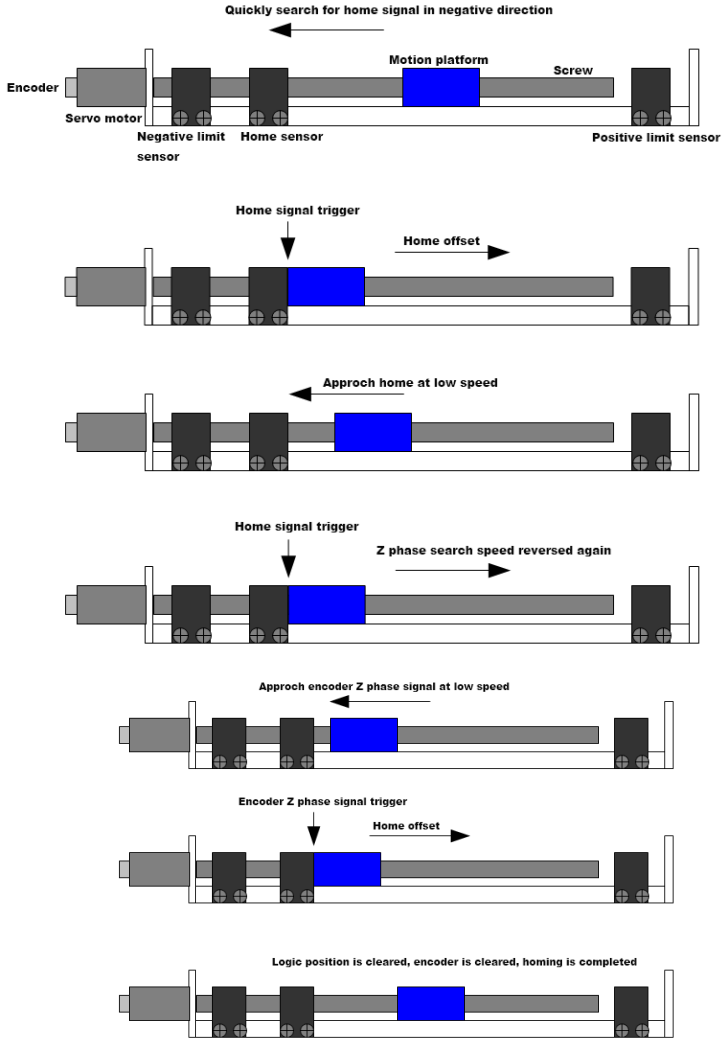
The assembly of the hardware negative limit sensor and the hardware home sensor without hardware positive limit sensor is similar.

③ Assemble hardware home sensor, no hardware positive/negative limit sensor

This assembly method can't ensure that the homing process has no risk. It is only possible to ensure that the module is always on the same side of the home signal sensor or ensure correct homing direction by other means. The home signal can be short-circuited with either the hardware positive limit signal or the hardware negative limit

signal on the terminal board. After homing, enable one of the hardware limit signals to increase the safety of the equipment. Do not short connect the home signal to the hardware positive/negative limit at the same time.

The principle of secondary homing is described as follows. Some processes can be skipped according to the parameter settings (e.g. enable/disable of home sensor STOP0 / encoder Z phase signal STOP1, home offset = 0, etc.).



Homing interface reference

adt_set_home_mode/adt_set_home_speed

adt_set_home_process/adt_get_home_status

8.2. Routine 8-1 Homing

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    /1st card, axis 1 homing
    // mode = 0 -- Negative homing, exit home in positive direction, then approach home
in negative direction
    //STOP0 signal is the machine home, STOP1 signal is the encoder home
    //stop0 = 0 -- Machine home active low
    //limit = 0 -- Hardware positive/negative limit enabled, active low
    //stop1 = 2 -- Do not search encoder Z-phase signal
    //back_range = 20 -- Exit machine home distance, in mm
    //z_range = 0 -- Do not search encoder Z-phase signal; this parameter is invalid
    //offset = 5 -- Home offset mm
    retn = adt_set_home_mode(index[0], 1, 0, 0, 0, 2, 20, 0, 5);
    VERIFY_RETURN_MSG(retn, "adt_set_home_mode", 1);
    retn = adt_set_home_speed(index[0], 1, 0, 20, 5, 40, 5);//Homing speed setting
    VERIFY_RETURN_MSG(retn, "adt_set_home_speed", 1);
    //Confirm execution
```

```
    CONFIRM_DRIVE(0);
    //Start homing
    char msg[100] = {0};
    retn = adt_set_home_process(index[0], 1);//Drive homing
    VERIFY_RETURN_MSG(retn, "adt_set_home_process", 1);
    while(true)
    {
        DoEvent();
        retn = adt_get_home_status(index[0], 1);
        if(0 < retn)
            continue;
        if(0 == retn)
        {
            printf("X-axis homing completed!");
            break;
        }
        int step = (-retn)%1000;
        sprintf(msg, "Axis homing step %d exception, homing failed!", step);
        printf(msg);//Print error message
        Sleep(1);
    }
    system("pause");
    return 0;
}
```

Chapter 9 Position Control

9.1. Function Introduction

The logic position is the total number of pulses that the control card issues immediately since the last time the coordinate position was reset.

Reference for control interface of logic position

adt_set_command_pos/adt_get_command_pos

The encoder position (known as the actual position in this manual) is the total number of pulses actually received by the encoder since the last reset of the coordinate position when there is encoder feedback.

Reference for control interface of actual position

adt_set_actual_pos/adt_get_actual_pos

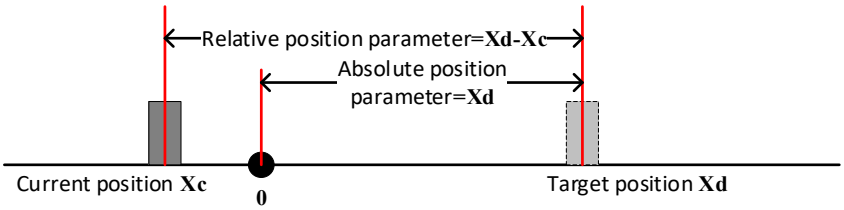
The target position is the logic position that the axis is about to arrive after the current instruction of the control card is executed. This position is usually used to verify whether the current position matches the target position after the end of the drive to ensure the drive accuracy.

Control interface reference for target location

adt_get_target_pos_unit is based on the target position of the pulse equivalent programming mode axis

adt_get_target_pos_pulse is based on the target position of the pulse programming mode axis

Relative position and absolute position are mostly used in position parameters of drive instructions. The relative position is the difference between the drive target position and the current position. The absolute position is the coordinate position of the drive target position with respect to 0 o'clock.



9.2. Routine 9-1 Position Control

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //Axis 1 is programmed based on pulse equivalent mode
    retn = adt_set_unit_mode(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
    //Axis 2 is programmed based on pulse mode
    retn = adt_set_unit_mode(index[0], 2, 1);
    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
    //Axis 1 pulse equivalent set to mm=1000pulse
    retn = adt_set_pulse_equiv(index[0], 1, 1000);
    VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
    //Clear position
    retn = adt_set_command_pos(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_command_pos", 1);
    retn = adt_set_actual_pos(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_command_pos", 1);
    retn = adt_set_command_pos(index[0], 2, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_command_pos", 1);
    retn = adt_set_actual_pos(index[0], 2, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_command_pos", 1);
    //Axis 1, startv=10mm/s, maxv=50mm/s, acc=100mm/s2
```

```
        retn = adt_set_startv(index[0], 1, 10);
        VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
        retn = adt_set_maxv(index[0], 1, 50);
        VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
        retn = adt_set_acc(index[0], 1, 100);
        VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
        //Axis2, startv=1000pulse/s, maxv=5000pulse/s, acc=10000pulse/s2
        retn = adt_set_startv(index[0], 2, 1000);
        VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
        retn = adt_set_maxv(index[0], 2, 5000);
        VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
        retn = adt_set_acc(index[0], 2, 10000);
        VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
        //Confirm execution
        CONFIRM_DRIVE(0);
        //Axis 1, relative drive mm
        retn = adt_pmove_unit(index[0], 1, 100, 0);
        VERIFY_RETURN_MSG(retn, "adt_pmove_unit", 1);
        //Axis 2, relative drive pulse
        retn = adt_pmove_pulse(index[0], 2, 50000, 0);
        VERIFY_RETURN_MSG(retn, "adt_pmove_pulse", 1);
        //Get the target location
        double tgt_mm = 0;
        long tgt_pulse = 0;
        retn = adt_get_target_pos_unit(index[0], 1, &tgt_mm);
        VERIFY_RETURN_MSG(retn, "adt_get_target_pos_unit", 1);
        retn = adt_get_target_pos_pulse(index[0], 2, &tgt_pulse);
        VERIFY_RETURN_MSG(retn, "adt_get_target_pos_pulse", 1);
        char msg[256] = {0};
        sprintf(msg, " Axis/2 has been started, axis target position = %.4f mm, axis target
position = %d pulse\n",
                tgt_mm, tgt_pulse);
```

```
printf(msg);
int stt1 = 0, stt2 = 0;
while(true)
{
    retn = adt_get_axis_status(index[0], 1, &stt1);
    VERIFY_RETURN_MSG(retn, "adt_get_axis_status", 1);
    retn = adt_get_axis_status(index[0], 2, &stt2);
    VERIFY_RETURN_MSG(retn, "adt_get_axis_status", 1);
    if(0 == stt1+stt2)
    {
        retn = adt_get_stopdata(index[0], 1, &stt1);
        VERIFY_RETURN_MSG(retn, "adt_get_stopdata", 1);
        if(0 != stt1)
            printf("Axis is abnormally stopped!\n");
        retn = adt_get_stopdata(index[0], 2, &stt2);
        VERIFY_RETURN_MSG(retn, "adt_get_stopdata", 1);
        if(0 != stt2)
            printf("Axis is abnormally stopped!\n");
        break;
    }
}
//Get current position
long cmd1 = 0, cmd2 = 0, act1 = 0, act2 = 0;
retn = adt_get_command_pos(index[0], 1, &cmd1);
VERIFY_RETURN_MSG(retn, "adt_get_stopdata", 1);
retn = adt_get_command_pos(index[0], 2, &cmd2);
VERIFY_RETURN_MSG(retn, "adt_get_stopdata", 1);
retn = adt_get_actual_pos(index[0], 1, &act1);
VERIFY_RETURN_MSG(retn, "adt_get_stopdata", 1);
retn = adt_get_actual_pos(index[0], 2, &act2);
VERIFY_RETURN_MSG(retn, "adt_get_stopdata", 1);
sprintf(msg, "axis 1 -- logic position: %d actual position: %d\n", cmd1, act1);
```

```
        printf(msg);
        sprintf(msg, "axis 2 -- logic position: %d actual position: %d\n", cmd2, act2);
        printf(msg);
        system("pause");
        return 0;
}
```

Chapter 10 Input/Output Control

10.1. Output Control

10.1.1. Function Introduction

Output control is to write or read the high/low state of the output port.

09 series motion control card supports single output port control adt_write_outbit and output port group control adt_write_outport, single output port status reading adt_read_outbit and output port status group reading adt_read_outport.

10.1.2. Routine 10-1 Output Port Control

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //Confirm execution
    CONFIRM_DRIVE(0);
    //Single output port control, high level output
    retn = adt_write_outbit(index[0], 16, 1);
    VERIFY_RETURN_MSG(retn, "adt_write_outbit", 1);
    //Output port group control, group OUT0~OUT15, level status is
    //1010101010101010
    retn = adt_write_outport(index[0], 1, 0xaaaa);
    VERIFY_RETURN_MSG(retn, "adt_write_outport", 1);
    Sleep(2000);//Maintain state ms
    //Read status of a single output port
```

```
    int stt = 0;
    retn = adt_read_outbit(index[0], 16, &stt);
    VERIFY_RETURN_MSG(retn, "adt_read_outbit", 1);
    if(1 != stt)
        printf("Output port status is abnormal!\n");
    //Read output port status by group
    unsigned long lv_map = 0;
    retn = adt_read_outport(index[0], 1, &lv_map);
    if(0xaaaa != lv_map)
        printf("Output port group (OUT0~OUT15) status abnormal!\n");
    //Output port status recover
    retn = adt_write_outbit(index[0], 16, 0);
    VERIFY_RETURN_MSG(retn, "adt_write_outbit", 1);
    retn = adt_write_outport(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_write_outport", 1);
    system("pause");
    return 0;
}
```

10.2. Input Control

10.2.1. Function Introduction

Input control is to read the high/low state of the input port.

09 series motion control card supports single input port status reading adt_read_inbit and input port status group reading adt_read_inport.

10.2.2. Routine 10-2 Input Port Control

```
int main(int argc, char* argv[])
{
    char msg[256] = {0};
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //Input filter control
    retn = adt_set_input_filter(index[0], 0, 10);
    VERIFY_RETURN_MSG(retn, "adt_set_input_filter", 1);
    retn = adt_set_input_filter(index[0], 1, 10);
    VERIFY_RETURN_MSG(retn, "adt_set_input_filter", 1);
    //Confirm execution
    CONFIRM_DRIVE(0);
    //Read status of a single input port
    int stt = 0;
    retn = adt_read_inbit(index[0], 16, &stt);
    VERIFY_RETURN_MSG(retn, "adt_read_inbit", 1);
    cout<<"Input port IN16 status: "<<stt<<"\n";
    //Read input port status by group, group 1 IN0~IN15
```

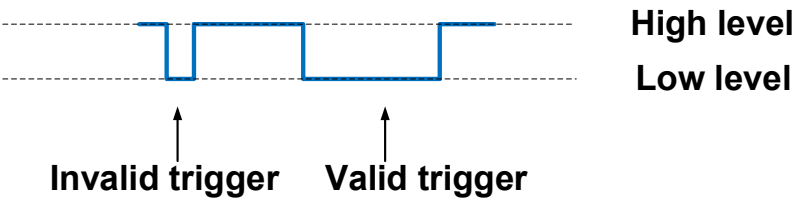
```
    bitset<16> bs;
    unsigned long lv_map = 0;
    retn = adt_read_inport(index[0], 1, &lv_map);
    VERIFY_RETURN_MSG(retn, "adt_write_outport", 1);
    bs = lv_map;
    cout<<"Input port IN0~IN15 status: "<<bs<<"\n";
    system("pause");
    return 0;
}
```

10.3. Input Filter

10.3.1. Function Introduction

When the user equipment is running, the input port may instantaneously jump due to external interference or other environmental factors, that is, the level is rapidly reversed, forming an illusion of triggering.

Taking the input port ground level as an example, the input filter phenomenon can be described as follows:



When the equipment is running normally, occasional fleeting accidental triggering of the input port can cause inexplicable downtime or other failures. If the sensor sensitivity is too high, the equipment environment dust may easily cause false triggering, or the hardware circuit wiring is vulnerable, the sensor signal is unstable. Input filter sets the input port trigger time threshold, and unqualified triggers are filtered and treated as noise. 09 series motion control card provides a filter mechanism for the fast input port and extended input port of the equipment. Refer to adt_set_input_filter for the input filter interface.

10.3.2. Routine 10-3 Input Filter Control

Refer to the filter section of Routine 10-2 Input Port Control.

Chapter 11 Cache Event

09 series motion control card provides users with 1000 cache control event capacity, that is, up to 1000 cache control events can be accommodated during high-speed multi-segment cache interpolation drive.

The current cache control instruction capacity of the system can be queried by adt_get_fifo_event_len, and the cache event instructions that haven't been executed can be cleared by calling adt_clear_fifo_event after the external stop.

11.1. Cache Output Control

11.1.1. Function Introduction

Cache output control implements hardware-level seamless control of specified output ports during cache interpolation.

During cache output control, the maximum speed at the time of output control can also be constrained. That is, when the maximum speed V0 has been reached during cache interpolation, the user agrees to control the output port B when the specified interpolation axis reaches the space coordinate P, and the speed of the interpolation axis at point P must be <=V1. If V0<=V1, the interpolation axis drive speed will not change during cache output control. If V0>V1, the interpolation axis will decelerate according to the predetermined setting before reaching the position P, ensuring that the speed is V1 when reaching position P, and the output of port B is under control. After the cache output control is completed, the interpolation axis will re-plan the drive speed of the remaining interpolation according to the remaining cache interpolation trajectory. The typical application of the cache output control is fly shot, that is, the high-speed camera starts shooting when it reaches the specified position and the drive does not slow down or decrease to the specified speed.

Refer to adt_set_fifo_outbit for the cache output control interface.

11.1.2. Routine 11-1 Cache Output Control

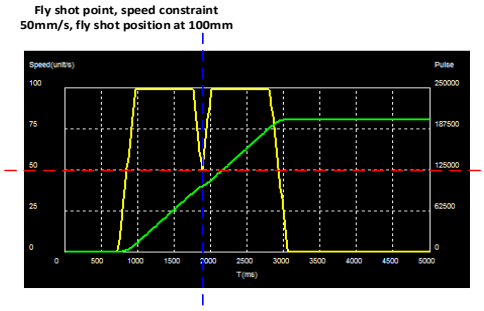
In the following case, the output port OUT16 is used to achieve simple fly shot function.

```
int main(int argc, char* argv[])
```

```
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //Speed look-ahead
    retn = adt_set_speed_pretreat_mode(index[0], INPA_AXIS, 1);
    VERIFY_RETURN_MSG(retn, "adt_set_speed_pretreat_mode", 1);
    // Axis 1 in pulse equivalent programming mode, pulse equivalent pulse/mm
    retn = adt_set_unit_mode(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
    retn = adt_set_pulse_equiv(index[0], 1, 1000);
    VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
    //Corresponding interpolation axis is set to programming mode based on pulse
equivalent
    retn = adt_set_unit_mode(index[0], INPA_AXIS, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
    //Speed setting
    retn = adt_set_startv(index[0], INPA_AXIS, 20);
    VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
    retn = adt_set_acc(index[0], INPA_AXIS, 400);
    VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
    retn = adt_set_maxv(index[0], INPA_AXIS, 100);
    VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
    //Confirm execution
    CONFIRM_DRIVE(0);
```

```
    //1. 驱动到拍照位mm
    //1. Drive to shooting position mm
    //2. Turn on OUT16
    //3. Drive mm to keep OUT16 on
    //4. Turn off OUT16
    //5. Drive to safe position mm
    int AxisList[1] = {1};
    double PosList1[1] = {100}, PosList2[1] = {2};
    retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, AxisList, PosList1, 0);
    VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
    retn = adt_set_fifo_outbit(index[0], INPA_AXIS, 0, 16, 1, 50);
    VERIFY_RETURN_MSG(retn, "adt_set_fifo_outbit", 1);
    retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, AxisList, PosList2, 0);
    VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
    retn = adt_set_fifo_outbit(index[0], INPA_AXIS, 0, 16, 0, -1);
    VERIFY_RETURN_MSG(retn, "adt_set_fifo_outbit", 1);
    retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, AxisList, PosList1, 0);
    VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
    system("pause");
    return 0;
}
```

Under the above functions, the speed curve of the axis 1 participating in the cache interpolation is as shown below.



11.2. Cache PWM control

11.2.1. Function Introduction

Cache PWM control realizes high-precision high-frequency PWM control during the cache interpolation process, that is, high-precision high-frequency flip control of the specified output port. Cache PWM control has a level duration accuracy up to 10ns. Cache PWM control can also constrain the maximum speed during control. Refer to 11.1 Cache Output Control Function Description.

During cache PWM control, the reading and writing operations corresponding to the output ports occupied by the PWM will be disabled.

The typical application of the cache PWM control is the control of the dispensing valve.

Refer to adt_set_fifo_pwm for the cache PWM control interface.

11.2.2. Routine 11-2 Cache PWM Control

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //Speed look-ahead
    retn = adt_set_speed_pretreat_mode(index[0], INPA_AXIS, 1);
    VERIFY_RETURN_MSG(retn, "adt_set_speed_pretreat_mode", 1);
    //Axis 1 in pulse equivalent programming mode, pulse equivalent pulse/mm
    retn = adt_set_unit_mode(index[0], 1, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
    retn = adt_set_pulse_equiv(index[0], 1, 1000);
```

```
    VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
    //Corresponding interpolation axis is set to programming mode based on pulse
equivalent
    retn = adt_set_unit_mode(index[0], INPA_AXIS, 0);
    VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
    //Speed setting
    retn = adt_set_startv(index[0], INPA_AXIS, 20);
    VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
    retn = adt_set_acc(index[0], INPA_AXIS, 400);
    VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
    retn = adt_set_maxv(index[0], INPA_AXIS, 100);
    VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
    //Confirm execution
    CONFIRM_DRIVE(0);
    //1. Drive to PWM position mm
    //2. Turn on PWM
    //3. Drive mm to keep PWM on
    //4. Turn off PWM
    //5. Drive to safe position mm
    int AxisList[1] = {1};
    double PosList[1] = {100};
    retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, AxisList, PosList, 0);
    VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
    retn = adt_set_fifo_pwm(index[0], INPA_AXIS, 0, 12, 1, 0.5, 0.5, -1);
    VERIFY_RETURN_MSG(retn, "adt_set_fifo_pwm", 1);
    retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, AxisList, PosList, 0);
    VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
    retn = adt_set_fifo_pwm(index[0], INPA_AXIS, 0, 12, 1, 0.5, 0.5, -1);
    VERIFY_RETURN_MSG(retn, "adt_set_fifo_pwm", 1);
    retn = adt_inp_move_unit(index[0], INPA_AXIS, 0, 1, AxisList, PosList, 0);
    VERIFY_RETURN_MSG(retn, "adt_inp_move_unit", 1);
    system("pause");
```

```
    return 0; }
```

11.3. Cache Delay Control

11.3.1. Function Introduction

The cache delay control achieves the state preservation of the current cache control during the cache interpolation process. During the delay control, all cache interpolation drives will stop and all cache event control states will be maintained.

Refer to adt_set_fifo_delay for the cache delay control interface.

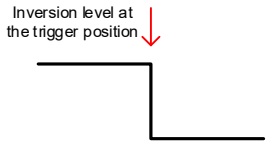
Chapter 12 Position Comparison

12.1.1. Function Introduction

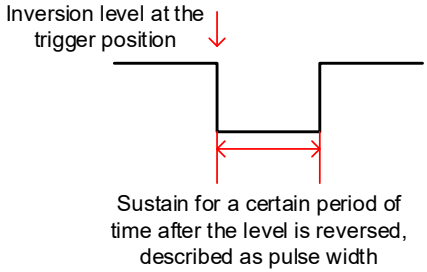
09 Series motion control card provides a one-dimensional position comparator that can accommodate 1000 position comparison points.

The data source for one-dimensional position comparison can be single-axis logic position or single-axis actual encoder position. The output control modes are level inversion and pulse width output.

Level inversion indicates that the normal level set in advance will be reversed immediately when the position comparison point is triggered. The level shape can be described as shown below.



Pulse width output indicates that the normal level set in advance will be reversed immediately and restore after a certain period of time when the position comparison point is triggered. The level shape can be described as shown below.



The mode of the above-mentioned one-dimensional position comparator can be set and gotten through the interface. adt_set_compare_mode/adt_get_compare_mode. The one-dimensional position comparison data can be added one by one, by means of array list, or linearly by setting the start point, spacing and the number of comparison data. References for related interfaces:

adt_add_compare_point

adt_add_compare_table

adt_add_compare_linear

Up to 1000 one-dimensional position comparison points can be added at the same time, but the comparison points can be increased infinitely by cache addition, that is, query the margin of the current one-dimensional position comparison point through the interface adt_get_compare_len, continue to wait if the margin is 0, and add comparison points if there is a margin.

You can use adt_get_compare_status to query the number of position comparison points that have been triggered and clear the issued position comparison points through adt_clear_compare_point. If you want to continue to use the one-dimensional position comparator after the position comparison points are cleared, you need to reset the comparison mode of the one-dimensional position comparator.

The one-dimensional position comparison control does not limit the programming mode of the axes participating in the position comparison.

Fly shot is one of the typical applications of one-dimensional position comparator. When the one-dimensional position comparator is in effect, the fast output port of the reading/writing comparator operation is invalid. The one-dimensional position comparator and the PWM function can't use the same fast output port at the same time.

12.1.2. Routine 12-1 Position Comparator

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    int len = 0, cmp_count = 0;
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
```

```
//Clear comparison points
retn = adt_clear_compare_point(index[0], 1);
VERIFY_RETURN_MSG(retn, "adt_clear_compare_point", 1);
//Set position comparison mode, take port, take axis, compare logic position, reverse
level, normal level is low
retn = adt_set_compare_mode(index[0], 1, 1, 0, 0, 0, 1);
VERIFY_RETURN_MSG(retn, "adt_set_compare_mode", 1);
//Add single position comparison point, the comparison point is
retn = adt_add_compare_point(index[0], 1, 1000);
VERIFY_RETURN_MSG(retn, "adt_add_compare_point", 1);
//Check comparison point margin
retn = adt_get_compare_len(index[0], 1, &len);
VERIFY_RETURN_MSG(retn, "adt_get_compare_len", 1);
//Wait for a margin to add comparison points
while(0 != len);
//Add a set of position comparison points
long table[10] = {1100, 1200, 1300, 1400, 1500, 1600, 1700,1800, 1900, 2000};
retn = adt_add_compare_table(index[0], 1, table, 10);
VERIFY_RETURN_MSG(retn, "adt_add_compare_table", 1);
//Check comparison point margin
retn = adt_get_compare_len(index[0], 1, &len);
VERIFY_RETURN_MSG(retn, "adt_get_compare_len", 1);
//Wait for a margin to add comparison points
while(0 != len);
//Wait for a margin to add comparison points
retn = adt_add_compare_linear(index[0], 1, 3000, 100, 10);
VERIFY_RETURN_MSG(retn, "adt_add_compare_linear", 1);
//Axis speed and configuration, based on pulse equivalent programming, pulse
equivalent
retn = adt_set_unit_mode(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
retn = adt_set_startv(index[0], 1, 20);
```

```
    VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
    retn = adt_set_acc(index[0], 1, 400);
    VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
    retn = adt_set_maxv(index[0], 1, 100);
    VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
    //Confirm execution
    CONFIRM_DRIVE(0);
    //Point drive axis
    retn = adt_pmove_unit(index[0], 1, 4, 0);
    VERIFY_RETURN_MSG(retn, "adt_pmove_unit", 1);
    //Check position comparison results in real-time
    int stt = 0, count = 0;
    while(true)
    {
        //Query number of comparison
        retn = adt_get_compare_status(index[0], 1, &stt);
        VERIFY_RETURN_MSG(retn, "adt_get_compare_status", 1);
        if(0!=stt && stt!=count)
            printf("%d points compared!\n", stt);
        count = stt;
        //Check whether all comparison points have been compared
        retn = adt_get_compare_len(index[0], 1, &len);
        VERIFY_RETURN_MSG(retn, "adt_get_compare_len", 1);
        if(1000 == len)
            break;
    }
    system("pause");
    return 0;
}
```

Chapter 13 Position Latching

13.1.1. Function Introduction

The position latch is a function that the logic position or encoder position is captured and saved to the hardware at the moment when the specified axis triggers a specific input port signal during the drive. Hardware-level capture performance guarantees latching accuracy within 1 pulse.

09 series motion control card can realize single position latch of home signal (STOP0) for each available solid axis, single position latch of encoder Z phase signal (STOP1), and continuous position latch of specified fast input port (IN32/IN33).

The latch position can be logic position (refer to adt_get_latch_command_pos for interface) or encoder location (refer to adt_get_latch_actual_pos for interface).

Usually, there is only one or less than one home signal, one or less than one encoder Z-phase signal for a single axis, so the latch of the home signal (STOP0) and the encoder Z-phase signal (STOP1) is only executed once. If multiple consecutive latches of the home signal (STOP0) and the encoder Z-phase signal (STOP1) are required, it is necessary to clear the latch data (refer to adt_clear_latch for the interface) and set the latch mode again (refer to adt_set_latch_mode for the interface) after each latch state trigger (refer to adt_get_latch_status for the interface).

Multiple consecutive latches can also be implemented using the fast input port provided by the system (IN32/IN33). The fast input port can continuously latch up to 128 position data of logic position and encoder position.

13.1.2. Routine 13-1 Position Latch Control

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
```

```
//Get available control card index
retn = adt_get_card_index(&card_count, index);
VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
//Axis in pulse equivalent programming mode, pulse equivalent pulse/mm
retn = adt_set_unit_mode(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
retn = adt_set_pulse_equiv(index[0], 1, 1000);
VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
//Speed setting
retn = adt_set_startv(index[0], 1, 20);
VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
retn = adt_set_acc(index[0], 1, 400);
VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
retn = adt_set_maxv(index[0], 1, 100);
VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
//Confirm execution
CONFIRM_DRIVE(0);
//Axis home signal (STOP0) latch mode, falling edge latch
retn = adt_set_latch_mode(index[0], 1, 0, 0);
VERIFY_RETURN_MSG(retn, "adt_set_latch_mode", 1);
//Axis positive drive
retn = adt_continue_move(index[0], 1, 0);
VERIFY_RETURN_MSG(retn, "adt_continue_move", 1);
//Latch state and latch position capture
int stt = 0;
long cmd = 0, enc = 0;
while(true)
{
    retn = adt_get_latch_status(index[0], 1, &stt);
    VERIFY_RETURN_MSG(retn, "adt_get_latch_status", 1);
    if(0 < stt)
    {
```

```
                printf("Axis home signal (STOP0) position latch triggered!\n");
                retn = adt_get_latch_command_pos(index[0], 1, &cmd);
                VERIFY_RETURN_MSG(retn,"adt_get_latch_command_pos",1);
                retn = adt_get_latch_actual_pos(index[0], 1, &enc);
                VERIFY_RETURN_MSG(retn, "adt_get_latch_actual_pos", 1);
                cout<<"Latch logic position: "<<cmd<<"\n";
                cout<<"Latch encoder position: "<<enc<<"\n";
                break;
            }
        }
        retn = adt_clear_latch(index[0], 1);
        VERIFY_RETURN_MSG(retn, "adt_clear_latch", 1);
        system("pause");
        return 0;
}
```

Chapter 14 DA Control

14.1.1. Function Introduction

A two-way DA output port is provided on the terminal block of 09 series 6-axis motion control card. This port can be used to continuously output 0~10V voltage stably, and the output precision is controlled at 0.01V.

Refer to adt_set_daout for the DA output interface.

14.1.2. Routine 14-1 DA Control

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //DA output, port 1 outputs 4V, port 2 outputs 8V
    retn = adt_set_daout(index[0], 1, 4);
    VERIFY_RETURN_MSG(retn, "adt_set_daout", 1);
    retn = adt_set_daout(index[0], 1, 8);
    VERIFY_RETURN_MSG(retn, "adt_set_daout", 1);
    system("pause");
    return 0;
}
```

Chapter 15 User Control

15.1.1. Function Introduction

09 series motion control card provides user password control. When the password has been set, the control card can't be used correctly if the verification fails.

The control card has no password when it leaves the factory. To use password control, you need to set the password first. The password must be within 128 digits, and the old and new passwords can't be the same.

If you forget your password, please send it back to the factory to crack it, so please backup it yourself after the password is set.

You can add password verification when the software is turned on. If the password verification fails more than 3 times, the card will be locked. To continue the verification, you must turn off the power and restart the control card, that is, restart the PC.

When user control is used for the first time, the old password will not be verified; once the user control password is set successfully, it will be verified in the future.

This function only provides a convenient solution for the application such as login, permission check, etc. Verification failure does not affect the normal use of the control card.

User control operator reference

adt_set_user_password –Modify/set user password

adt_check_password –User password check

15.1.2. Routine 15-1 User Control

```
int main(int argc, char* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
```

```
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    //User control, password is set to adtech
    retn = adt_set_user_password(index[0], "", "adtech");
    VERIFY_RETURN_MSG(retn, "adt_set_user_password", 1);
    retn = adt_check_password(index[0], "111");
    cout<<"Password '111' check"<<(retn?"Failed!\n":"Success!\n");
    retn = adt_check_password(index[0], "adtech");
    cout<<"Password'adtech' check"<<(retn?"Failed!\n":"Success!\n");
    //Password is set to null
    retn = adt_set_user_password(index[0], "adtech", "");
    VERIFY_RETURN_MSG(retn, "adt_set_user_password", 1);
    system("pause");
    return 0;
}
```

Chapter 16 Handwheel Control

16.1.1. Function Introduction

09 series motion control card provides single-axis handwheel motion function, which has two control modes for axis selection:

1. Select the following axis through the IO on the handheld box; support 1-6 axes. (ADT-CNC6A or the handheld box of same wiring definition is recommended for this function. For the wiring definition, please refer to 3.1.2 Port Definition of 15-pin Handwheel in the manual of the 09 series motion control card)

2. Select axis according to the axis parameters; support 1-8 axes

User control operator reference

adt_set_handwheel_move - Turn on the handwheel

adt_stop_handwheel - Turn off the handwheel

16.1.2. Routine 16-1 Handwheel Control

```
int _tmain(int argc, _TCHAR* argv[])
{
    int card_count = 0;//Number of system control cards
    int index[10] = {0};//Index of available control cards of the system
    int axis_count = 0;//Number of available axes of the control card
    //Initialize control card
    int retn = adt_initial(&card_count);
    //Parse and print error message
    VERIFY_RETURN_MSG(retn, "adt_initial", 1);
    //Get available control card index
    retn = adt_get_card_index(&card_count, index);
    VERIFY_RETURN_MSG(retn, "adt_get_card_index", 1);
    for (int i=1;i<=6; i++)
    {
        //Axis is set based on pulse + direction, pulse logic is , direction is
        retn = adt_set_pulse_mode(index[0], i, 1, 0, 0);
        VERIFY_RETURN_MSG(retn, "adt_set_pulse_mode", 1);
```

```
            //Axis is set based on pulse programming mode
            retn = adt_set_unit_mode(index[0], i, 0);
            VERIFY_RETURN_MSG(retn, "adt_set_unit_mode", 1);
            //Pulse equivalent is set to
            retn = adt_set_pulse_equiv(index[0], i, 1000);
            VERIFY_RETURN_MSG(retn, "adt_set_pulse_equiv", 1);
            //Start speed set to mm/s, the maximum speed mm/s, acceleration mm/s^2
            retn = adt_set_startv(index[0], i, 5);
            VERIFY_RETURN_MSG(retn, "adt_set_startv", 1);
            retn = adt_set_maxv(index[0], i, 20);
            VERIFY_RETURN_MSG(retn, "adt_set_maxv", 1);
            retn = adt_set_acc(index[0], i, 100);
            VERIFY_RETURN_MSG(retn, "adt_set_acc", 1);
        }
        //Turn on handwheel mode, and use mode 0 to select the following axis through the IO on
the handheld box
        retn = adt_set_handwheel_move(index[0], 0, 1);
        VERIFY_RETURN_MSG(retn, "adt_set_handwheel_move", 1);
        return 0;
}
```

Chapter 17 Details of Standard Instructions

17.1. Basic Control Class

adt_initial

Instruction Format		
int _stdcall adt_initial(int *CardNum, int mode=0)		
Instruction Description		
Initialize the control card. Only when the motion control card is initialized by calling this function, it is meaningful to call other functions.		
Parameter Description		
Name	Range	Description
CardNum	Not null	The number of available 09 series motion control cards of the system
mode	[0,1]	Alternate parameter, default option is 0
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Basic control class	
Related Instructions	adt_close_card	
Related Chapter	Initialize control card	
Related Routine	Routine 4-1 Basic Control	

adt_close_card

Instruction Format	
int _stdcall adt_close_card()	
Instruction Description	
After using, the motion control card should be closed to release the resources	
Parameter Description	
None	
Returned Value	
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction Basic control class

Related Instructions	adt_initial
Related Chapter	Close control card
Related Routine	Routine 4-1 Basic Control

adt_reset_card

Instruction Format		
int _stdcall adt_reset_card(int cardno)		
Instruction Description		
Reset the motion control card. It is often used to clear hardware emergency stop signal, abnormal stop information, invalid cache data, and motion library stop information. When the hardware emergency stop function is enabled and the hardware emergency stop signal is triggered, the axis can be driven normally when the hardware emergency stop lock is cleared by resetting the motion control card. When the motion library has exception, the exception must be cleared by resetting the motion control card before the system can be driven normally when the problem has been solved.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		

Instruction Classification	Standard instruction Basic control class
Related Instructions	adt_get_motion_err、adt_set_emergency_stop
Related Chapter	Reset control card
Related Routine	Routine 4-1 Basic Control

adt_soft_reboot

Instruction Format
int _stdcall adt_soft_reboot(int cardno)
Instruction Description
Software restart of the control card will first decelerate to stop all the drives of the current control card and clear the cache, and then reset all data of the control card to the initial state after power-on.

Software restart of the control card does not power down the control card. After the control card restarts, the application must be restarted and re-initialized in order to use the control card normally

Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number

Returned Value
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure

Instruction Classification	Standard instruction Basic control class
Related Instructions	
Related Chapter	Control card soft restart
Related Routine	

adt_get_card_index

Instruction Format
int _stdcall adt_get_card_index(int *count, int index[10])

Instruction Description
Get the number of available control cards and their indexes for the current system after initialization.

Parameter Description		
Name	Range	Description
count	Not null	The number of available 09 series motion control cards of the current system
index	Not null	The index of available 09 series motion control cards of the current system. The array size must be greater than the number of available 09 series motion control cards of the current system

Returned Value
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure

Instruction Classification	Standard instruction Basic control class

Related Instructions	
Related Chapter	Get available control card index
Related Routine	Routine 4-1 Basic Control

adt_get_total_axis

Instruction Format		
int _stdcall adt_get_total_axis(int cardno, int *count)		
Instruction Description		
Get the number of available solid axes for the currently available control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
count	Not null	Number of available solid axes of the control card
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Basic control class	
Related Instructions		
Related Chapter	Get the number of available solid axes of the control card	
Related Routine	Routine 4-1 Basic Control	

adt_get_motion_err

Instruction Format		
int _stdcall adt_get_motion_error(int cardno, int *errno)		
Instruction Description		
Get the error code of motion library exception of current motion control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
errno	Not null	Error code of motion library exception
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		

Instruction Classification	Standard instruction Basic control class
Related Instructions	
Related Chapter	
Related Routine	

adt_get_communication_err

Instruction Format		
int _stdcall adt_get_communication_err(int cardno, unsigned long *ErrCard,, unsigned long *ErrBox)		
Instruction Description		
Get the number of terminal block communication error of the current motion control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
ErrCard	Not null	Number of control card communication error
ErrBox	Not null	Number of terminal board communication error
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Basic control class	
Related Instructions		
Related Chapter		
Related Routine		

17.2. Version Information Class

adt_get_lib_ver

Instruction Format		
int _stdcall adt_get_lib_ver(int cardno, int *lib)		
Instruction Description		
Get the LIB library version number of current available control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
lib	Not null	LIB library version number of current available control card
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Basic control class	
Related Instructions	adt_get_motion_ver、adt_get_firmware_ver、adt_get_board_ver	
Related Chapter	Get control card version information	
Related Routine	Routine 4-1 Basic Control	

adt_get_motion_ver

Instruction Format		
int _stdcall adt_get_ motion_ver(int cardno, int * MotionVer)		
Instruction Description		
Get the motion library version number of current available control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
MotionVer	Not null	MOTION library version number of current available control card
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard		

Error Code for the reason of failure	
Instruction Classification	Standard instruction Basic control class
Related Instructions	adt_get_lib_ver、adt_get_firmware_ver、adt_get_board_ver
Related Chapter	Get control card version information
Related Routine	Routine 4-1 Basic Control

adt_get_firmware_ver

Instruction Format		
int _stdcall adt_get_firmware_ver(int cardno, int *fmw)		
Instruction Description		
Get the firmware version number of current available control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
fmw	Not null	Firmware version number of current available control card
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Basic control class	
Related Instructions	adt_get_lib_ver、adt_get_motion_ver、adt_get_board_ver	
Related Chapter	Get control card version information	
Related Routine	Routine 4-1 Basic Control	

adt_get_board_ver

Instruction Format		
int _stdcall adt_get_board_ver(int cardno, int *board)		
Instruction Description		
Get the terminal board version number of current available control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
board	Not	Terminal board version number of current available

	null	control card
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Basic control class	
Related Instructions	adt_get_lib_ver、adt_get_motion_ver、adt_get_firmware_ver	
Related Chapter	Get control card version information	
Related Routine	Routine 4-1 Basic Control	

adt_get_output_alarm

Instruction Format		
int _stdcall adt_get_output_alarm (int cardno, int *status)		
Instruction Description		
Get the output port overload alarm		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
status	Not null	Load status， 0:normal， 1: overload
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Basic control class	
Related Instructions		
Related Chapter		
Related Routine		

17.3. Resource configuration class

adt_set_emergency_stop

Instruction Format		
int _stdcall adt_set_emergency_stop(int cardno, int enable, int logic)		
Instruction Description		
Set the emergency stop signal mode of current control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
enable	[0,1]	Whether the hardware stop function is enabled 0: Disable 1: Enable
logic	[0,1]	Hardware stop signal active level 0: Active low 1: Active high
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_reset_card adt_get_emergency_stop	
Related Chapter	Hardware stop signal (EMGN)	
Related Routine	Routine 5-1 Configuration Method Routines	

adt_get_emergency_stop

Instruction Format		
int _stdcall adt_get_emergency_stop(int cardno, int *enable, int *logic)		
Instruction Description		
Get the emergency stop signal mode of current control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
enable	Not null	Whether the hardware stop function is enabled 0: Disable 1: Enable
logic	Not	Hardware stop signal active level

	null	0: Active low 1: Active high
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_emergency_stop	
Related Chapter	Hardware stop signal (EMGN)	
Related Routine		

adt_set_pulse_mode

Instruction Format		
int _stdcall adt_set_pulse_mode(int cardno, int axis, int type, int logic, int dir)		
Instruction Description		
Set the pulse mode of current control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
type	[0-2]	Pulse output mode 0: CCW/CW double pulse 1: PUL + DIR (pulse + direction) 2: 90° phase difference 2-phase pulse
logic	[0,1]	Pulse output logic 0: Positive 1: Negative
dir	[0,1]	Direction signal logic 0: Positive 1: Negative
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_get_pulse_mode	
Related Chapter	Pulse mode	

Related Routine	Routine 5-1 Configuration Method Routines

adt_get_pulse_mode

Instruction Format		
int _stdcall adt_get_pulse_mode(int cardno, int axis, int *type, int *logic, int *dir)		
Instruction Description		
Get the pulse mode of current control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
type	Not null	Pulse output mode 0: CCW/CW double pulse 1: PUL + DIR (pulse + direction) 2: 90° phase difference 2-phase pulse
logic	Not null	Pulse output logic 0: Positive 1: Negative
dir	Not null	Direction signal logic 0: Positive 1: Negative
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_pulse_mode	
Related Chapter	Pulse mode	
Related Routine		

adt_set_actual_count_mode

Instruction Format		
int _stdcall adt_set_actual_count_mode(int cardno, int axis, int type, int dir)		
Instruction Description		
Set the encoder count mode of current control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
type	[0,1]	Pulse input method 0: A/B phase pulse 1: Up/down (PPIN+PMIN)
dir	[0,1]	Direction signal logic 0: Positive 1: Negative
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_get_actual_count_mode	
Related Chapter	Encoder working mode	
Related Routine	Routine 5-1 Configuration Method Routines	

adt_get_actual_count_mode

Instruction Format		
int _stdcall adt_get_actual_count_mode(int cardno, int axis, int *type, int *dir)		
Instruction Description		
Get the encoder count mode of current control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
type	[0,1]	Pulse input method 0: A/B phase pulse 1: Up/down (PPIN+PMIN)
dir	[0,1]	Direction signal logic 0: Positive 1: Negative
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_actual_count_mode	
Related Chapter	Encoder working mode	
Related Routine		

adt_set_unit_mode

Instruction Format		
int _stdcall adt_set_unit_mode(int cardno, int axis, int mode)		
Instruction Description		
Set the current axis programming mode of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Solid axis or interpolation axis number
mode	[0,1]	Axis programming mode

		0: Based on unit 1: Based on pulse
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_get_unit_mode	
Related Chapter	Programming mode	
Related Routine	Routine 5-1 Configuration Method Routines	

adt_get_unit_mode

Instruction Format		
int _stdcall adt_get_unit_mode(int cardno, int axis, int *mode)		
Instruction Description		
Get the current axis programming mode of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Solid axis or interpolation axis number
mode	Not null	Axis programming mode 0: Based on unit 1: Based on pulse
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_unit_mode	
Related Chapter	Programming mode	
Related Routine		

adt_set_pulse_equiv

Instruction Format
int _stdcall adt_set_pulse_equiv(int cardno, int axis, double equiv)

Instruction Description		
Set the current axis pulse equivalent of the control card. It is suitable for axes based on pulse equivalent programming mode.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
equiv	[50, 20000]	Axis pulse equivalent
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_unit_mode、adt_get_unit_mode、adt_get_pulse_equiv	
Related Chapter	Pulse equivalent	
Related Routine	Routine 5-1 Configuration Method Routines	

adt_get_pulse_equiv

Instruction Format		
int _stdcall adt_get_pulse_equiv(int cardno, int axis, double *equiv)		
Instruction Description		
Get the current axis pulse equivalent of the control card. It is suitable for axes based on pulse equivalent programming mode.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
equiv	Not null	Axis pulse equivalent
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	

Related Instructions	adt_set_unit_mode、adt_get_unit_mode、adt_set_pulse_equiv
Related Chapter	Pulse equivalent
Related Routine	

adt_set_stop0_mode

Instruction Format		
int _stdcall adt_set_stop0_mode(int cardno, int axis, int enable, int logic, int admode)		
Instruction Description		
Set the current axis machine home signal (STOP0) mode of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
enable	[0,1]	Whether the home signal is enabled 0: Disable 1: Enable
logic	[0,1]	Home signal active level 0: Active low 1: Active high
admode	[0,1]	Trigger stop mode 0: Deceleration 1: Immediate
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_get_stop0_mode	
Related Chapter	Machine home signal (STOP0)	
Related Routine	Routine 5-1 Configuration Method Routines	

adt_get_stop0_mode

Instruction Format
int _stdcall adt_get_stop0_mode(int cardno, int axis, int *enable, int *logic, int *admode)
Instruction Description
Get the current axis machine home signal (STOP0) mode of the control card
Parameter Description

Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
enable	Not null	Whether the home signal is enabled 0: Disable 1: Enable
logic	Not null	Home signal active level 0: Active low 1: Active high
admode	Not null	Trigger stop mode 0: Deceleration 1: Immediate
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_stop0_mode	
Related Chapter	Machine home signal (STOP0)	
Related Routine		

adt_set_stop1_mode

Instruction Format		
int _stdcall adt_set_stop1_mode(int cardno, int axis, int enable, int logic, int admode)		
Instruction Description		
Set the current axis encoder Z-phase signal (STOP1) mode of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
enable	[0,1]	Whether the encoder Z phase signal is enabled 0: Disable 1: Enable
logic	[0,1]	Encoder Z phase signal active level 0: Active low 1: Active high
admode	[0,1]	Trigger stop mode 0: Deceleration 1: Immediate
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_get_stop1_mode	
Related Chapter	Encoder Z-phase signal (STOP1)	
Related Routine	Routine 5-1 Configuration Method Routines	

adt_get_stop1_mode

Instruction Format		
int _stdcall adt_get_stop1_mode(int cardno, int axis, int *enable, int *logic, int *admode)		
Instruction Description		
Get the current axis encoder Z-phase signal (STOP1) mode of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number

enable	Not null	Whether the encoder Z phase signal is enabled 0: Disable 1: Enable
logic	Not null	Encoder Z phase signal active level 0: Active low 1: Active high
admode	Not null	Trigger stop mode 0: Deceleration 1: Immediate

Returned Value
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure

Instruction Classification	Standard instruction Resource configuration class
Related Instructions	adt_set_stop1_mode
Related Chapter	Encoder Z-phase signal (STOP1)
Related Routine	

adt_set_limit_mode

Instruction Format
int _stdcall adt_set_limit_mode(int cardno, int axis, int pel_enable, int mel_enable, int logic)
Instruction Description
Set the current axis hardware limit signal mode of the control card
Parameter Description

Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pel_enable	[0,1]	Whether the hardware positive limit is enabled 0: Disable 1: Enable
mel_enable	[0,1]	Whether the hardware negative limit is enabled 0: Disable 1: Enable
logic	[0,1]	Active level 0: Active low 1: Active high

Returned Value
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure

Instruction Classification	Standard instruction Resource configuration class
Related Instructions	adt_get_limit_mode
Related Chapter	Hardware limit signal (LMT+/LMT-)
Related Routine	Routine 5-1 Configuration Method Routines

adt_get_limit_mode

Instruction Format		
int _stdcall adt_get_limit_mode(int cardno, int axis, int *pel_enable, int *mel_enable, int *logic)		
Instruction Description		
Get the current axis hardware limit signal mode of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pel_enable	Not null	Whether the hardware positive limit is enabled 0: Disable 1: Enable
mel_enable	Not null	Whether the hardware negative limit is enabled 0: Disable 1: Enable
logic	Not null	Active level 0: Active low 1: Active high
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_limit_mode	
Related Chapter	Hardware limit signal (LMT+/LMT-)	
Related Routine		

adt_set_axis_io_map

Instruction Format		
int _stdcall adt_set_axis_io_map(int cardno, int axis, int mode, int board, int port)		
Instruction Description		
Custom hardware signal mapping of available axis of the control card. Normally, the default configuration is adopted, which is subject to silkscreen.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
mode	[0-4]	Hardware signal identification 0: Positive limit 1: Negative limit 2: Home signal (STOP0) 3: Encoder Z-phase signal (STOP1) 4: Hardware stop signal
IoBoard	0	Alternate
port	[0-59]	New mapping port number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions		
Related Chapter		
Related Routine		

adt_get_axis_io_map

Instruction Format		
int _stdcall adt_get_axis_io_map(int cardno, int axis, int mode, int *board, int *port)		
Instruction Description		
Get hardware signal mapping of available axis of the control card. Normally, the default configuration is retained, which is subject to silkscreen.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
mode	[0-4]	Hardware signal identification 0: Positive limit 1: Negative limit 2: Home signal (STOP0) 3: Encoder Z-phase signal (STOP1) 4: Hardware stop signal
board	Not null	Alternate
port	Not null	mapping port number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions		
Related Chapter		
Related Routine		

adt_set_softlimit_mode

Instruction Format		
int _stdcall adt_set_softlimit_mode(int cardno, int axis, int enable, double Ppos, double Npos, int admode)		
Instruction Description		
Set the working mode of software limit function of current axis		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
enable	[0, 1]	Whether the software limit is enabled 0: Disable 1: Enable
Ppos		Software positive limit position
Npos		Software negative limit position
admode	[0, 1]	Trigger stop mode 0: Deceleration 1: Immediate
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_get_softlimit_mode	
Related Chapter	Software limit	
Related Routine	Routine 5-1 Configuration Method Routines	

adt_get_softlimit_mode

Instruction Format		
int _stdcall adt_get_softlimit_mode(int cardno, int axis, int *enable, double *Ppos, double *Npos, int *admode)		
Instruction Description		
Get the working mode of software limit function of current axis		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
enable	Not null	Whether the software limit is enabled 0: Disable 1: Enable
Ppos	Not null	Software positive limit position
Npos	Not null	Software negative limit position
admode	Not null	Trigger stop mode 0: Deceleration 1: Immediate
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_softlimit_mode	
Related Chapter	Software limit	
Related Routine		

adt_set_pos_variable_loop

Instruction Format		
int _stdcall adt_set_pos_variable_loop(int cardno, int axis, int enable, long CompPos)		
Instruction Description		
Set the variable loop function of logic position		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
enable	[0,1]	Whether the logic position variable loop function is enabled 0: Disable 1: Enable
CompPos	[0,∞)	Variable loop position
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_get_pos_variable_loop	
Related Chapter	Logic variable loop mode	
Related Routine	Routine 5-1 Configuration Method Routines	

adt_get_pos_variable_loop

Instruction Format		
int _stdcall adt_get_pos_variable_loop(int cardno, int axis, int *enable, long * CompPos)		
Instruction Description		
Get the variable loop function of logic position		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
enable	Not null	Whether the logic position variable loop function is enabled

		0: Disable 1: Enable
CompPos	Not null	Variable loop position
返回值 Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions	adt_set_pos_variable_loop	
Related Chapter	Logic variable loop mode	
Related Routine		

adt_get_axis_io_status

Instruction Format		
int _stdcall adt_get_axis_io_status (int cardno, int axis, int * status)		
Instruction Description		
Gets the status of axis binding IO signal		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
status	Not null	B0:Positive limit signal; B1:Negative limit signal; B2:The origin signal(stop0); B3:Encoder Z-phase signal (STOP1); B4:Hardware stop signal(EMG); B5:Servo alarm signal; B6:S-ON;
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions		

Related Chapter	
Related Routine	

adt_set_axis_alarm_mode

Instruction Format		
int _stdcall adt_set_axis_alarm_mode (int cardno, int axis, int enable, int logic)		
Instruction Description		
Set axis alarm mode		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
enable	[0,1]	Axis alarm is enabled 0: Disable 1: Enable
logic	[0,1]	Active level 0:Active low 1:Active high
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions		
Related Chapter		
Related Routine		

adt_set_limit_lock

Instruction Format		
int _stdcall adt_set_limit_lock (int cardno, int enable)		
Instruction Description		
Hardware limit lock enable		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
enable	[0,1]	Hardware limit lock is enabled

		0: Disable 1: Enable
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Resource configuration class	
Related Instructions		
Related Chapter		
Related Routine		

17.4. Speed Planning Class

adt_set_startv

Instruction Format		
int _stdcall adt_set_startv(int cardno, int axis, double speed)		
Instruction Description		
Set the current axis drive start speed of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
speed	Pulse equivalent programming mode: speed*equiv<5M Pulse programming mode: speed<5M equiv -- Pulse equivalent	Drive start speed
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
elated Instructions	adt_get_startv	
Related Chapter	Basic speed planning	
Related Routine	Routine 6-1 Basic Speed Planning and Quantitative Driving	

adt_get_startv

Instruction Format		
int _stdcall adt_get_startv(int cardno, int axis, double *speed)		
Instruction Description		
Get the current axis drive start speed of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
speed	Not null	Drive start speed
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_set_startv	
Related Chapter	Basic speed planning	
Related Routine		

adt_set_endv

Instruction Format		
int _stdcall adt_set_endv(int cardno, int axis, double speed)		
Instruction Description		
Set the current axis drive end speed of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
speed	Pulse equivalent	Drive end speed

	programming mode: speed*equiv<5M Pulse programming mode: speed<5M equiv -- Pulse equivalent	

Returned Value	
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction Speed planning class
Related Instructions	adt_get_endv
Related Chapter	Basic speed planning
Related Routine	Routine 6-1 Basic Speed Planning and Quantitative Driving

adt_get_endv

Instruction Format		
int _stdcall adt_get_endv(int cardno, int axis, double *speed)		
Instruction Description		
Get the current axis drive end speed of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
speed	Not null	Drive end speed
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_set_endv	
Related Chapter	Basic speed planning	
Related Routine		

adt_set_maxv

Instruction Format		
int _stdcall adt_set_maxv(int cardno, int axis, double speed)		
Instruction Description		
Set the maximum drive speed of the current axis of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
speed	Pulse equivalent programming mode: speed*equiv<5M Pulse programming mode: speed<5M equiv -- Pulse equivalent	Maximum drive speed
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_get_maxv	
Related Chapter	Basic speed planning	
Related Routine	Routine 6-1 Basic Speed Planning and Quantitative Driving	

adt_get_maxv

Instruction Format		
int _stdcall adt_get_maxv(int cardno, int axis, double *speed)		
Instruction Description		
Get the maximum drive speed of the current axis of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
speed	Not null	Maximum drive speed
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_set_maxv	
Related Chapter	Basic speed planning	
Related Routine		

adt_set_acc

Instruction Format		
int _stdcall adt_set_acc(int cardno, int axis, double acc)		
Instruction Description		
Set the current axis drive acceleration of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis

acc	Pulse equivalent programming mode: acc*equiv<100M Pulse programming mode: acc<100M equiv -- Pulse equivalent	Drive acceleration
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_get_acc	
Related Chapter	Basic speed planning	
Related Routine	Routine 6-1 Basic Speed Planning and Quantitative Driving	

adt_get_acc

Instruction Format		
int _stdcall adt_get_acc(int cardno, int axis, double *acc)		
Instruction Description		
Get the current axis drive acceleration of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
acc	Not null	Drive acceleration
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_set_acc	
Related Chapter	Basic speed planning	
Related Routine		

adt_set_dec

Instruction Format		
int _stdcall adt_set_dec(int cardno, int axis, double dec)		
Instruction Description		
Set the current axis drive deceleration of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
dec	Pulse equivalent programming mode: dec*equiv<100M Pulse programming mode: dec<100M equiv -- Pulse equivalent	Drive deceleration
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_get_dec	
Related Chapter	Basic speed planning	
Related Routine	Routine 6-1 Basic Speed Planning and Quantitative Driving	

adt_get_dec

Instruction Format		
int _stdcall adt_get_dec(int cardno, int axis, double *dec)		
Instruction Description		
Get the current axis drive deceleration of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
dec	Not null	Drive deceleration
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_set_dec	
Related Chapter	Basic speed planning	
Related Routine		

adt_set_admode

Instruction Format		
int _stdcall adt_set_admode(int cardno, int axis, int mode)		
Instruction Description		
Set the current axis acceleration/deceleration mode of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
mode	[0-3]	Axis acceleration/deceleration mode

		0: S type 1: T-type 2: EXP (exponential) type 3: COS (cosine) type
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions		
Related Chapter	Basic speed planning	
Related Routine	Routine 6-1 Basic Speed Planning and Quantitative Driving	

adt_get_admode

Instruction Format		
int _stdcall adt_get_admode(int cardno, int axis, int *mode)		
Instruction Description		
Get the current axis acceleration/deceleration mode of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number ,[1-8]: Single axis [62,63]: Interpolation axis
mode	Not null	axis acceleration/deceleration mode
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_set_admode	
Related Chapter	Basic speed planning	
Related Routine	Routine 6-1 Basic Speed Planning and Quantitative Driving	

adt_set_rate

Instruction Format

int _stdcall adt_set_rate(int cardno, int axis, double rate)		
Instruction Description		
Set the speed ratio of current axis of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
rate	[0-2.0]	Drive speed ratio
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions	adt_get_rate	
Related Chapter	Speed ratio	
Related Routine	Routine 6-4 Speed Ratio	

adt_get_rate

Instruction Format		
int _stdcall adt_get_rate(int cardno, int axis, double *rate)		
Instruction Description		
Get the speed ratio of current axis of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
rate	Not null	Drive speed ratio
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard		

Error Code for the reason of failure	
Instruction Classification	Standard instruction Speed planning class
Related Instructions	adt_set_rate
Related Chapter	Speed ratio
Related Routine	

adt_get_speed

Instruction Format		
int _stdcall adt_get_speed(int cardno, int axis, double *speed)		
Instruction Description		
Get the current drive speed of the current axis of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
speed	Not null	Current drive speed
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions		
Related Chapter	Get current drive speed	
Related Routine		

adt_set_corner_speed_smooth_level

Instruction Format		
int _stdcall adt_set_corner_speed_smooth_level(int cardno, int axis, int level)		
Instruction Description		
Set the current axis corner acceleration smoothing level of the control card		
Parameter Description		
Name	Range	Description

cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
level	[1-100]	Axis corner speed smoothing factor, default 50
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions		
Related Chapter	Corner acceleration smoothing level	
Related Routine		

adt_set_arc_speed_clamp_unit

Instruction Format		
int _stdcall adt_set_arc_speed_clamp_unit(int cardno, int Inpaxis, double radius, double speed)		
Instruction Description		
Set the interpolation axis corner arc speed constraint of the control card		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
Inpaxis	[62,63]	Interpolation axis number
radius	[0.001, ∞)	Radius factor, default 10mm
speed	[0.01, 5000]	Speed factor, default 100mm/s
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions		
Related Chapter	Arc interpolation speed constraint	
Related Routine	Routine 6-3 Arc Interpolation Speed Constraint	

adt_set_speed_pretreat_mode

Instruction Format		
int _stdcall adt_set_speed_pretreat_mode(int cardno, int Inpaxis, int enable)		
Instruction Description		
Set the speed look-ahead function for the interpolation axis of the control card to enable the cache interpolation mode.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
Inpaxis	[62,63]	Interpolation axis number
enable	[0,1]	Speed look-ahead function identification 0: Disable 1: Enable
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Speed planning class	
Related Instructions		
Related Chapter	Cache interpolation	
Related Routine	Routine 6-2 Speed Look-ahead, Routine 7-5 Cache Interpolation	

17.5. Position Control Class

adt_set_command_pos

Instruction Format		
int _stdcall adt_set_command_pos(int cardno, int axis, long pos)		
Instruction Description		
Set the current logic position of specified axis of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos		Logic position value
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position control class	
Related Instructions	adt_get_command_pos	
Related Chapter	Position control	
Related Routine	Routine 9-1 Position Control	

adt_get_command_pos

Instruction Format		
int _stdcall adt_get_command_pos(int cardno, int axis, long *pos)		
Instruction Description		
Get the current logic position of specified axis of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos	Not null	Logic position value
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		

Instruction Classification	Standard instruction Position control class
Related Instructions	adt_set_command_pos
Related Chapter	Position control
Related Routine	Routine 9-1 Position Control

adt_set_actual_pos

Instruction Format		
int _stdcall adt_set_actual_pos(int cardno, int axis, long pos)		
Instruction Description		
Set the actual position of current encoder of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos		Encoder position value
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position control class	
Related Instructions	adt_get_actual_pos	
Related Chapter	Position control	
Related Routine	Routine 9-1 Position Control	

adt_get_actual_pos

Instruction Format		
int _stdcall adt_get_actual_pos(int cardno, int axis, long *pos)		
Instruction Description		
Get the actual position of current encoder of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos	Not null	Encoder position value

Returned Value	
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction Position control class
Related Instructions	adt_set_actual_pos
Related Chapter	Position control
elated Routine	Routine 9-1 Position Control

adt_get_target_pos_unit

Instruction Format		
int _stdcall adt_get_target_pos_unit(int cardno, int axis, double *pos)		
Instruction Description		
Get the target position of specified axis of the control card (unit: mm); suitable for axes based on pulse equivalent programming mode.		
参数说明 Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos	Not null	Axis target position value (mm)
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position control class	
Related Instructions		
Related Chapter	Position control	
Related Routine	Routine 9-1 Position Control	

adt_get_target_pos_pulse

Instruction Format
int _stdcall adt_get_target_pos_pulse(int cardno, int axis, long *pos)
Instruction Description
Get the target position of specified axis of the control card (unit: pulse); suitable for axes based

on pulse programming mode.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos	Not null	Axis target position value (pulse)
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position control class	
Related Instructions		
Related Chapter	Position control	
Related Routine	Routine 9-1 Position Control	

17.6. Drive control class

adt_pmove_unit

Instruction Format		
int _stdcall adt_pmove_unit(int cardno, int axis, double pos, int PosType)		
Instruction Description		
Single-axis quantitative drive based on pulse-equivalent programming mode		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos		Target position (mm)
PosType	[0, 1]	Axis position mode 0: Relative position coordinates 1: Absolute position coordinates
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions	adt_pmove_pulse	
Related Chapter	Quantitative drive	
Related Routine	Routine 7-1 Quantitative Drive	

adt_pmove_pulse

Instruction Format		
int _stdcall adt_pmove_pulse(int cardno, int axis, long pos, int PosType)		
Instruction Description		
Single-axis quantitative drive based on pulse-equivalent mode		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos		Target position (pulse)

PosType	[0, 1]	Axis position mode 0: Relative position coordinates 1: Absolute position coordinates
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions	adt_pmove_unit	
Related Chapter	Quantitative drive	
Related Routine	Routine 7-1 Quantitative Drive	

adt_continue_move

Instruction Format		
int _stdcall adt_continue_move(int cardno, int axis, int dir)		
Instruction Description		
Single axis continuous drive		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
dir	[0, 1]	Driving direction 0: Positive 1: Negative
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter	Continuous drive	
Related Routine	Routine 7-2 Continuous Drive	

adt_inp_move_unit

Instruction Format
int _stdcall adt_inp_move_unit(int cardno, int InpAxis, int index, int AxisNum, int AxisList[], double PosList[], int PosType)

Instruction Description		
Cacheable multi-axis linear interpolation based on pulse equivalent programming mode		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	0	Reserved
AxisNum	[1-8]	Number of axes participating in linear interpolation
AxisList	Not null	List of axis numbers participating in linear interpolation, the array size must be greater than or equal to AxisNum
PosList	Not null	List of axis target positions participating in linear interpolation, the array size must be greater than or equal to AxisNum
PosType	[0, 1]	Axis position mode 0: Relative position coordinates 1: Absolute position coordinates
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions	adt_inp_move_pulse	
Related Chapter	Linear interpolation	
Related Routine	Routine 7-3 Linear Interpolation	

adt_inp_move_pulse

Instruction Format
int _stdcall adt_inp_move_pulse(int cardno, int InpAxis, int index, int AxisNum, int AxisList[], long PosList[], int mode)
Instruction Description
Cacheable multi-axis linear interpolation based on pulse programming mode

Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	0	Reserved
AxisNum	[1-8]	Number of axes participating in linear interpolation
AxisList	Not null	List of axis numbers participating in linear interpolation, the array size must be greater than or equal to AxisNum
PosList	Not null	List of axis target positions participating in linear interpolation, the array size must be greater than or equal to AxisNum
mode	[0, 1]	Axis position mode 0: Relative position coordinates 1: Absolute position coordinates
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions	adt_inp_move_unit	
Related Chapter	Linear interpolation	
Related Routine	Routine 7-3 Linear Interpolation	

adt_inp_arc2_unit

指令格式 Instruction Format
int _stdcall adt_inp_arc2_unit(int cardno, int InpAxis, int index, int AxisList[2], double TargetList [2], double CenterList [2], int dir, int PosType)
指令描述 Instruction Description
Cacheable planar circular interpolation based on pulse equivalent programming mode
Parameter Description

Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	0	Reserved
AxisList	Not null	List of axis numbers participating in plane arc interpolation, the array size must be greater than or equal to 2
TargetList	Not null	List of axis target positions participating in plane arc interpolation, the array size must be greater than or equal to 2
CenterList	Not null	List of axis center positions participating in plane arc interpolation, the array size must be greater than or equal to 2
dir	[0,1]	Plane arc interpolation direction 0: CCW 1: CW
PosType	[0,1]	Axis position mode 0: Relative position coordinates 1: Absolute position coordinates
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter	Plane arc interpolation	
Related Routine	Routine 7-4 Plane Arc Interpolation	

adt_inp_arc3_unit

Instruction Format
int _stdcall adt_inp_arc3_unit(int cardno, int InpAxis, int index, int AxisList[3], double Pos2List [3], double Pos3List[3], int ArcFlag, int PosType)
Instruction Description
Cacheable spherical circular interpolation based on pulse equivalent programming mode
Parameter Description

Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	0	Reserved
AxisList	Not null	List of axis numbers participating in spherical arc interpolation, the array size must be greater than or equal to 3
Pos2List	Not null	List of second point axis target positions on the arc participating in spherical arc interpolation, the array size must be greater than or equal to 3
Pos3List	Not null	List of third point axis target positions on the arc participating in spherical arc interpolation, the array size must be greater than or equal to 3
ArcFlag	[0,1]	Spherical arc type 0: Semicircle 1: Full circle
PosType	[0,1]	Axis position mode 0: Relative position coordinates 1: Absolute position coordinates

Returned Value	
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction Drive control class
Related Instructions	
Related Chapter	Spherical arc interpolation
Related Routine	

adt_get_axis_status

Instruction Format
int _stdcall adt_get_axis_status(int cardno, int axis, int *status)
Instruction Description
Get the drive status of specified axis of the control card, either a solid axis or an interpolation axis. If the speed ratio of current axis is 0 , the drive status of current axis is 0;
Parameter Description

Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8],[62,63]	Axis number [1-8]: Single axis [62,63]: Interpolation axis
status	Not null	Axis drive status
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter		
Related Routine		

adt_get_all_axis_move_status

Instruction Format
int _stdcall adt_get_all_axis_move_status (int cardno, unsigned int *status)
Instruction Description
Get the drive status of all axis on the control card. If the speed ratio of current axis is 0 , the drive status of current axis is 0.
Parameter Description

Name	Range	Description
cardno	[0-9]	Control card index number
status	Not null	Axis drive status bit0-7 : Axis 1-8; 0: Axis stop 1: Axis movement
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		

Related Chapter	
Related Routine	

adt_get_all_axis_reach_status

Instruction Format		
int _stdcall adt_get_all_axis_reach_status (int cardno, unsigned int *status)		
Instruction Description		
Get the reach status of all axis on the control card. If the speed ratio of current axis is 0 , the reach status of current axis is 1.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
status	Not null	Axis drive status bit0-7 : Axis 1-8; 0: Axis has reached 1: Axis movement
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter		
Related Routine		

adt_get_stopdata

Instruction Format		
int _stdcall adt_get_stopdata(int cardno, int axis, int *value)		
Instruction Description		
Get the stop information of specified axis of the control card.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number

value	Not null	Axis stop information
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter	Drive status detection, drive stop and stop information	
Related Routine	Routine 7-7 Drive Status and Stop Information	

adt_set_axis_stop

Instruction Format		
int _stdcall adt_set_axis_stop(int cardno, int axis, int admode)		
Instruction Description		
Stop drive of specified axis, either a solid axis or an interpolation axis, or specify the stop mode.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[0-8],[62,63]	Axis number 0: All axes [1-8]: Single axis [62,63]: Interpolation axis
admode	[0,1]	Axis stop mode 0: Deceleration 1: Immediate
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter	Drive status detection, drive stop and stop information	
Related Routine	Routine 7-7 Drive Status and Stop Information	

adt_set_follow_axis

Instruction Format		
int _stdcall adt_set_follow_axis(int cardno, int SlaveAxis, int MasterAxis)		
Instruction Description		
Set the axis following drive mode (open loop gantry double drive)		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
SlaveAxis	[1-8]	Slave axis number
MasterAxis	[0-8]	Principal axis number, 0 means to cancel follow
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter	Gantry double drive	
Related Routine	Routine 7-6 Gantry Double Drive	

adt_get_inp_fifo_len

Instruction Format		
int _stdcall adt_get_inp_fifo_len(int cardno, int InpAxis, int *len)		
Instruction Description		
Get the cache interpolation margin of specified interpolation axis		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
len	Not null	Cache interpolation margin
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		

Instruction Classification	Standard instruction Drive control class
Related Instructions	
Related Chapter	Cache interpolation
Related Routine	Routine 7-5 Cache Interpolation

adt_get_inp_index

Instruction Format		
int _stdcall adt_get_inp_index(int cardno, int InpAxis, int *index)		
Instruction Description		
Get the index of latest triggered cache interpolation instruction		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
Index	Not null	Index of latest triggered cache interpolation instruction
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter		
Related Routine		

adt_change_pmove_pos_unit

Instruction Format		
int _stdcall adt_change_pmove_pos_unit (int cardno, int axis, double TargetPos)		
Instruction Description		
Get the index of latest triggered cache interpolation instruction		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number

TargetPos	Not null	Target position(absolute position), unit: mm
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Drive control class	
Related Instructions		
Related Chapter		
Related Routine		

17.7. Input/Output Control Class

adt_write_outport

Instruction Format		
int _stdcall adt_write_outport(int cardno, int group, unsigned long outmap)		
Instruction Description		
Control output port by group		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
group	[1-3]	Group number 1:(OUT0-15) 2:(OUT16-31) 3:(OUT32-41)
outmap	[0, 65535]	Control level of output port group, 16-bit binary number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Input and output control class	
Related Instructions	adt_read_outport	
Related Chapter	Output control	
Related Routine	Routine 10-1 Output Port Control	

adt_read_outport

Instruction Format		
int _stdcall adt_read_outport(int cardno, int group, unsigned long * outmap)		
Instruction Description		
Read status of control output port by group		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
group	[1, 3]	Group number 1:(OUT0-15)

		2:(OUT16-31) 3:(OUT32-41)
outmap	Not null	Control level of output port group, 16-bit binary number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Input and output control class	
Related Instructions	adt_write_outport	
Related Chapter	Output control	
Related Routine	Routine 10-1 Output Port Control	

adt_write_outbit

Instruction Format		
int _stdcall adt_write_outbit(int cardno, int index, int value)		
Instruction Description		
Control a single output port		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
index	[0-41]	Output port number
value	[0, 1]	Output port level status 1: Low 0: High
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Input and output control class	
Related Instructions	adt_read_outbit	
Related Chapter	Output control	
Related Routine	Routine 10-1 Output Port Control	

adt_read_outbit

Instruction Format
int _stdcall adt_read_outbit(int cardno, int index, int * value)

Instruction Description		
Read status of a single output port		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
index	[0-41]	Output port number
value	Not null	Output port level status 1: Low 0: High
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Input and output control class	
Related Instructions	adt_write_outbit	
Related Chapter	Output control	
Related Routine	Routine 10-1 Output Port Control	

adt_read_inport

Instruction Format		
int _stdcall adt_read_inport(int cardno, int group, unsigned long * inmap)		
Instruction Description		
Read status of control input port by group		
Parameter Description		
Name	Range	Description
cardno	[0-9]	控制卡索引号 Control card index number
group	[1-5]	Group number 1:(INT0-15) 2:(IN16-31) 3:(IN32-47) 4: (IN48-59) 50-59 is the wheel IO input 5: (IN60-67) is 1-8 axis Z phase signal, (IN68-75)is 1-8 axis A phase signal, (IN76-83)is 1-8 axis B phase signal

inmap	Not null	Control level of output port group, 16-bit binary number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Input and output control class	
Related Instructions		
Related Chapter	Input control	
Related Routine	Routine 10-2 Input Port Control	

adt_read_inbit

Instruction Format		
int _stdcall adt_read_inbit(int cardno, int index, int *status)		
Instruction Description		
Read status of a single input port		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
index	[0-67]	Input port number
status	Not null	Input port level status 1: Low 0: High
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Input and output control class	
Related Instructions		
Related Chapter	Input control	
Related Routine	Routine 10-2 Input Port Control	

adt_set_input_filter

Instruction Format
int _stdcall adt_set_input_filter(int cardno, int type, int grade)
Instruction Description
Set input filter function to distinguish between the quick input port and the general purpose

input port.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
type	[0, 1]	Input port type 0: Quick input (limit, home, emergency stop) 1: Extended general purpose input
grade	[0-18] when type=0 [0-15] when type=1	Filter level
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Input and output control class	
Related Instructions		
Related Chapter	Input filter	
Related Routine	Routine 10-3 Input Filter Control	

17.8. Cache Event Control Class

adt_set_fifo_outbit

Instruction Format		
int _stdcall adt_set_fifo_outbit(int cardno, int InpAxis, int index, int port, int value, double speed = -1)		
Instruction Description		
Cache output control (based on pulse equivalent programming)		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	[0-999]	Cache event index
port	[16-41]	Cache control port number
value	[0, 1]	Cache control level 0: Low 1: High
speed		Speed constraint during cache output control, -1 means no constraint, Range (0.0 – 5000 mm/sec)
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		
Related Chapter	Cache output control	
Related Routine	Routine 11-1 Cache Output Control	

adt_set_fifo_pwm

Instruction Format
int _stdcall adt_set_fifo_pwm(int cardno, int InpAxis, int index, int port, int enable, double time_h, double time_l, double speed = -1)
Instruction Description
Cache high precision PWM output control
Parameter Description

Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	[0,999]	Cache event index
port	[16,17]	Cache control port number
enable	[0, 1]	Cache PWM control enable 0: Disable 1: Output
time_h	[0-10000]	Output port high level hold time, ms (ms)
Time_l	[0-10000]	Output port low level hold time, ms (ms)
speed		Speed constraint during cache output control, -1 means no constraint, Range (0.0 - 100000.0 mm/sec)
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		
Related Chapter	Cache PWM control	
Related Routine	Routine 11-2 Cache PWM Control	

adt_set_pwm_output

Instruction Format		
int _stdcall adt_set_pwm_output (int cardno, int port, int enable, double time_h, double time_l, int count= 0)		
Instruction Description		
Output PWM pulse with OUT16, 17 (non-cached, output immediately after setting)		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
port	[16,17]	Cache control port number
enable	[0, 1]	Cache PWM control enable 0: Disable 1: Enable
time_h	[0-10000]	Output port high level hold time, unit (ms)
Time_l	[0-10000]	Output port low level hold time, unit (ms)

count		Number of PWM pulse output，Max:1000; The default parameter 0 is to keep the output, not limit the number;
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		
Related Chapter		
Related Routine		

adt_get_pwm_output

Instruction Format		
int _stdcall adt_get_pwm_output (int cardno, int port, int *enable, double *time_h, double *time_l, int *count)		
Instruction Description		
Output PWM pulse with OUT16, 17 (non-cached, output immediately after setting)		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
port	[16,17]	Cache control port number
enable	[0, 1]	Cache PWM control enable 0: Disable 1: Enable
time_h	[0-10000]	Output port high level hold time, unit (ms)
Time_l	[0-10000]	Output port low level hold time, unit (ms)
count		Number of PWM pulse output，Max:1000; The default parameter 0 is to keep the output, not limit the number;
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		

Related Chapter	
Related Routine	

adt_set_fifo_delay

Instruction Format		
int _stdcall adt_set_fifo_delay(int cardno, int InpAxis, int index, int millisecond)		
Instruction Description		
Cache delay control (based on pulse equivalent programming)		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	[0-999]	Cache event index
millisecond	[0,∞)	Cache delay time, unit:ms
返回值 Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		
Related Chapter	Cache Delay control	
Related Routine		

adt_clear_fifo_event

Instruction Format		
int _stdcall adt_clear_fifo_event(int cardno, int InpAxis)		
Instruction Description		
Clear cache event		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
Returned Value		

0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction Cache event control class
Related Instructions	
Related Chapter	Cache event
Related Routine	

adt_get_fifo_event_len

Instruction Format		
int _stdcall adt_get_fifo_event_len(int cardno, int InpAxis, int *len)		
Instruction Description		
Get the available cache event margin		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
len	Not null	Available cache event margin
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		
Related Chapter	Cache event	
Related Routine		

adt_get_fifo_event_index

Instruction Format		
int _stdcall adt_get_fifo_event_index (int cardno, int InpAxis, int *index)		
Instruction Description		
Get the index of latest triggered cache event instruction		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number

InpAxis	[62,63]	Interpolation axis number
index	Not null	Index of latest triggered cache event instruction
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		
Related Chapter	Cache event	
Related Routine		

adt_set_fifo_multi_io

Instruction Format		
int _stdcall adt_set_fifo_multi_io (int cardno, int InpAxis, int index, int gp, int iomap, int levelmap, double speed		
Instruction Description		
Get the index of latest triggered cache event instruction		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	Not null	Cache event index
gp	[1,3]	group number, 1(0-15); 2(16-31); 3(32-41);
iomap	[0 – 65535]	(bit0 ~ bit15)Specifies the output port to operate on; A bit value of 1 operates on the corresponding output port; The output port with a bit value of 0 is not affected;
levelmap	[0,1]	Set the status of the group output port; According to a map; 0:off; 1:on;
speed	[0.0 - 100000.0]	Speed constraint during cache output control, -1 means no constraint, unit:

		mm/sec
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		
Related Chapter	Cache event	
Related Routine		

adt_set_fifo_pwm_count

Instruction Format		
int _stdcall adt_set_fifo_pwm_count (int cardno, int InpAxis, int port, int gp, int time_h, int time_l, int count, double speed)		
Instruction Description		
Get the index of latest triggered cache event instruction		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
InpAxis	[62,63]	Interpolation axis number
index	Not null	Cache event index
port	[1,3]	group number, 1(0-15); 2(16-31); 3(32-41);
time_h	[0-1000]	Output port high level hold time, unit(ms)
time_l	[0-1000]	Output port low level hold time, unit(ms)
count	[0-1000]	Number of pulse output
speed	[0.0 - 100000.0]	Speed constraint during cache output control, -1 means no constraint, unit: mm/sec
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Cache event control class	
Related Instructions		

Related Chapter	Cache event
Related Routine	

17.9. Position comparison control class

adt_get_compare_len

Instruction Format		
int _stdcall adt_get_compare_len (int cardno, int cmp, int *len)		
Instruction Description		
Get the one-dimensional position comparator margin		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification, 0 corresponds to OUT16, 1 corresponds to OUT17
len		One-dimensional position comparator margin
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions		
Related Chapter	Position Comparison	
Related Routine		

adt_get_compare_status

Instruction Format		
int _stdcall adt_get_compare_status (int cardno, int cmp, int *status)		
Instruction Description		
Get the number of comparison points that have been completed in one-dimensional position		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17

status		Number of comparison points that have been triggered
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions		
Related Chapter	Position Comparison	
Related Routine		

adt_clear_compare_point

Instruction Format		
int _stdcall adt_clear_compare_point (int cardno, int cmp)		
Instruction Description		
Clear all comparison points of the one-dimensional position comparator and reset the position comparator status.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions		
Related Chapter	Position Comparison	
Related Routine		

adt_set_compare_mode

Instruction Format
int _stdcall adt_set_compare_mode (int cardno, int cmp, int enable, int axis, int source, int

mode, int level, double time)		
Instruction Description		
Set one-dimensional position comparator mode		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
enable	[0,1]	Cache one-dimensional position comparator control enable 0: Disable 1: Enable
axis	[1-8]	Position comparison axis number
source	[0,1]	Position comparison data source 0: Logic position 1: Actual position
mode	[0,1]	Output port level control mode 0: Level inversion 1: Pulse width output
level	[0,1]	Normal level 0: Low 1: High
time	[0.1 - 6500]	Pulse width duration, unit:ms
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions	adt_get_compare_mode	
Related Chapter	Position Comparison	
Related Routine		

adt_get_compare_mode

Instruction Format
int _stdcall adt_get_compare_mode (int cardno, int cmp, int *axis, int *source, int *mode, int *level, double *time)
Instruction Description
Get one-dimensional position comparator mode

Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
axis	Not null	Position comparison axis number
source	Not null	Position comparison data source 0: Logic position 1: Actual position
mode	Not null	Output port level control mode 0: Level inversion 1: Pulse width output
level	Not null	Normal level 0: Low 1: High
time	Not null	Pulse width duration
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions	adt_set_compare_mode	
Related Chapter	Position Comparison	
Related Routine		

adt_add_compare_point

Instruction Format		
int _stdcall adt_add_compare_point(int cardno, int cmp, long pos)		
Instruction Description		
Add comparison position point of the one-dimensional position comparator		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17

pos		One-dimensional position comparison point
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions		
Related Chapter	Chapter 12 One-dimensional Position Comparison	
Related Routine		

adt_add_compare_table

Instruction Format		
int _stdcall adt_add_compare_table(int cardno, int cmp, long *table, int table_len)		
Instruction Description		
Add the list of comparison positions for the one-dimensional position comparator.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
table	Not null	List of position points, must be monotonically increasing or decreasing; position mode is absolute position, unit:pulse
table_len	[1,999]	Data number of one-dimensional position comparison point
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions		
Related Chapter	Position Comparison	
Related Routine		

adt_add_compare_linear

Instruction Format		
int _stdcall adt_add_compare_linear(int cardno, int cmp, long startPos, int interval, int cmp_times)		
Instruction Description		
Add a set of comparison position points of one-dimensional position linearly		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
startPos	Not null	Linear start point of one-dimensional position comparison point, unit: pulse
interval	!=0	Spacing of linear comparison points, in pulse, either positive or negative
cmp_times	[1,999]	Number of linear comparison position points
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions		
Related Chapter	Position Comparison	
Related Routine		

adt_set_compare_xd_mode

Instruction Format
int _stdcall adt_set_compare_xd_mode (int cardno, int cmp, int enable, int xd, int axis[], int source, int mode, int level)
Instruction Description
Set position comparator mode
Parameter Description

Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
enable	[0,1]	Cache one-dimensional position comparator control enable 0: Disable 1: Enable
xd	[1-3]	Comparator dimension
axis		Comparator axis selection list
source	[0,1]	Position comparison data source 0: Logic position 1: Actual position
mode	[0,1]	Output port level control mode 0: Level inversion 1: Pulse width output
level	[0,1]	Normal level 0: Low 1: High

Returned Value	
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction position comparison control class
Related Instructions	adt_get_compare_xd_mode, adt_set_compare_xd_diffout, adt_set_compare_xd_area, adt_add_compare_xd_point, adt_add_compare_2d_linear
Related Chapter	
Related Routine	

adt_get_compare_xd_mode

Instruction Format		
int _stdcall adt_get_compare_xd_mode (int cardno, int cmp, int *enable, int *xd, int axis[], int *source, int *mode, int *level)		
Instruction Description		
Get position comparator mode		
Parameter Description		
Name	Range	Description

cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
enable	[0,1]	Cache one-dimensional position comparator control enable 0: Disable 1: Enable
xd	[1-3]	Comparator dimension
axis		Comparator axis selection list
source	[0,1]	Position comparison data source 0: Logic position 1: Actual position
mode	[0,1]	Output port level control mode 0: Level inversion 1: Pulse width output
level	[0,1]	Normal level 0: Low 1: High
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions	adt_set_compare_xd_mode, adt_set_compare_xd_diffout, adt_set_compare_xd_area, adt_add_compare_xd_point, adt_add_compare_2d_linear	
Related Chapter		
Related Routine		

adt_set_compare_xd_diffout

Instruction Format		
int _stdcall adt_set_compare_xd_diffout (int cardno, int dir, double freq, int count)		
Instruction Description		
Set position comparator，Pulse differential output parameter		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
dir	[0,1]	Differential output direction

		0: positive direction 1: negative direction
freq	[1-2000000]	impulse frequency, unit:HZ
count	[1-INTMAX]	The number of pulses
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions	adt_set_compare_xd_mode, adt_get_compare_xd_mode, adt_set_compare_xd_area, adt_add_compare_xd_point, adt_add_compare_2d_linear	
Related Chapter		
Related Routine		

adt_set_compare_xd_area

Instruction Format		
int _stdcall adt_set_compare_xd_area (int cardno, int cmp, long area[])		
Instruction Description		
Set the 2/3D comparison area range		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
area []		the 2/3D comparison area range，unit：pulse two-dimensional position comparator range， 0:dx; 1:dy; [x-dx, y+dy] [x+dx, y+dy] [x-dx, y-dy] [x+dx, y-dy] three-dimensional position comparator range， 0:dx; 1:dy; 2:dz
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		

Instruction Classification	Standard instruction position comparison control class
Related Instructions	adt_set_compare_xd_mode, adt_get_compare_xd_mode, adt_set_compare_xd_diffout, adt_add_compare_xd_point, adt_add_compare_2d_linear
Related Chapter	
Related Routine	

adt_add_compare_xd_point

Instruction Format		
int _stdcall adt_add_compare_xd_point (int cardno, int cmp, long pos[], double time_h=0, double time_l=0)		
Instruction Description		
Add comparison position point of the position comparator		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
pos []		position comparison point
time_h		PWM pulse output,OUT open time, unit(ms); mode=1; Pulse width output, OUT open time
time_l		PWM pulse output,OUT close time, unit(ms); Only mode=2, This parameter is valid
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions	adt_set_compare_xd_mode, adt_get_compare_xd_mode, adt_set_compare_xd_area ,adt_set_compare_xd_diffout, adt_add_compare_2d_linear	
Related Chapter		

Related Routine	

adt_add_compare_2d_linear

Instruction Format		
int _stdcall adt_add_compare_2d_linear (int cardno, int cmp, long startPos [2], double interval, int cmp_times, double time_h=0)		
Instruction Description		
Add comparison position points of two-dimensional position linearly		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
cmp	[0,1]	Port identification 0 corresponds to OUT16, 1 corresponds to OUT17
startPos [2]		Linear start point of one-dimensional position comparison point； Position mode is absolute position[0]:X_axis; [1]:Y_axis unit：pulse
interval	[0-65536]	Spacing of linear comparison points, in pulse, either positive or negative； unit:mm
cmp_times		Number of linear comparison position points;
time_h	[0-6553.5]	Pulse width output, OUT open time,unit(ms),precision(0.1ms), Level inversion，time_h = 0
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction position comparison control class	
Related Instructions	adt_set_compare_xd_mode, adt_get_compare_xd_mode, adt_set_compare_xd_area ,adt_set_compare_xd_diffout, adt_add_compare_xd_point	
Related Chapter		
Related Routine		

17.10. Position latch control class

adt_set_latch_mode

Instruction Format		
int _stdcall adt_set_latch_mode(int cardno, int axis, int mode, int logic)		
Instruction Description		
Set single axis position latch mode		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
mode	[0,3]	Position latch mode 0: Home signal (STOP0) single latch 1: Encoder Z phase signal (STOP1) single latch 2: Input port IN32 high speed continuous latch 3: Input port IN33high speed continuous latch
logic	[0,1]	Latch signal trigger level 0: Falling edge trigger (level changes from high to low) 1: Rising edge trigger (level changes from low to high)
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position latch control class	
Related Instructions	adt_get_latch_status, adt_get_latch_command_pos, adt_get_latch_actual_pos, adt_clear_latch, adt_close_latch	
Related Chapter	Position latch	
Related Routine	Routine 13-1 Position Latch Control	

adt_get_latch_status

Instruction Format
int _stdcall adt_get_latch_status(int cardno, int axis, int *status)
Instruction Description

Get single axis position latch status		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
status	Not null	Position latched state
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position latch control class	
Related Instructions	adt_set_latch_mode, adt_get_latch_command_pos, adt_get_latch_actual_pos, adt_clear_latch, adt_close_latch	
Related Chapter	Position latch	
Related Routine	Routine 13-1 Position Latch Control	

adt_get_latch_command_pos

Instruction Format		
int _stdcall adt_get_latch_command_pos(int cardno, int axis, long *pos, int index = 0, int count = 1)		
Instruction Description		
Get latch logic position		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos	Not null	Single axis latch logic position
index	[0,49]	Start index for getting latch position; in home signal latch and encoder Z-phase signal latch mode, index=0; in continuous latch mode, up to 50 latch data can be extracted at one time, index [0,49]
count	[1,50]	Number of gotten latch positions, in home signal

		latch and encoder Z-phase signal latch mode, count=1; in continuous latch mode, up to 50 latch data can be extracted at one time, index [1, 50]

Returned Value	
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction Position latch control class
Related Instructions	adt_set_latch_mode, adt_get_latch_status, adt_get_latch_actual_pos, adt_clear_latch, adt_close_latch
Related Chapter	Position latch
Related Routine	Routine 13-1 Position Latch Control

adt_get_latch_actual_pos

Instruction Format		
int _stdcall adt_get_latch_actual_pos(int cardno, int axis, long *pos, int index = 0, int count = 1)		
Instruction Description		
Get latch encoder position		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
pos	Not null	Single axis latch encoder position
index	[0,49]	Start index for getting latch position; in home signal latch and encoder Z-phase signal latch mode, index=0; in continuous latch mode, up to 50 latch data can be extracted at one time, index [0,49]
count	[1,50]	Number of gotten latch positions, in home signal latch and encoder Z-phase signal latch mode, count=1; in continuous latch mode, up to 50 latch data can be extracted at one time, index [1,

		50]
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position latch control class	
Related Instructions	adt_set_latch_mode, adt_get_latch_status, adt_get_latch_command_pos, adt_clear_latch, adt_close_latch	
Related Chapter	Position latch	
Related Routine	Routine 13-1 Position Latch Control	

adt_clear_latch

Instruction Format		
int _stdcall adt_clear_latch(int cardno, int axis)		
Instruction Description		
Clear latch data		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position latch control class	
Related Instructions	adt_set_latch_mode, adt_get_latch_status, adt_get_latch_command_pos, adt_get_latch_actual_pos, adt_close_latch	
Related Chapter	Position latch	
Related Routine	Routine 13-1 Position Latch Control	

adt_close_latch

Instruction Format
int _stdcall adt_close_latch (int cardno, int axis)
Instruction Description

Clear latch data and close Position latch		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Position latch control class	
Related Instructions	adt_set_latch_mode, adt_get_latch_status, adt_get_latch_command_pos, adt_get_latch_actual_pos, adt_clear_latch	
Related Chapter		
Related Routine		

17.11. DA control class

adt_set_daout

Instruction Format		
int _stdcall adt_set_daout(int cardno, int port, double value)		
Instruction Description		
Set DA output function		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
port	[1, 2]	DA port number
value	[0-10]	DA output voltage
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction DA control class	
Related Instructions		
Related Chapter	DA control	
Related Routine	Routine 14-1 DA Control	

adt_set_da_adjust

Instruction Format		
int _stdcall adt_set_da_adjust (int cardno, int port, double VoltageMap[21])		
Instruction Description		
Set the DA correction table		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
port	[1, 2]	DA port number
VoltageMap	[0-10]	unit:V， The output voltage is set by calling the "adt_set_daout" with the monotone increasing interval of 0.5 and 0-10 in the state without opening correction

Returned Value	
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction DA control class
Related Instructions	adt_set_daout
Related Chapter	
Related Routine	

17.12. User control class

adt_set_user_password

Instruction Format		
int _stdcall adt_set_user_password(int cardno, const char* oldpw, const char* newpw)		
Instruction Description		
Modify/set user password		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
oldpw		Old password verification; the password will not be verified when the function is first used
newpw		New password setting
Returned Value		
0: passwd all authentication tokens updated successfully; 1: The password length is greater than 128, or the new password is the same as the old password; other values indicate execution failed; refer to Standard Error Code for the reason of failure;		
Instruction Classification	Standard instruction User control class	
Related Instructions	adt_check_password	
Related Chapter	User control	
Related Routine	Routine 15-1 User Control	

adt_check_password

Instruction Format		
int _stdcall adt_check_password(int cardno, const char* password)		
Instruction Description		
Verify user password		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
password		Verify password
Returned Value		

0: Password check pass; 1: Password check failed，other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction User control class
Related Instructions	adt_set_user_password
Related Chapter	User control
Related Routine	Routine 15-1 User Control

17.13. Homing Control Class

adt_set_home_mode

Instruction Format		
int _stdcall adt_set_home_mode(int cardno, int axis, int mode, int stop0, int limit, int stop1, double BackRange, double EZRange, double offset)		
Instruction Description		
Homing mode setting		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
HomeMode	[0-7]	Homing direction and homing type bit0: Homing direction 0: Negative; 1: Positive bit1: Homing motion type 0: Linear motion; When accurately positioning the home, exit the home in a fixed positive direction, then approach the home in the negative direction 1: Circumferential motion; When accurately positioning the home, first exit the home in reverse, then approach the home in the set homing direction $\text{HomeMode} = \text{bit1} \times 2^1 + \text{bit0}$
Stop0Active	[0, 1]	STOP0 (machine home) active level 0: Active low 1: Active high
LimitActive	[0-7]	Positive/negative limit setting bit0: Active level 0: Active low, 1: Active high bit1: Negative limit enable: 0: On; 1: Off bit2: Positive limit enable: 0: On; 1: Off $\text{LmtActive} = \text{bit2} \times 2^2 + \text{bit1} \times 2^1 + \text{bit0}$
Stop1Active		STOP1 (encoder Z phase) active level 0: Low 1: High Other: No encoder Z phase
BackRange		Reverse distance, value range >1, less than the

		distance between the positive limit and the home, unit: mm
EZRange		Encoder Z phase search distance, ==0 no offset, >0 positive offset, <0 negative offset, unit: mm
offset		Home offset, ==0 no offset, >0 positive offset, <0 negative offset, unit: mm
Returned Value		
0:indicates successful execution,-x :Error on x th parameter		
Instruction Classification	Standard instruction Homing control class	
Related Instructions	adt_set_home_speed、adt_set_home_process、adt_get_home_status adt_set_home_speed, adt_set_home_process, adt_get_home_status	
Related Chapter	Homing	
Related Routine	Routine 8-1 Homing	

adt_set_home_speed

Instruction Format		
int _stdcall adt_set_home_speed(int cardno, int axis, double startv, double searchv, double homev, double acc, double EZspeed)		
Instruction Description		
Homing speed setting		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
startv		Homing start speed, unit: mm/s
searchv		Homing search high speed, unit: mm/s
homev		Homing search low speed, unit: mm/s
acc		Homing search acceleration, unit: mm/s^2
EZspeed		Homing encoder Z-phase search speed, unit:

		mm/s
Returned Value		
0:indicates successful execution,-x :Error on x th parameter		
Instruction Classification	Standard instruction Homing control class	
Related Instructions	adt_set_home_mode、adt_set_home_process、adt_get_home_status adt_set_home_mode, adt_set_home_process, adt_get_home_status	
Related Chapter	Homing	
Related Routine	Routine 8-1 Homing	

adt_set_home_process

Instruction Format		
int _stdcall adt_set_home_process(int cardno, int axis)		
Instruction Description		
Single-axis drive homing, combined with adt_get_home_status to complete single-axis homing.		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Homing control class	
Related Instructions	adt_set_home_mode、adt_set_home_speed、adt_get_home_status adt_set_home_mode, adt_set_home_speed, adt_get_home_status	
Related Chapter	Homing	
Related Routine	Routine 8-1 Homing	

adt_get_home_status

Instruction Format		
int _stdcall adt_get_home_status(int cardno, int axis)		
Instruction Description		
Single-axis homing status query, combined with adt_set_home_process to complete single-axis homing		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
axis	[1-8]	Axis number
Returned Value		
0 indicates successful homing, >0 indicates homing step number, <0 indicates homing failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Homing control class	
Related Instructions	adt_set_home_mode、adt_set_home_speed、adt_set_home_process adt_set_home_mode, adt_set_home_speed, adt_set_home_process	
Related Chapter	Homing	
Related Routine	Routine 8-1 Homing	

17.14. Handwheel Control Class

adt_set_handwheel_move

Instruction Format		
int _stdcall adt_set_handwheel_move(int cardno, int mode, int axis)		
Instruction Description		
Handwheel following motion, make corresponding axis follow the position by the change of the handwheel		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
mode	[0, 1]	Axis selection mode: 0: Select the following axis through IO on the handheld box, up to 6 axes; 1: Specify the axis number according to the axis parameter, up to 8 axes;
axis	[1-8]	Axis number
Returned Value		
0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure		
Instruction Classification	Standard instruction Handwheel control class	
Related Instructions	adt_stop_handwheel	
Related Chapter	Chapter 16 Handwheel Control	
Related Routine	Routine 16-1 Handwheel Control	

adt_stop_handwheel

Instruction Format		
int _stdcall adt_stop_handwheel(int cardno)		
Instruction Description		
Stop handwheel function		
Parameter Description		
Name	Range	Description
cardno	[0-9]	Control card index number
Returned Value		

0 indicates successful execution, other values indicate execution failed; refer to Standard Error Code for the reason of failure	
Instruction Classification	Standard instruction Handwheel control class
Related Instructions	adt_set_handwheel_move
Related Chapter	Chapter 16 Handwheel Control
Related Routine	Routine 16-1 Handwheel Control

Chapter 18 Appendix

18.1. Routine Index

Routine 4-1 Basic Control
Routine 5-1 Standard Configuration Method Routines
Routine 6-1 Basic Speed Planning and Quantitative Driving
Routine 6-2 Speed Look-ahead
Routine 6-3 Arc Interpolation Speed Constraint
Routine 6-4 Speed Ratio
Routine 7-1 Quantitative Drive
Routine 7-2 Continuous Drive
Routine 7-3 Linear Interpolation
Routine 7-4 Plane Arc Interpolation
Routine 7-5 Cache Interpolation
Routine 7-6 Gantry Double Drive
Routine 7-7 Drive Status and Stop Information
Routine 8-1 Homing
Routine 9-1 Position Control
Routine 10-1 Output Port Control
Routine 10-2 Input Port Control
Routine 10-3 Input Filter Control
Routine 11-1 Cache Output Control
Routine 11-2 Cache PWM Control
Routine 12-1 Position Comparator
Routine 13-1 Position Latch Control
Routine 14-1 DA Control
Routine 15-1 User Control
Routine 16-1 Handwheel Control

18.2. Auxiliary Interfaces and Definitions

DecodeErrorCode -- Example of Error Code Parsing

DecodeStopData -- Example of Drive Stop Information Parsing

DoEvent – Transfer of Control of the Operating System

DoEvent passes control to the operating system. Control is returned when the operating system has processed the events in the queue and sent all the keys in the SendKeys queue.

DoEvent is especially useful for simplifying procedures such as allowing a user to cancel a started process, e.g. searching for a file. For long-term processes, it is best to give up control by using a timer or by assigning tasks to ActiveX EXE components. Later, tasks are still completely independent of the application, and multitasking and time slices are handled by the operating system.

If the system requires high on real-time performance and does not take into account the operating system event queue, consider putting the process flow into a thread and not using DoEvent.

```
void DoEvent()
{
	MSG msg;
	if (::PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
	{
		::TranslateMessage(&msg);
		::DispatchMessage(&msg);
	}
}
```

VERIFY_RETURN

```
#define VERIFY_RETURN(x)   {int retn = x; if(retn)        return retn;}
```

VERIFY_RETURN_VALUE

```
#define VERIFY_RETURN_VALUE(x, val)   {if(x)   return val;}
```

VERIFY_RETURN_MSG

```
#define VERIFY_RETURN_MSG(retn, itf, val)      {       \
	if(retn)        \
	{       \
	char msg[256] = {0};       \
```

```
        DecodeErrorCode(msg, 256, retn, itf);\
        printf(msg);  \
        system("pause");   \
        return val;    \
        }       \
}
```

CONFIRM_DRIVE

```
#define CONFIRM_DRIVE(val)          {       \
        printf("Parameters have been set!/n Please enter: 1--Drive Other--Do not drive and
exit\n");       \
        int flag = 0;  cin>>flag;    if(1 != flag)   return val;    \
}
```