WIKAN

WIMC Library Reference GUIDE

WIMC Library Reference Guide R01

http://WWW.WIKAN.CO.KR

All rights reserved

2013-10-17

Document History

Date	Description
2013/10/17	[배포] WIMC Library Reference Guide R01
2023/10/24	[함수 업데이트;라이브러리 버전(R5.20)]

함수 업데이트 리스트(R5.20)

```
//-----------------------------------------------------------------------------------------------------------
// Gerneral
//-----------------------------------------------------------------------------------------------------------
WIMC_INT32 WIMC_Open_Tcp( WIMC_UINT32 ip, WIMC_INT32 port, WIMC_INT32 baudrate );
WIMC_BOOL WIMC_Load_wConfigFile( WIMC_CHAR* pPathName );
WIMC_BOOL WIMC_Save_wConfigFile( WIMC_CHAR* pPathName );
WIMC_BOOL WIMC_Load_ConfigFile( char* pPathName );
WIMC_BOOL WIMC_Save_ConfigFile( char* pPathName );
void WIMC_Get_wDllVer( WIMC_CHAR* pVerName, WIMC_INT32 numOfChar );
void WIMC_Get_DllVer( char* pVerName, WIMC_INT32 numOfChar );
void WIMC_Sleep(WIMC_INT32 msec);

void WIMC_Get_EngineVer( char* pVerName, WIMC_INT32 numOfChar );


//-----------------------------------------------------------------------------------------------------------
// Settings
//-----------------------------------------------------------------------------------------------------------
WIMC_UINT32 WIMC_Set_GIOMap( WIMC_UINT32 mapAxis );
WIMC_UINT32 WIMC_Get_GIOMap( void);

WIMC_BOOL WIMC_Set_VelMode3StartMode( WIMC_BOOL mode);
WIMC_INT32 WIMC_Get_VelMode3StartMode( void);
WIMC_BOOL WIMC_MainLinkAlarmOn( WIMC_BOOL on);
WIMC_BOOL WIMC_Is_MainLinkAlarmOn( void);

WIMC_BOOL WIMC_Set_AxisPulseWidth( WIMC_UINT32 pulseWidth);
WIMC_UINT32 WIMC_Get_AxisPulseWidth( void);

WIMC_BOOL WIMC_HomeLimitAxis( WIMC_UINT8 axis, WIMC_BOOL on);
WIMC_BOOL WIMC_Is_HomeLimitAxis( WIMC_UINT8 axis);
WIMC_BOOL WIMC_HomeBackSearchAxis( WIMC_UINT8 axis, WIMC_BOOL on);
WIMC_BOOL WIMC_Is_HomeBackSearchAxis( WIMC_UINT8 axis);


//-----------------------------------------------------------------------------------------------------------
// Axis Pulse
//-----------------------------------------------------------------------------------------------------------
WIMC_BOOL WIMC_Set_DirLevel( WIMC_UINT8 axis, WIMC_BOOL bLevel);
WIMC_BOOL WIMC_Get_DirLevel( WIMC_UINT8 axis);
WIMC_BOOL WIMC_Set_CwLevel( WIMC_UINT8 axis, WIMC_BOOL bLevel);
WIMC_BOOL WIMC_Get_CwLevel( WIMC_UINT8 axis);
WIMC_BOOL WIMC_Set_CcwLevel( WIMC_UINT8 axis, WIMC_BOOL bLevel);
WIMC_BOOL WIMC_Get_CcwLevel( WIMC_UINT8 axis);


//-----------------------------------------------------------------------------------------------------------
// Limit
//-----------------------------------------------------------------------------------------------------------
WIMC_BOOL WIMC_IsEnable_LowerLimit( WIMC_UINT8 axis);
WIMC_BOOL WIMC_IsEnable_UpperLimit( WIMC_UINT8 axis);
```

```
//-----------------------------------------------------------------------------------------------------------
// Soft Limit
//-----------------------------------------------------------------------------------------------------------
WIMC_INT32 WIMC_Get_SoftLowerLimit( WIMC_UINT8 axis);
WIMC_INT32 WIMC_Get_SoftUpperLimit( WIMC_UINT8 axis);
WIMC_BOOL WIMC_Enable_SoftLowerLimit( WIMC_UINT8 axis, WIMC_BOOL bEnable );
WIMC_BOOL WIMC_Enable_SoftUpperLimit( WIMC_UINT8 axis, WIMC_BOOL bEnable );

WIMC_BOOL WIMC_IsEnable_SoftLowerLimit( WIMC_UINT8 axis);
WIMC_BOOL WIMC_IsEnable_SoftUpperLimit( WIMC_UINT8 axis);

//-----------------------------------------------------------------------------------------------------------
//              속도 동작 옵션설정
//-----------------------------------------------------------------------------------------------------------
// 속도 오버라이딩
WIMC_BOOL WIMC_Set_VelOverride( WIMC_INT32 percentagex10 );
WIMC_INT32 WIMC_Get_VelOverride( void );
WIMC_BOOL WIMC_Set_S1VelOverride( WIMC_INT32 percentagex10 );
WIMC_INT32 WIMC_Get_S1VelOverride( void );
WIMC_BOOL WIMC_Set_S2VelOverride( WIMC_INT32 percentagex10 );

//-----------------------------------------------------------------------------------------------------------
// 개별축동작명령
//-----------------------------------------------------------------------------------------------------------
// 조건이송
WIMC_BOOL  WIMC_XMove( WIMC_UINT8 axis, WIMC_INT32 pulse[], WIMC_UINT32 decCond, WIMC_UINT32 stopCond,
WIMC_INT32 decVel, WIMC_INT32 decPos );



//-----------------------------------------------------------------------------------------------------------
// 원점복귀 옵션
//-----------------------------------------------------------------------------------------------------------
WIMC_BOOL WIMC_Is_HomeSensor( WIMC_UINT8 axis );


//-----------------------------------------------------------------------------------------------------------
// 축 상태
//-----------------------------------------------------------------------------------------------------------
WIMC_BOOL WIMC_Is_HomeReturnAlarm( WIMC_UINT8 axis );

WIMC_INT32 WIMC_Get_VelStateAxis(WIMC_UINT8 axis); // 속도 상태= VELSTATE_[STOP/ACC/STEADY/DEC]
WIMC_BOOL WIMC_Is_HomeReturnExe( WIMC_UINT8 axis );
WIMC_BOOL WIMC_Is_LimitL(WIMC_UINT8 axis); // 하한 리미트 검사
WIMC_BOOL WIMC_Is_LimitH(WIMC_UINT8 axis); // 상한 리미트 검사
WIMC_BOOL WIMC_Is_SoftLimitL(WIMC_UINT8 axis); // 하한 소프트 리미트 검사
WIMC_BOOL WIMC_Is_SoftLimitH(WIMC_UINT8 axis); // 상한 소프트 리미트 검사
WIMC_BOOL WIMC_Is_CW(WIMC_UINT8 axis);
WIMC_BOOL WIMC_Is_CCW(WIMC_UINT8 axis);
WIMC_BOOL WIMC_Is_AlarmAxis(WIMC_UINT8 axis);
WIMC_BOOL WIMC_Is_PositionAlarm(WIMC_UINT8 axis);

//-----------------------------------------------------------------------------------------------------------
// GIO
//-----------------------------------------------------------------------------------------------------------
WIMC_BOOL WIMC_Set_GIOPattern(WIMC_UINT32 bitSize, WIMC_UINT8 patternBuf[]);
WIMC_BOOL WIMC_Clr_GIOPattern(WIMC_BOOL end_level);

//-----------------------------------------------------------------------------------------------------------
// EXTEND IO
//-----------------------------------------------------------------------------------------------------------
```

```
WIMC_BOOL WIMC_Is_Enable_ExtIO( void );

//------------------------------------------------------------------------------------------------------------
// CAPTURE
//------------------------------------------------------------------------------------------------------------
WIMC_UINT32 WIMC_Enable_CaptureMode( WIMC_BOOL enable, WIMC_UINT32 mode = 0 );
WIMC_UINT32 WIMC_Is_CaptureMode( void );
WIMC_UINT32 WIMC_Init_Capture( void );
WIMC_INT32 WIMC_Get_CaptureCnt( void );
WIMC_INT32 WIMC_Get_Capture( int n, int axisno );
```

TABLE OF CONTENTS

WIMC Library Reference Guide

1. General 18

1.1. General Function 18

1.1.1. WIMC_Open_Comm 18
1.1.2. WIMC_Change_Comm_Baudrate 20
1.1.3. WIMC_Open_Tcp 21
1.1.4. WIMC_Close 23
1.1.5. WIMC_Set_BoardID 24
1.1.6. WIMC_Get_BoardID 25
1.1.7. WIMC_Load_ConfigFile 26
1.1.8. WIMC_Load_wConfigFile 33
1.1.9. WIMC_Save_ConfigFile 40
1.1.10. WIMC_Save_wConfigFile 41
1.1.11. WIMC_Is_Link 42
1.1.12. WIMC_Get_DllVer 43
1.1.13. WIMC_Get_wDllVer 45
1.1.14. WIMC_Is_Err 47
1.1.15. WIMC_Get_LastErr 48
1.1.16. WIMC_Enable_ErrBeep 49
1.1.17. WIMC_Get_EnginVer 50

2. System Command 51

2.1. 시스템 명령 51

2.1.1. WIMC_Reset 52
2.1.2. WIMC_EStop 53
2.1.3. WIMC_ImStop 55
2.1.4. WIMC_DecStop 57
2.1.5. WIMC_AlarmReset 59
2.1.6. WIMC_Enable_Exe 60
2.1.7. WIMC_Pause 61
2.1.8. WIMC_Resume 62
2.1.9. WIMC_Cancel 63

3. System State 64

3.1. System State 64

3.1.1. WIMC_Is_EStop 64

3.1.2. WIMC_Is_Pause 66
3.1.3. WIMC_Is_Alarm 67
3.1.4. WIMC_Is_EmgExe 68
3.1.5. WIMC_Is_EnableExe 69

4. Main / Aux com port 70

4.1. Main com 70
4.1.1. WIMC_Set_MainBaudrate 70
4.1.2. WIMC_Get_MainBaudrate 71

4.2. Aux com 72
4.2.1. WIMC_Set_AuxBaudrate 72
4.2.2. WIMC_Get_AuxBaudrate 73

4.3. Aux Com Options 74
4.3.1. WIMC_Enable_AuxPortAllCommand 74
4.3.2. WIMC_Enable_AuxPortStatePacket 76
4.3.3. WIMC_Enable_AuxPortStateTimer 78

4.4. Update Baudrate 80
4.4.1. WIMC_Update_Baudrate 80

5. Estop 81

5.1. Emergency Stop 81
5.1.1. WIMC_Set_ImStopDec 81
5.1.2. WIMC_Get_ImStopDec 83
5.1.3. WIMC_Set_ExtEmgStopMode 84
5.1.4. WIMC_Get_ExtEmgStopMode 85

6. Command ID 86

6.1. Command ID 86
6.1.1. WIMC_Set_CmdID 86
6.1.2. WIMC_Get_CmdID 88
6.1.3. WIMC_Get_ExeCmdID 90
6.1.4. WIMC_Get_ReadyCmdID 91

7. Queue 92

7.1. Command Queue 92
7.1.1. WIMC_Enter_QueueCmd 92
7.1.2. WIMC_Leave_QueueCmd 93
7.1.3. WIMC_Enter_ImCmd 94
7.1.4. WIMC_Leave_ImCmd 95

7.1.5. WIMC_Reallocate_SoftQueue 96
7.1.6. WIMC_Get_SoftQueueSize 97
7.1.7. WIMC_Get_CmdQueueLeftCmd 98
7.1.8. WIMC_Get_SoftQueueLeftCmd 99
7.1.9. WIMC_Get_EngineQueueLeftCmd 100
7.1.10. WIMC_Clr_Queue 101

8. State Packet 102

8.1. Busy 102
8.1.1. WIMC_Enable_BusyStatePacket 102
8.1.2. WIMC_Is_BusyStatePacket 104
8.1.3. WIMC_Set_BusyStatePacketCount 105
8.1.4. WIMC_Get_BusyStatePacketCount 106

8.2. Complete 107
8.2.1. WIMC_Enable_CompleteStatePacket 107
8.2.2. WIMC_Is_CompleteStatePacket 109
8.2.3. WIMC_Set_CompleteStatePacketCount 111
8.2.4. WIMC_Get_CompleteStatePacketCount 113

8.3. Interrupt 115
8.3.1. WIMC_Enable_InterruptStatePacket 115
8.3.2. WIMC_Is_InterruptStatePacket 118
8.3.3. WIMC_Set_InterruptStatePacketCount 120
8.3.4. WIMC_Get_InterruptStatePacketCount 122

8.4. Event 124
8.4.1. WIMC_Enable_EventStatePacket 124
8.4.2. WIMC_Is_EventStatePacket 126
8.4.3. WIMC_Set_EventStatePacketCount 127
8.4.4. WIMC_Get_EventStatePacketCount 129

9. Settings 130

9.1. Settings 130
9.1.1. WIMC_MainLinkAlarmOn 130
9.1.2. WIMC_Is_MainLinkAlarmOn 131
9.1.3. WIMC_Set_AxisPulseWidth 132
9.1.4. WIMC_Get_AxisPulseWidth 133
9.1.5. WIMC_Set_InportMode 134
9.1.6. WIMC_Get_InportMode 135
9.1.7. WIMC_Set_OutportMode 136
9.1.8. WIMC_Get_OutportMode 137

9.1.9. WIMC_Set_PosDispMode 138
9.1.10. WIMC_Get_PosDispMode 140
9.1.11. WIMC_Set_PosMode 142
9.1.12. WIMC_Get_PosMode 144
9.1.13. WIMC_Set_BusySize 146
9.1.14. WIMC_Get_BusySize 148
9.1.15. WIMC_Set_InterruptMode 149
9.1.16. WIMC_Get_InterruptMode 151
9.1.17. WIMC_Set_ProcCmdNo 153
9.1.18. WIMC_Get_ProcCmdNo 155
9.1.19. WIMC_Enable_RestrictedImCmd 156
9.1.20. WIMC_Is_RestrictedImCmd 158
9.1.21. WIMC_Enable_IEndComp 160
9.1.22. WIMC_Is_IEndComp 162

10. StateTimer 164

10.1. 상태타이머 (State Timer) 164
10.1.1. WIMC_Set_StateTimer 166
10.1.2. WIMC_Clr_StateTimer 166
10.1.3. WIMC_Get_StateTimer 168
10.1.4. WIMC_Is_StateTimer 169

11. Progress 171

11.1. Progress 171
11.1.1. WIMC_Get_CurCmdPulse 171
11.1.2. WIMC_Get_CmdPulseMax 172

12. Scan Time 173

12.1. Scan Time 173
12.1.1. WIMC_Get_ScanTime 173

13. CExe 174

13.1. 연속실행모드와 비연속 실행모드 174
13.1.1. WIMC_Enable_Cexe 175
13.1.2. WIMC_Is_CExe 176

14. Step Exe 177

14.1. 스텝실행 177
14.1.1. WIMC_Enable_StepExe 178
14.1.2. WIMC_Is_StepExe 179

15. Delay 180

15.1. 동작지연명령 180

15.1.1. WIMC_Delay 181

15.1.2. WIMC_XDelay 182

15.1.3. WIMC_Get_CurDelayTime 186

15.1.4. WIMC_Sleep 187

16. Sync 189

16.1. 명령 동기 189

16.1.1. WIMC_Sync 190

16.1.2. WIMC_Is_Sync 191

17. Wait 192

17.1. 대기명령 192

17.1.1. WIMC_Wait 193

18. Wait Motion 194

18.1. 명령 ID 설정과 Wait Motion 194

18.1.1. WIMC_Enable_Notify_Complete 196

18.1.2. WIMC_Notify_Complete 197

18.1.3. WIMC_WaitMotion 198

18.1.4. WIMC_Register_Complete_Message 199

18.1.5. WIMC_Unregister_Complete_Message 200

19. Command State 201

19.1. 명령 실행 상태 201

19.1.1. WIMC_Get_CmdState 201

19.1.2. WIMC_Is_PreparedCmd 202

20. Motion State 203

20.1. WIMC 엔진의 주요상태값들 203

20.1.1. WIMC_Is_Busy 204

20.1.2. WIMC_Is_OnMotion 205

20.1.3. WIMC_Is_CompleteAxis 206

20.1.4. WIMC_Is_Complete 207

20.1.5. WIMC_Get_VelState 208

20.1.6. WIMC_Get_VelStateAxis 209

21. Axis Pulse 210

21.1. 축이동 펄스신호 기능설정 210

21.1.1. WIMC_Set_PulseMode 211
21.1.2. WIMC_Get_PulseMode 212
21.1.3. WIMC_Set_PulseLevel 213
21.1.4. WIMC_Get_PulseLevel 214
21.1.5. WIMC_Set_DirLevel 215
21.1.6. WIMC_Get_DirLevel 216
21.1.7. WIMC_Set_CwLevel 217
21.1.8. WIMC_Get_CwLevel 218
21.1.9. WIMC_Set_CcwLevel 219
21.1.10. WIMC_Get_CcwLevel 220
21.1.11. WIMC_Is_CW 221
21.1.12. WIMC_Is_CCW 222

22. Limit 223

22.1. 리미트신호 기능설정 223

22.1.1. WIMC_Enable_LowerLimit 224
22.1.2. WIMC_IsEnable_LowerLimit 225
22.1.3. WIMC_Enable_UpperLimit 226
22.1.4. WIMC_IsEnable_UpperLimit 227
22.1.5. WIMC_Set_LowerLimitLevel 228
22.1.6. WIMC_Get_LowerLimitLevel 229
22.1.7. WIMC_Set_UpperLimitLevel 230
22.1.8. WIMC_Get_UpperLimitLevel 231
22.1.9. WIMC_Is_LimitL 232
22.1.10. WIMC_Is_LimitH 233

23. Soft Limit 234

23.1. 소프트 리미트 기능 설정 234

23.1.1. WIMC_Set_SoftLimitMode 235
23.1.2. WIMC_Get_SoftLimitMode 236
23.1.3. WIMC_Set_SoftLowerLimit 237
23.1.4. WIMC_Get_SoftLowerLimit 238
23.1.5. WIMC_Set_SoftUpperLimit 239
23.1.6. WIMC_Get_SoftUpperLimit 240
23.1.7. WIMC_Enable_SoftLowerLimit 241
23.1.8. WIMC_Enable_SoftUpperLimit 242
23.1.9. WIMC_IsEnable_SoftLowerLimit 243
23.1.10. WIMC_IsEnable_SoftUpperLimit 244
23.1.11. WIMC_Is_SoftLimitL 245
23.1.12. WIMC_Is_SoftLimitH 246

24. 속도제어 설정 247

24.1. 속도모드 설정 249

24.1.1. WIMC_Set_VelMode 249
24.1.2. WIMC_Get_VelMode 250
24.1.3. WIMC_Set_VelModeAxis 251
24.1.4. WIMC_Get_VelModeAxis 252
24.1.5. WIMC_Set_VelMode3StartMode 253
24.1.6. WIMC_Get_VelMode3StartMode 254

24.2. 현재속도 255

24.2.1. WIMC_Get_CurVel 255
24.2.2. WIMC_Get_CurVelAxis 256

24.3. 구동속도 257

24.3.1. WIMC_Set_DriveVel 257
24.3.2. WIMC_Get_DriveVel 258
24.3.3. WIMC_Set_DriveVelAxis 259
24.3.4. WIMC_Get_DriveVelAxis 260

24.4. 초속도 261

24.4.1. WIMC_Set_StartVel 261
24.4.2. WIMC_Get_StartVel 262
24.4.3. WIMC_Set_StartVelAxis 263
24.4.4. WIMC_Get_StartVelAxis 264

24.5. 1차 가속도 265

24.5.1. WIMC_Set_Acc 265
24.5.2. WIMC_Get_Acc 266
24.5.3. WIMC_Set_AccAxis 267
24.5.4. WIMC_Get_AccAxis 268

24.6. 1차 감속도 269

24.6.1. WIMC_Set_Dec 269
24.6.2. WIMC_Get_Dec 270
24.6.3. WIMC_Set_DecAxis 271
24.6.4. WIMC_Get_DecAxis 272

24.7. 2차 가속도 273

24.7.1. WIMC_Set_Acc2 273
24.7.2. WIMC_Get_Acc2 274
24.7.3. WIMC_Set_Acc2Axis 275
24.7.4. WIMC_Get_Acc2Axis 276

24.8. 2차 감속도 277

24.8.1. WIMC_Set_Dec2 277
24.8.2. WIMC_Get_Dec2 278
24.8.3. WIMC_Set_Dec2Axis 279
24.8.4. WIMC_Get_Dec2Axis 280

25. 속도 동작 옵션설정 281

25.1. 속도 배율수 281
25.1.1. WIMC_Set_VelMultiplier 282
25.1.2. WIMC_Get_VelMultiplier 283

25.2. 선속도 일정기능 284
25.2.1. WIMC_Enable_CLV 286
25.2.2. WIMC_Is_CLV 287
25.2.3. WIMC_Set_CLVAxis 288
25.2.4. WIMC_Get_CLVAxis 289

25.3. 속도 오버라이딩 290
25.3.1. WIMC_Set_VelOverride 291
25.3.2. WIMC_Get_VelOverride 292
25.3.3. WIMC_Set_S1VelOverride 293
25.3.4. WIMC_Get_S1VelOverride 294
25.3.5. WIMC_Set_S2VelOverride 295
25.3.6. WIMC_Get_S2VelOverride 296
25.3.7. WIMC_Lock_VelOverride 297
25.3.8. WIMC_Unlock_VelOverride 298
25.3.9. WIMC_Is_Lock_VelOverride 299

25.4. 드웰타임 300
25.4.1. WIMC_Set_DwellTime 303
25.4.2. WIMC_Get_DwellTime 304
25.4.3. WIMC_Enable_DwellTimeAxis 305
25.4.4. WIMC_Set_DwellTimeAxis 306
25.4.5. WIMC_Get_DwellTimeAxis 307

25.5. 속도제어 주파수 308
25.5.1. WIMC_Set_VelCtrlHz 308
25.5.2. WIMC_Get_VelCtrlHz 309

25.6. 속도강제수렴 310
25.6.1. WIMC_Enable_VelForceConvergence 310
25.6.2. WIMC_Is_VelForceConvergence 311

25.7. 피드포워드제어기능 FFC 312
25.7.1. WIMC_Enable_FFC 313

25.7.2. WIMC_Is_FFC 314
25.7.3. WIMC_Set_FFC_PMax 315

25.8. Automatic transistion속도/위치제어 자동절환 316
25.8.1. WIMC_Is_VelPosAutoTransition 317
25.8.2. WIMC_Enable_VelPosAutoTransition 318

26. 위치 제어 319

26.1. Current Position 319
26.1.1. WIMC_Set_CurPos 319
26.1.2. WIMC_Get_CurPos 320

26.2. Encoder 321
26.2.1. WIMC_Set_EncPos 321
26.2.2. WIMC_Get_EncPos 322
26.2.3. WIMC_Get_EncZCount 323

26.3. Gear 325
26.3.1. WIMC_Set_GearRatio 327
26.3.2. WIMC_Get_GearRatio 328

26.4. Backlash 329
26.4.1. WIMC_Enable_Backlash 330
26.4.2. WIMC_Set_Backlash 331
26.4.3. WIMC_Get_Backlash 332

27. 보간명령 333

27.1. 직선보간 333
27.1.1. WIMC_Line 333

27.2. 원호보간 335
27.2.1. WIMC_ArcCCW 338
27.2.2. WIMC_ArcCW 339

27.3. 비트패턴 340
27.3.1. WIMC_Set_GIOPattern 341
27.3.2. WIMC_Clr_GIOPattern 344

27.4. 원점복귀 347
27.4.1. WIMC_Home 347

28. 다중보간 349

28.1. 다중보간(Multiple Interpolation) 349
28.1.1. WIMC_Enable_MI 352

28.1.2. WIMC_Is_MI 353
28.1.3. WIMC_MIExe 354

29. 개별축동작명령 355

29.1. 독립이송 355
29.1.1. WIMC_IMove 355

29.2. 연속이송 357
29.2.1. WIMC_CMove 357

29.3. 조건이송 358
29.3.1. WIMC_XMove 358

29.4. 스핀들회전 361
29.4.1. WIMC_Spindle 361

29.5. 원점복귀 363
29.5.1. WIMC_IHome 363

30. 원점복귀 옵션 365

30.1. 원점복귀 365
30.1.1. WIMC_Set_HomeMode 369
30.1.2. WIMC_Get_HomeMode 370
30.1.3. WIMC_Set_HomeDev 371
30.1.4. WIMC_Get_HomeDec 372
30.1.5. WIMC_Set_HomeDir 373
30.1.6. WIMC_Is_HomeDir 374
30.1.7. WIMC_Set_HomeSeekOffsetPulse 375
30.1.8. WIMC_Get_HomeSeekOffsetPulse 376
30.1.9. WIMC_Set_HomeSeekVel 377
30.1.10. WIMC_Get_HomeSeekVel 378
30.1.11. WIMC_Set_HomeSensorLevel 379
30.1.12. WIMC_Is_HomeSensorLevel 380
30.1.13. WIMC_Is_HomeSensor 381
30.1.14. WIMC_Enable_HomeReturnAutoZero 382
30.1.15. WIMC_Is_HomeReturnAutoZero 383
30.1.16. WIMC_Enable_HomeSensorAutoZero 384
30.1.17. WIMC_Is_HomeSensorAutoZero 385
30.1.18. WIMC_Enable_HomeServoReset 386
30.1.19. WIMC_Is_HomeServoReset 387
30.1.20. WIMC_Enable_HomeDevAlarm 388
30.1.21. WIMC_Is_HomeDevAlarm 389
30.1.22. WIMC_Is_HomeReturnExe 390

30.1.23. WIMC_Is_HomeReturnAlarm 391
30.1.24. WIMC_ HomeLimitAxis 392
30.1.25. WIMC_Is_HomeLimitAxis 393
30.1.26. WIMC_ HomeBackSearchAxis 394
30.1.27. WIMC_Is_HomeBackSearchAxis 395

31. SERVO 396

31.1. Sv.On 396
31.1.1. WIMC_Enable_ServoOnAuto 396
31.1.2. WIMC_Is_ServoOnAuto 397
31.1.3. WIMC_Set_ServoOnLevel 398
31.1.4. WIMC_Is_ServoOnLevel 399
31.1.5. WIMC_Set_ServoOnMode 400
31.1.6. WIMC_Get_ServoOnMode 401
31.1.7. WIMC_Set_ServoOn 402
31.1.8. WIMC_Is_ServoOn 403

31.2. Sv.Reset 404
31.2.1. WIMC_Enable_ServoResetAuto 404
31.2.2. WIMC_Is_ServoResetAuto 405
31.2.3. WIMC_Set_ServoResetLevel 406
31.2.4. WIMC_Is_ServoResetLevel 407
31.2.5. WIMC_Set_ServoPulseTime 408
31.2.6. WIMC_Get_ServoPulseTime 409
31.2.7. WIMC_Set_ServoReset 410
31.2.8. WIMC_Is_ServoReset 411

31.3. Sv.EStop 412
31.3.1. WIMC_Enable_ServoEStopAuto 412
31.3.2. WIMC_Is_ServoEStopAuto 413
31.3.3. WIMC_Set_ServoEStopLevel 414
31.3.4. WIMC_Is_ServoEStopLevel 415
31.3.5. WIMC_Set_ServoEStop 416
31.3.6. WIMC_Is_ServoEStop 417

31.4. Sv.UserOut 418
31.4.1. WIMC_Enable_ServoUserOutAuto 418
31.4.2. WIMC_Is_ServoUserOutAuto 419
31.4.3. WIMC_Set_ServoUserOutLevel 420
31.4.4. WIMC_Is_ServoUserOutLevel 421
31.4.5. WIMC_Set_ServoUserOut 422
31.4.6. WIMC_Is_ServoUserOut 423

31.5. Sv.Ready 424
31.5.1. WIMC_Enable_ServoReadyAuto 424
31.5.2. WIMC_Is_ServoReadyAuto 425
31.5.3. WIMC_Set_ServoReadyLevel 426
31.5.4. WIMC_Is_ServoReadyLevel 427
31.5.5. WIMC_Is_ServoReady 428

31.6. Sv.Inpos 429
31.6.1. WIMC_Enable_ServoInposAuto 429
31.6.2. WIMC_Is_ServoInposAuto 431
31.6.3. WIMC_Enable_ServoInposAlarm 432
31.6.4. WIMC_Is_ServoInposAlarm 434
31.6.5. WIMC_Set_ServoInposLevel 435
31.6.6. WIMC_Is_ServoInposLevel 437
31.6.7. WIMC_Set_ServoInposTime 438
31.6.8. WIMC_Get_ServoInposTime 439
31.6.9. WIMC_Is_ServoInpos 440

31.7. Sv.Alarm 441
31.7.1. WIMC_Enable_ServoAlarmAuto 441
31.7.2. WIMC_Is_ServoAlarmAuto 442
31.7.3. WIMC_Set_ServoAlarmLevel 443
31.7.4. WIMC_Is_ServoAlarmLevel 444
31.7.5. WIMC_Is_ServoAlarm 445

31.8. Sv.UserIn 446
31.8.1. WIMC_Set_ServoUserInLevel 446
31.8.2. WIMC_Is_ServoUserInLevel 447
31.8.3. WIMC_Is_ServoUserIn 448

32. GIO 449

32.1. 범용 및 전용 입출력 449
32.1.1. WIMC_Set_Gout 450
32.1.2. WIMC_SetBit_GOut 451
32.1.3. WIMC_ClrBit_GOut 452
32.1.4. WIMC_Pulse_GOut 453
32.1.5. WIMC_Get_GOut 454
32.1.6. WIMC_Get_GIn 455
32.1.7. WIMC_Enable_GIOEncoder 456
32.1.8. WIMC_Set_GIOUpCounter 458
32.1.9. WIMC_Get_GIOUpCounter 459
32.1.10. WIMC_Set_GIOEncoder 460

32.1.11. WIMC_Get_GIOEncoder 462
32.1.12. WIMC_Get_ADC 464
32.1.13. WIMC_Set_PWM 465
32.1.14. WIMC_Get_PWMPeriodCount 467
32.1.15. WIMC_Get_PWMDutyCount 469

33. EXTEND IO 471

33.1. 확장 입출력 471
33.1.1. WIMC_Enable_ExtIO 471
33.1.2. WIMC_Is_Enable_ExtIO 473
33.1.3. WIMC_Set_GIOMap 475
33.1.4. WIMC_Get_GIOMap 477
33.1.5. WIMC_Enable_ExtServoIO 478
33.1.6. WIMC_Is_ExtIO_Link0 479
33.1.7. WIMC_Is_ExtIO_Link1 480
33.1.8. WIMC_Is_ExtIO_Link2 481
33.1.9. WIMC_Is_ExtIO_Link3 482
33.1.10. WIMC_Reset_ExtIO 483
33.1.11. WIMC_EmgStop_ExtIO 484
33.1.12. WIMC_Restart_ExtIO 485
33.1.13. WIMC_Set_ExtIO 486
33.1.14. WIMC_Get_ExtIO 488
33.1.15. WIMC_Get_ExtIO_MPG_Status 489
33.1.16. WIMC_Get_ExtIO_MPG_Count 490

34. 비교기 491

34.1. 위치비교기 491
34.1.1. WIMC_Enable_Comparator 492
34.1.2. WIMC_Register_Comparator_Message 493
34.1.3. WIMC_Unregister_Comparator_Message 494
34.1.4. WIMC_Set_PosComparator 495
34.1.5. WIMC_Set_PosComparatorX 497

35. 캡쳐기능 499

35.1. 위치 캡쳐기능 499
35.1.1. WIMC_Enable_CaptureMode 500
35.1.2. WIMC_Is_CaptureMode 501
35.1.3. WIMC_Init_Capture 502
35.1.4. WIMC_Get_CaptureCnt 503
35.1.5. WIMC_Get_Capture 504

36. PID 505

36.1. PID 제어기 505

36.1.1. WIMC_Off_PID 512
36.1.2. WIMC_Reset_PID 513
36.1.3. WIMC_Run_PID 514
36.1.4. WIMC_Setup_PID 515
36.1.5. WIMC_Set_PIDSV 517
36.1.6. WIMC_Is_PID_On 519
36.1.7. WIMC_Get_PID_Setup 520

37. ALARM 522

37.1. 알람동작 522

37.1.1. WIMC_Is_AlarmAxis 524
37.1.2. WIMC_Is_PositionAlarm 525
37.1.3. WIMC_Get_Alarm 526
37.1.4. WIMC_Clr_AlarmOption 528
37.1.5. WIMC_Set_AlarmOption 529
37.1.6. WIMC_Get_AlarmOption 530
37.1.7. WIMC_Enable_StopRotateByAlarm 531
37.1.8. WIMC_Is_StopRotateByAlarm 532

부록 I. WIMC 라이브러리 찾아보기 533

WIMC Library Reference

WIMC Library 함수 목록입니다.

1. General

1.1. General Function

1.1.1. WIMC_Open_Comm

➲ Syntax

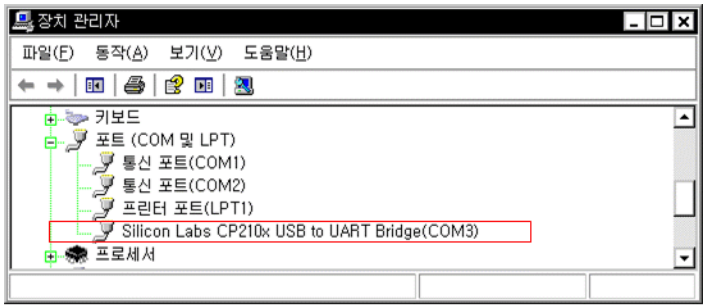
```
WIMC_INT32      WIMC_API            WIMC_Open_Comm
(
                WIMC_INT32          portNo,
                WIMC_INT32          baudrate
);
```

➲ Description

시리얼포트 장치를 Open 하여 모션 보드와 통신을 시작합니다.
WIMC_Is_Link() 함수를 통해 모션보드와 연결상태를 확인하실 수 있습니다.

❶ 모션보드 연결

연결에 맞는 PC의 통신포트(portNo)를 선택합니다. USB로 연결하더라도 가상시리얼통신포트로 변환되므로, 아래 그림과 같이 해당 USB포트와 링크된 가상시리얼포트번호를 선택하셔야 합니다.



연결포트를 선택하셨다면 연결속도(baudrate)를 선택합니다.

※ (주의) 230kbps 이상의 고속으로 연결 시 연결된 직렬포트가 지원하는 최대 통신속도를 반드시 확인하셔야 합니다. 일반적으로 PC 내장 시리얼 포트(ex. COM1, COM2)의 최대 전송속도는 115,200 bps 입니다.

❷ 모션보드 ID

모션보드 ID란 하나의 프로그램에서 2개이상의 모션보드를 사용할 경우 특정 모션보드를 지정하는데 사용됩니다.
모션보드 ID는 WIMC_Open_xxx 함수들 또는 WIMC_Get_BoardID()를 통해 얻을 수 있습니다.
아래 코드예시를 참고하세요.

Ex.)

```
WIMC_Set_BoardID( 첫번재 모션보드 ID);      // 모션보드를 지정합니다.
if( WIMC_Is_Link() == WIMC_TRUE)          {// 첫번째 모션보드가 Link 되었습니다.}
WIMC_Set_BoardID( 두번째 모션보드 ID);
if( WIMC_Is_Link() == WIMC_TRUE)          {// 두번째 모션보드가 Link 되었습니다.}
```

※ 내부적으로 WIMC_Set_BoardID를 호출하여 현재 모션보드 ID가 새로운 모션 보드 ID로 설정됩니다.

➲ Parameters

portNo

시리얼 포트 번호 입니다.(COM1 이면 값은 1이 됩니다.)

baudrate

연결 속도를 설정합니다.

설정가능한 속도는 115200/ 230400/ 460800/ 921600 입니다.

```
WIMC_MAINBPS_115200( =115200)
WIMC_MAINBPS_230400( =230400)
WIMC_MAINBPS_460800( =460800)
WIMC_MAINBPS_921600( =921600)
```

➲ Return Value

정상일 경우 보드 ID(0~16)를 반환합니다. (최대 ID 값은 라이브러리 버젼마다 다를 수 있습니다.)

Return code/value	Description
WIMC_ERR	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

```
WIMC_INT32              comport;
WIMC_UINT32             baudrate;
WIMC_INT32              retVal;

comport = 12;                       // COM12 일 경우
baudrate = WIMC_MAINBPS_115200;

retVal = WIMC_Open_Comm( comport, baudrate);

if( retVal == WIMC_ERR)
{
        WIMC_INT32      errCode;
        errCode = WIMC_Get_LastErr();
        printf("WIMC Err> errCode is %d\r\n", errCode);
}
else
{
        printf("WIMC> Success to open a comport( port=%d, baudrate=%d)\r\n", comport, baudrate);
}
```

➲ See Also

1.1.2. WIMC_Change_Comm_Baudrate

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Change_Comm_Baudrate
(
              WIMC_INT32              baurdrate
);
```

➲ Description

시리얼포트의 통신속도를 변경합니다. 시리얼 포트장치가 Open 되어 있어야 적용됩니다.

> RS232C 통신설정 사양은 아래와 같습니다.
통신속도 : 115200/230400/460800/921600 bps
데이터 비트 : 8 bit
스톱비트 : 1 bit
패리티 비트 : no-parity

➲ Parameters

baudrate
변경가능한 속도는 115200/230400/460800/921600 입니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상 처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

1.1.3. WIMC_Open_Tcp

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Open_Tcp
(
              WIMC_UINT32             ip,
              WIMC_UINT32             port,
              WIMC_UINT32             baudrate
);
```

➲ Description

ETHERNET으로 모션보드와 연결 합니다.

ETHERNET 연결시 WIMC 라이브러리는 tcp client로 동작합니다.

그리고 모션보드는 tcp server로 동작합니다.

WIMC_Is_Link() 함수를 통해 모션보드와 연결상태를 확인하실 수 있습니다.

❶ 모션보드 ID

모션보드 ID란 하나의 프로그램에서 2개이상의 모션보드를 사용할 경우 특정 모션보드를 지정하는데 사용됩니다.

모션보드 ID는 WIMC_Open_xxx 함수들 또는 WIMC_Get_BoardID()를 통해 얻을 수 있습니다.

WIMC_Set_BoardID() 함수를 참고하시어 사용할 수 있습니다.

➲ Parameters

ip

모션보드 ip 주소입니다.

ex) (공장 출하 값: 10.1.0.1)

```
#define MAKE_IP(a,b,c,d) (( (DWORD)a<<0) | ((DWORD)b<<8) | ((DWORD)c<<16) | ((DWORD)d<<24) )
WIMC_Open_Tcp( MAKE_IP(10,1,0,1), 2000, 230400);
```

port

port 번호 입니다.(공장출하 값: 2000)

baudrate

모션보드 통신설정(공장출하 값 : 230400)

➲ Return Value

정상일 경우 보드 ID(0~8)를 반환합니다. (최대 ID 값은 라이브러리 버젼마다 다를 수 있습니다.)

Return code/value	Description
WIMC_ERR	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

```
//-----------------------------------------------------------------------
//      ETHERNET 연결 예시
//-----------------------------------------------------------------------
```

```
#define MAKE_IP(a,b,c,d) (( (DWORD)a<<0) | ((DWORD)b<<8) | ((DWORD)c<<16) | ((DWORD)d<<24) )

        if ( WIMC_Open_Tcp( MAKE_IP(192,168,0,10), 2000, 115200) )
        {
                WIMC_INT32       errCode;
                errCode = WIMC_Get_LastErr();

                printf("WIMC Err> errCode is %d\r\n", errCode);
        }
        else
        {
                printf("WIMC> Success to tcp open \r\n");
        }
```

➲ See Also

1.1.4. WIMC_Close

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Close( void);
```

➲ Description

모션보드와의 연결을 종료합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상 처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

1.1.5. WIMC_Set_BoardID

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_BoardID
(
              WIMC_INT32              boardID
);
```

➲ Description

현재 사용중인 모션보드 ID를 변경합니다.
이후에 사용되는 모든 명령들은 변경된 모션보드 ID로 처리됩니다.
2개이상의 모션보드를 이용할 경우에 사용합니다.

❶ 모션보드 ID

모션보드 ID란 하나의 프로그램에서 2개이상의 모션보드를 사용할 경우 특정 모션보드를 지정하는데 사용됩니다.
모션보드 ID는 WIMC_Open_xxx 함수들 또는 WIMC_Get_BoardID()를 통해 얻을 수 있습니다.
아래 코드예시를 참고하세요.

Ex.)

```
WIMC_Set_BoardID( 첫번재 모션보드 ID);      // 모션보드를 지정합니다.
if( WIMC_Is_Link() == WIMC_TRUE)            {// 첫번째 모션보드가 Link 되었습니다.}
WIMC_Set_BoardID( 두번째 모션보드 ID);
if( WIMC_Is_Link() == WIMC_TRUE)            {// 두번째 모션보드가 Link 되었습니다.}
```

➲ Parameters

boardId

모션보드 ID

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

WIMC_Get_BoardId();

1.1.6. WIMC_Get_BoardID

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_BoardID( void);
```

➲ Description

현재 사용중인 엔진보드 ID를 읽습니다.

❶ 모션보드 ID

모션보드 ID란 하나의 프로그램에서 2개이상의 모션보드를 사용할 경우 특정 모션보드를 지정하는데 사용됩니다.
모션보드 ID는 WIMC_Open_xxx 함수들 또는 WIMC_Get_BoardID()를 통해 얻을 수 있습니다.
아래 코드예시를 참고하세요.

Ex.)

```
WIMC_Set_BoardID( 첫번재 모션보드 ID);      // 모션보드를 지정합니다.
if( WIMC_Is_Link() == WIMC_TRUE)          {// 첫번째 모션보드가 Link 되었습니다.}
WIMC_Set_BoardID( 두번째 모션보드 ID);
if( WIMC_Is_Link() == WIMC_TRUE)          {// 두번째 모션보드가 Link 되었습니다.}
```

➲ Parameters

➲ Return Value

정상일 경우 board ID(= 0이상의 값)를 반환합니다.

Return code/value	Description
WIMC_ERR	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

WIMC_Set_BoardID();
WIMC_Close();

1.1.7. WIMC_Load_ConfigFile

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Load_ConfigFile
(
              char*                   pPathName
);
```

➲ Description

모션마스터의 장치 구성 파일을 이용하여 모션보드를 초기화 합니다.
(장치 구성 설정 내용은 아래 Example2 를 참고 하실 수 으며, 코드방식 설정으로도 초기화 가능합니다.(권장))

아래 그림과 같이 “모션마스터>모니터링” 화면의 **엔진구성전용 옵션**을 체크 한후 저장한 장치 구성 파일을 사용해 주십시오.(CNC 설정을 제외하기 위한 옵션)

➲ Parameters

pPathName
파일의 경로 및 이름입니다.
※ 구성설정 파일은 절대 경로를 사용합니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	성공했습니다.
WIMC_FALSE	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

```
WIMC_BOOL        bResult;

bResult = WIMC_Load_ConfigFile("WIMC_Config.ini");

if ( bResult == WIMC_FALSE)
{
        WIMC_INT32       errCode;
        errCode = WIMC_Get_LastErr();

        printf("WIMC Err> errCode is %d\r\n", errCode);
}
else
{
        printf("WIMC> Success to load a config file\r\n");
}
```

➲ Example2(코드 방식 장치구성 초기화)

```
//-----------------------------------------------------------------------
//      > SOFTWARE RESET
//-----------------------------------------------------------------------
WIMC_Enter_ImCmd();

WIMC_Clr_Queue();
WIMC_Sleep(10);
WIMC_AlarmReset();
WIMC_Sleep(10);
WIMC_Enable_Exe();
WIMC_Sleep(10);
WIMC_Cancel();
WIMC_Sleep(10);

WIMC_Leave_ImCmd();



#define ToAxis(axisNo) (0x01<<axisNo)

//-----------------------------------------
//
// 칩 메인 구성설정
//
//-----------------------------------------
// 큐명령진입
WIMC_Enter_QueueCmd();

// 명령큐 BUSY 25%
WIMC_Set_BusySize( 25 );
// 상태 타이머 레지스터 설정
WIMC_Set_StateTimer(
        WIMC_STATETIMER_100Hz |
        WIMC_STATETIMER_COMPRESSED |
        WIMC_STATETIMER_CHIPSTATUS |
        WIMC_STATETIMER_AXISSTATUS |
        WIMC_STATETIMER_CURPOSITION |
        WIMC_STATETIMER_ENCPOSITION |
        WIMC_STATETIMER_CURVELOCITY |
        WIMC_STATETIMER_GIO |
        WIMC_STATETIMER_USER_R23
        );
// 입출력 포트 사양
WIMC_Set_InportMode( WIMC_INPORTMODE_PULLUP );
WIMC_Set_OutportMode( WIMC_OUTPORTMODE_TOTEMPOLE_PULLUP );
// 위치결정모드
```

```
WIMC_Set_PosMode( WIMC_POSMODE_REL );
// 1회 명령처리수 1~16
WIMC_Set_ProcCmdNo( 4 );

// 알람동작 옵션
WIMC_Set_AlarmOption(
        WIMC_ALARMOPTION_IENDCOMP
        // | WIMC_ALARMOPTION_WDT
        // | WIMC_ALARMOPTION_RECOVERREG
        | WIMC_ALARMOPTION_EMGSTOPIMSTOP
        );
// 비상정지 감속도
WIMC_Set_ImStopDec( 65535 );
// 위치표시모드
WIMC_Set_PosDispMode( WIMC_POSDISPMODE_IGNORE_GEAR );
// 위치모드
WIMC_Set_PosMode( WIMC_POSMODE_REL );
// 상태패킷전송
WIMC_Enable_BusyStatePacket(TRUE);
WIMC_Enable_CompleteStatePacket(TRUE);
WIMC_Enable_InterruptStatePacket(FALSE);
WIMC_Enable_EventStatePacket(FALSE);
// 인터럽트동작모드
WIMC_Set_InterruptMode( WIMC_INTERRUPTMODE_CHANGE_BYTE_R01 );
// 메인속도옵션
WIMC_Set_VelCtrlHz(400);
WIMC_Enable_VelForceConvergence(TRUE);
WIMC_Enable_FFC(TRUE);
WIMC_Set_FFC_PMax(100);
WIMC_Set_DwellTime(0);
// 보간끝점 자동보상
WIMC_Enable_IEndComp(TRUE);

//-----------------------------------------
//
// 각 축의 전자기어비
//
//-----------------------------------------
WIMC_Set_GearRatio( WIMC_X, 0);
WIMC_Set_GearRatio( WIMC_Y, 0);
WIMC_Set_GearRatio( WIMC_Z, 0);
WIMC_Set_GearRatio( WIMC_U, 0);
WIMC_Set_GearRatio( WIMC_V, 0);
WIMC_Set_GearRatio( WIMC_W, 0);
WIMC_Set_GearRatio( WIMC_S, 0);
WIMC_Set_GearRatio( WIMC_S2,0);
```

```
//-----------------------------------------
//
// 각 축의 백래시
//
//-----------------------------------------
WIMC_Set_Backlash( WIMC_X, 0 );
WIMC_Set_Backlash( WIMC_Y, 0);
WIMC_Set_Backlash( WIMC_Z, 0);
WIMC_Set_Backlash( WIMC_U, 0);
WIMC_Set_Backlash( WIMC_V, 0);
WIMC_Set_Backlash( WIMC_W, 0);
WIMC_Set_Backlash( WIMC_S, 0);
WIMC_Set_Backlash( WIMC_S2,0);
WIMC_Enable_Backlash( WIMC_ALL, FALSE);

//-----------------------------------------
//
// 공통축 동작모드
//
//-----------------------------------------
{
        // 선속도 일정제어
        WIMC_Set_CLVAxis( WIMC_XYZ );
        WIMC_Enable_CLV(TRUE);

        // 연속보간동작
        WIMC_Enable_CExe(TRUE);

        // 스텝동작OFF
        WIMC_Enable_StepExe(FALSE);

}

//-----------------------------------------
//
// 보간동작 축 속도설정
//
//-----------------------------------------
{
        // 전체 속도배율수
        WIMC_Set_VelMultiplier(2);

        // 보간속도모드
        WIMC_Set_VelMode( WIMC_VELMODE_LINEAR );

        // 속도오버라이딩
        //WIMC_Set_VelOverride(100);
```

```
        // 초속,가속
        WIMC_Set_StartVel(100);
        WIMC_Set_Acc( 30000 );
        WIMC_Set_Acc2( 60000 );
        WIMC_Set_Dec( 30000 );
        WIMC_Set_Dec2( 60000 );

        // 기본구동속도
        WIMC_Set_DriveVel( 10000 );
}


//-----------------------------------------
//
// 각 축별 공통특성 초기화
//
//-----------------------------------------
{
        UINT8 axis;
        int n=0 ;
        for(n=0; n<WIMC_AXIS_MAX; n++)
        {
                // 축변환
                axis = ToAxis(n);

                // 펄스모드
                WIMC_Set_PulseMode( axis, WIMC_PULSEMODE_PULSEDIR );

                // 펄스 레벨
                WIMC_Set_PulseLevel( axis, 1 );
                WIMC_Set_DirLevel( axis, 1 );

                // 리미트
                WIMC_Set_LowerLimitLevel( axis, 0 );
                WIMC_Set_UpperLimitLevel( axis, 0 );
                WIMC_Enable_LowerLimit( axis, FALSE);
                WIMC_Enable_UpperLimit( axis, FALSE);

                // 소프트리미트
                WIMC_Set_SoftLowerLimit( axis, -1000000000);
                WIMC_Set_SoftUpperLimit( axis, 100000000);
                WIMC_Enable_SoftLowerLimit(axis, FALSE);
                WIMC_Enable_SoftUpperLimit(axis, FALSE);
                WIMC_Set_SoftLimitMode(axis, WIMC_SOFTLIMITMODE_DECSTOP);

                // 속도모드
                WIMC_Set_VelModeAxis(axis, WIMC_VELMODE_LINEAR );
                WIMC_Enable_VelPosAutoTransition( axis, TRUE);
```

```
                WIMC_Enable_StopRotateByAlarm( axis, TRUE);

                // 초속,가속,구동속도
                WIMC_Set_StartVelAxis( axis, 100 );
                WIMC_Set_AccAxis( axis, 30000 );
                WIMC_Set_Acc2Axis( axis, 60000 );
                WIMC_Set_DecAxis( axis, 30000 );
                WIMC_Set_Dec2Axis( axis, 60000 );
                WIMC_Set_DriveVelAxis( axis, 10000 );

                // 원점복귀동작
                WIMC_Set_HomeMode( axis, WIMC_HOMEMODE_0 ); // 홈모드
                WIMC_Set_HomeSensorLevel( axis, 1 ); // 홈센서레벨
                WIMC_Set_HomeDir( axis, 0 );
                WIMC_Enable_HomeServoReset( axis, TRUE); // 홈완료시 자동 리셋출력
                WIMC_Enable_HomeSensorAutoZero( axis, FALSE); // 홈센서발견시 자동 위치 ZERO기능
                WIMC_Enable_HomeReturnAutoZero( axis, FALSE ); // 홈동작완료시 자동 위치 ZERO기능
                WIMC_Set_HomeSeekVel( axis, 100 ); // 홈센서 탐색속도
                WIMC_Set_DwellTimeAxis( axis, 500 ); // 홈동작 드웰시간
                WIMC_Enable_DwellTimeAxis( axis, TRUE);

                // 서보신호 자동동작
                WIMC_Set_ServoReadyLevel( axis, 0 );
                WIMC_Set_ServoAlarmLevel( axis, 0 );
                WIMC_Set_ServoInposLevel( axis, 0 );
                WIMC_Set_ServoUserInLevel( axis, 0 );
                WIMC_Set_ServoOnLevel( axis, 0);
                WIMC_Set_ServoEStopLevel( axis, 0);
                WIMC_Set_ServoResetLevel( axis, 0);
                WIMC_Set_ServoUserOut( axis, 0 );
                WIMC_Enable_ServoReadyAuto( axis, FALSE);
                WIMC_Enable_ServoAlarmAuto( axis, FALSE);
                WIMC_Enable_ServoInposAuto( axis, FALSE);
                WIMC_Enable_ServoOnAuto( axis, FALSE);
                WIMC_Enable_ServoEStopAuto( axis, FALSE);
                WIMC_Enable_ServoResetAuto( axis, FALSE);

                // 서보신호 자동동작 옵션
                WIMC_Set_ServoInposTime( axis, 100);
                WIMC_Enable_ServoInposAlarm( axis, FALSE );
                WIMC_Set_ServoOnMode( axis, WIMC_SERVOONMODE_0 );
                WIMC_Set_ServoPulseTime( axis, 500 );

                // 한텀쉬고 보내기
                Sleep(50);

        }
}
```

```
//-----------------------------------------
//
// 사용자 시스템에 맞는 각 축별 개별특성 초기화
//
//-----------------------------------------
{

}

// ExtIO 활성화
WIMC_Enable_ExtIO(TRUE); // 확장IO 연결
WIMC_Set_GIOMap( WIMC_X | WIMC_V ); // GIO IN/OUT으로 맵핑연결

//-----------------------------------------
//
// 초기화 완료
//
//-----------------------------------------
// 큐명령탈출
WIMC_Leave_QueueCmd();
```

➲ See Also

```
WIMC_Load_wConfigFile();
WIMC_Save_ConfigFile();
```

1.1.8. WIMC_Load_wConfigFile

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Load_wConfigFile
(
              WIMC_CHAR*              pPathName
);
```

➲ Description

모션마스터의 장치 구성 파일을 이용하여 모션보드를 초기화 합니다.(유니코드 방식)
(장치 구성 설정 내용은 아래 Example2 를 참고 하실 수 으며, 코드방식 설정으로도 초기화 가능합니다.(권장))

아래 그림과 같이 "모션마스터>모니터링" 화면의 **엔진구성전용 옵션**을 체크 한후 저장한 장치 구성 파일을 사용해 주십시오.(CNC 설정을 제외하기 위한 옵션)

➲ Parameters

pPathName
파일의 경로 및 이름입니다.(UNICODE 용)
※ 구성설정 파일은 절대 경로를 사용합니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	성공했습니다.
WIMC_FALSE	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

```
WIMC_BOOL        bResult;

bResult = WIMC_Load_wConfigFile( _T("WIMC_Config.ini"));

if ( bResult == WIMC_FALSE)
{
        WIMC_INT32       errCode;
        errCode = WIMC_Get_LastErr();

        printf("WIMC Err> errCode is %d\r\n", errCode);
}
else
{
        printf("WIMC> Success to load a config file\r\n");
```

```
}
```

➲ Example(코드 방식 장치구성 초기화)

```
//-----------------------------------------------------------------------
//      > SOFTWARE RESET
//-----------------------------------------------------------------------
WIMC_Enter_ImCmd();

WIMC_Clr_Queue();
WIMC_Sleep(10);
WIMC_AlarmReset();
WIMC_Sleep(10);
WIMC_Enable_Exe();
WIMC_Sleep(10);
WIMC_Cancel();
WIMC_Sleep(10);

WIMC_Leave_ImCmd();



#define ToAxis(axisNo) (0x01<<axisNo)

//-----------------------------------------
//
// 칩 메인 구성설정
//
//-----------------------------------------
// 큐명령진입
WIMC_Enter_QueueCmd();

// 명령큐 BUSY 25%
WIMC_Set_BusySize( 25 );
// 상태 타이머 레지스터 설정
WIMC_Set_StateTimer(
        WIMC_STATETIMER_100Hz |
        WIMC_STATETIMER_COMPRESSED |
        WIMC_STATETIMER_CHIPSTATUS |
        WIMC_STATETIMER_AXISSTATUS |
        WIMC_STATETIMER_CURPOSITION |
        WIMC_STATETIMER_ENCPOSITION |
        WIMC_STATETIMER_CURVELOCITY |
        WIMC_STATETIMER_GIO |
        WIMC_STATETIMER_USER_R23
        );
// 입출력 포트 사양
WIMC_Set_InportMode( WIMC_INPORTMODE_PULLUP );
WIMC_Set_OutportMode( WIMC_OUTPORTMODE_TOTEMPOLE_PULLUP );
```

```
// 위치결정모드
WIMC_Set_PosMode( WIMC_POSMODE_REL );
// 1회 명령처리수 1~16
WIMC_Set_ProcCmdNo( 4 );

// 알람동작 옵션
WIMC_Set_AlarmOption(
        WIMC_ALARMOPTION_IENDCOMP
        // | WIMC_ALARMOPTION_WDT
        // | WIMC_ALARMOPTION_RECOVERREG
        | WIMC_ALARMOPTION_EMGSTOPIMSTOP
        );
// 비상정지 감속도
WIMC_Set_ImStopDec( 65535 );
// 위치표시모드
WIMC_Set_PosDispMode( WIMC_POSDISPMODE_IGNORE_GEAR );
// 위치모드
WIMC_Set_PosMode( WIMC_POSMODE_REL );
// 상태패킷전송
WIMC_Enable_BusyStatePacket(TRUE);
WIMC_Enable_CompleteStatePacket(TRUE);
WIMC_Enable_InterruptStatePacket(FALSE);
WIMC_Enable_EventStatePacket(FALSE);
// 인터럽트동작모드
WIMC_Set_InterruptMode( WIMC_INTERRUPTMODE_CHANGE_BYTE_R01 );
// 메인속도옵션
WIMC_Set_VelCtrlHz(400);
WIMC_Enable_VelForceConvergence(TRUE);
WIMC_Enable_FFC(TRUE);
WIMC_Set_FFC_PMax(100);
WIMC_Set_DwellTime(0);
// 보간끝점 자동보상
WIMC_Enable_IEndComp(TRUE);

//-----------------------------------------
//
// 각 축의 전자기어비
//
//-----------------------------------------
WIMC_Set_GearRatio( WIMC_X, 0);
WIMC_Set_GearRatio( WIMC_Y, 0);
WIMC_Set_GearRatio( WIMC_Z, 0);
WIMC_Set_GearRatio( WIMC_U, 0);
WIMC_Set_GearRatio( WIMC_V, 0);
WIMC_Set_GearRatio( WIMC_W, 0);
WIMC_Set_GearRatio( WIMC_S, 0);
WIMC_Set_GearRatio( WIMC_S2,0);
```

```
//-----------------------------------------
//
// 각 축의 백래시
//
//-----------------------------------------
WIMC_Set_Backlash( WIMC_X, 0 );
WIMC_Set_Backlash( WIMC_Y, 0);
WIMC_Set_Backlash( WIMC_Z, 0);
WIMC_Set_Backlash( WIMC_U, 0);
WIMC_Set_Backlash( WIMC_V, 0);
WIMC_Set_Backlash( WIMC_W, 0);
WIMC_Set_Backlash( WIMC_S, 0);
WIMC_Set_Backlash( WIMC_S2,0);
WIMC_Enable_Backlash( WIMC_ALL, FALSE);

//-----------------------------------------
//
// 공통축 동작모드
//
//-----------------------------------------
{
        // 선속도 일정제어
        WIMC_Set_CLVAxis( WIMC_XYZ );
        WIMC_Enable_CLV(TRUE);

        // 연속보간동작
        WIMC_Enable_CExe(TRUE);

        // 스텝동작OFF
        WIMC_Enable_StepExe(FALSE);

}

//-----------------------------------------
//
// 보간동작 축 속도설정
//
//-----------------------------------------
{
        // 전체 속도배율수
        WIMC_Set_VelMultiplier(2);

        // 보간속도모드
        WIMC_Set_VelMode( WIMC_VELMODE_LINEAR );

        // 속도오버라이딩
        //WIMC_Set_VelOverride(100);
```

```
        // 초속,가속
        WIMC_Set_StartVel(100);
        WIMC_Set_Acc( 30000 );
        WIMC_Set_Acc2( 60000 );
        WIMC_Set_Dec( 30000 );
        WIMC_Set_Dec2( 60000 );

        // 기본구동속도
        WIMC_Set_DriveVel( 10000 );
}


//-----------------------------------------
//
// 각 축별 공통특성 초기화
//
//-----------------------------------------
{
        UINT8 axis;
        int n=0 ;
        for(n=0; n<WIMC_AXIS_MAX; n++)
        {
                // 축변환
                axis = ToAxis(n);

                // 펄스모드
                WIMC_Set_PulseMode( axis, WIMC_PULSEMODE_PULSEDIR );

                // 펄스 레벨
                WIMC_Set_PulseLevel( axis, 1 );
                WIMC_Set_DirLevel( axis, 1 );

                // 리미트
                WIMC_Set_LowerLimitLevel( axis, 0 );
                WIMC_Set_UpperLimitLevel( axis, 0 );
                WIMC_Enable_LowerLimit( axis, FALSE);
                WIMC_Enable_UpperLimit( axis, FALSE);

                // 소프트리미트
                WIMC_Set_SoftLowerLimit( axis, -1000000000);
                WIMC_Set_SoftUpperLimit( axis, 100000000);
                WIMC_Enable_SoftLowerLimit(axis, FALSE);
                WIMC_Enable_SoftUpperLimit(axis, FALSE);
                WIMC_Set_SoftLimitMode(axis, WIMC_SOFTLIMITMODE_DECSTOP);

                // 속도모드
                WIMC_Set_VelModeAxis(axis, WIMC_VELMODE_LINEAR );
```

```
        WIMC_Enable_VelPosAutoTransition( axis, TRUE);
        WIMC_Enable_StopRotateByAlarm( axis, TRUE);

        // 초속,가속,구동속도
        WIMC_Set_StartVelAxis( axis, 100 );
        WIMC_Set_AccAxis( axis, 30000 );
        WIMC_Set_Acc2Axis( axis, 60000 );
        WIMC_Set_DecAxis( axis, 30000 );
        WIMC_Set_Dec2Axis( axis, 60000 );
        WIMC_Set_DriveVelAxis( axis, 10000 );

        // 원점복귀동작
        WIMC_Set_HomeMode( axis, WIMC_HOMEMODE_0 ); // 홈모드
        WIMC_Set_HomeSensorLevel( axis, 1 ); // 홈센서레벨
        WIMC_Set_HomeDir( axis, 0 );
        WIMC_Enable_HomeServoReset( axis, TRUE); // 홈완료시 자동 리셋출력
        WIMC_Enable_HomeSensorAutoZero( axis, FALSE); // 홈센서발견시 자동 위치 ZERO기능
        WIMC_Enable_HomeReturnAutoZero( axis, FALSE ); // 홈동작완료시 자동 위치 ZERO기능
        WIMC_Set_HomeSeekVel( axis, 100 ); // 홈센서 탐색속도
        WIMC_Set_DwellTimeAxis( axis, 500 ); // 홈동작 드웰시간
        WIMC_Enable_DwellTimeAxis( axis, TRUE);

        // 서보신호 자동동작
        WIMC_Set_ServoReadyLevel( axis, 0 );
        WIMC_Set_ServoAlarmLevel( axis, 0 );
        WIMC_Set_ServoInposLevel( axis, 0 );
        WIMC_Set_ServoUserInLevel( axis, 0 );
        WIMC_Set_ServoOnLevel( axis, 0);
        WIMC_Set_ServoEStopLevel( axis, 0);
        WIMC_Set_ServoResetLevel( axis, 0);
        WIMC_Set_ServoUserOut( axis, 0 );
        WIMC_Enable_ServoReadyAuto( axis, FALSE);
        WIMC_Enable_ServoAlarmAuto( axis, FALSE);
        WIMC_Enable_ServoInposAuto( axis, FALSE);
        WIMC_Enable_ServoOnAuto( axis, FALSE);
        WIMC_Enable_ServoEStopAuto( axis, FALSE);
        WIMC_Enable_ServoResetAuto( axis, FALSE);

        // 서보신호 자동동작 옵션
        WIMC_Set_ServoInposTime( axis, 100);
        WIMC_Enable_ServoInposAlarm( axis, FALSE );
        WIMC_Set_ServoOnMode( axis, WIMC_SERVOONMODE_0 );
        WIMC_Set_ServoPulseTime( axis, 500 );

        // 한텀쉬고 보내기
        Sleep(50);

}
```

```
}
//-----------------------------------------
//
// 사용자 시스템에 맞는 각 축별 개별특성 초기화
//
//-----------------------------------------
{

}

// ExtIO 활성화
WIMC_Enable_ExtIO(TRUE); // 확장IO 연결
WIMC_Set_GIOMap( WIMC_X | WIMC_V ); // GIO IN/OUT으로 맵핑연결

//-----------------------------------------
//
// 초기화 완료
//
//-----------------------------------------
// 큐명령탈출
WIMC_Leave_QueueCmd();
```

➲ See Also

```
WIMC_Load_ConfigFile();
WIMC_Save_ConfigFile();
```

1.1.9. WIMC_Save_ConfigFile

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Save_ConfigFile
(
              char*                   pPathName
);
```

➲ Description

현재의 장치구성 설정을 파일로 저장합니다.
(장치구성 설정 저장 내용은 WIMC_Load_ConfigFile의 Example2 내용을 참고 하실 수 있습니다.)

➲ Parameters

pPathName
파일의 경로 및 이름입니다.
※ 구성설정 파일은 절대 경로를 사용합니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	성공했습니다.
WIMC_FALSE	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

WIMC_Load_ConfigFile()
WIMC_Load_wConfigFile()
WIMC_Save_wConfigFile()

1.1.10. WIMC_Save_wConfigFile

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Save_wConfigFile
(
              WIMC_CHAR*              pPathName
);
```

➲ Description

현재의 장치구성 설정을 파일로 저장합니다.(유니코드용)
(장치구성 설정 저장 내용은 WIMC_Load_ConfigFile의 Example2 내용을 참고 하실 수 있습니다.)

➲ Parameters

pPathName
파일의 경로 및 이름입니다.(UNICODE 용)
※ 구성설정 파일은 절대 경로를 사용합니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	성공했습니다.
WIMC_FALSE	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

WIMC_Load_ConfigFile()
WIMC_Load_wConfigFile()
WIMC_Save_ConfigFile()

1.1.11. WIMC_Is_Link

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_Link( void);
```

➲ Description

모션보드와의 연결상태를 확인합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	모션보드와 Link 되어 있습니다.
WIMC_FALSE	모션보드와 Link 되어있지 않습니다.

➲ Example

➲ See Also

1.1.12. WIMC_Get_DllVer

➲ Syntax

```
void        WIMC_API                WIMC_Get_DllVer
(
            char*                   pVerName,
            WIMC_INT32              numOfChar
);
```

➲ Description

라이브러리의 버전을 읽습니다.(버전정보 최대길이는 512 문자입니다.)

➲ Parameters

pVerName

버전정보를 저장할 버퍼입니다. (WIMC-DLL-V01-R05.20; 라이브러리 버젼마다 다를 수 있습니다.)

bufferSize

읽어들일 문자 수입니다. 버전정보 최대길이는 512 문자입니다.)

➲ Return Value

Return code/value	Description

➲ Example

```
if( 1)
{
        // UNICODE
        WIMC_CHAR               dllVer[512]={0,};
        WIMC_Get_wDllVer( dllVer, 512);

        CString         window_text;
        window_text.Format(_T("%s"), dllVer);

        GetDlgItem( IDC_STATIC_DLL_VER)->SetWindowText( window_text);
}
else
{
        char            dllVer[512]={0,};
        WIMC_Get_DllVer( dllVer, 512);

        CString         window_text;
```

```
        window_text.Format(_T("%S"), dllVer);

        GetDlgItem( IDC_STATIC_DLL_VER)->SetWindowText( window_text);
}
```

➲ See Also

WIMC_Get_DllVer()
WIMC_Get_wDllVer()

1.1.13. WIMC_Get_wDllVer

➲ Syntax

```
void        WIMC_API                WIMC_Get_DllVer
(
            WIMC_CHAR*              pVerName,
            WIMC_INT32              numOfChar
);
```

➲ Description

라이브러리의 버전을 읽습니다.(유니코드용;버전정보 최대길이는 512 문자입니다.)

➲ Parameters

pVerName

버전정보를 저장할 버퍼입니다. (UNICODE용)

(WIMC-DLL-V01-R05.20; 라이브러리 버젼마다 다를 수 있습니다.)

bufferSize

읽어들일 문자 수입니다. 버전정보 최대길이는 512 문자입니다.)

➲ Return Value

Return code/value	Description

➲ Example

```
if( 1)
{
        // UNICODE
        WIMC_CHAR               dllVer[512]={0,};
        WIMC_Get_wDllVer( dllVer, 512);

        CString         window_text;
        window_text.Format(_T("%s"), dllVer);

        GetDlgItem( IDC_STATIC_DLL_VER)->SetWindowText( window_text);
}
else
{
        char            dllVer[512]={0,};
        WIMC_Get_DllVer( dllVer, 512);
```

```
        CString          window_text;
        window_text.Format(_T("%S"), dllVer);

        GetDlgItem( IDC_STATIC_DLL_VER)->SetWindowText( window_text);
}
```

➲ See Also

WIMC_Get_DllVer()
WIMC_Get_wDllVer()

1.1.14. WIMC_Is_Err

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_Err( void);
```

➲ Description

가장 최근에 호출함 함수에서 오류가 있는지 확인합니다.

상세 내용은 WIMC_Get_Last_Err() 함수를 통해 확인 할 수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	최종 호출한 함수에는 오류가 없습니다.
WIMC_FALSE	최종 호출함 함수에 오류가 있습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

WIMC_Get_LastErr();

1.1.15. WIMC_Get_LastErr

➲ Syntax

```
WIMC_INT32     WIMC_API                    WIMC_Get_LastErr( void);
```

➲ Description

마지막에 발생한 에러코드를 반환합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_ERR_NONE(0)	오류가 없습니다.
WIMC_ERR(-1)	유효하지 않은 모션보드 접근(ID,Open)등의 오류입니다.
WIMC_ERR_INVALID_BOARD_ID(-10)	유효하지 않은 모션엔진 ID 입니다.
WIMC_ERR_REALLOCATE_CMD_QUEUE(-11)	소프트웨어 명령큐의 메모리 용량 할당에 실패했습니다.
WIMC_ERR_NO_ROOM_FOR_BOARD(-20)	라이브러리가 제공하는 모션엔진 사용 수를 초과 했습니다.
WIMC_ERR_NO_OPEN_BOARD(-21)	사용할 수 있는 통신 포트(시리얼/TCPIP등)가 없습니다.
WIMC_ERR_OPEN_FAIL(-30)	통신 포트(시리얼/TCPIP등) Open에 실패하였습니다.
WIMC_ERR_ALREADY_OPENED(-31)	통신포트가 이미 Open 되어 있습니다.
WIMC_ERR_SAVE_CONFIG_FAIL(-40)	초기화 구성 파일(Config) 저장에 실패했습니다.
WIMC_ERR_LOAD_CONFIG_FAIL(-41)	초기화 구성 파일(Config) 읽기에 실패했습니다.
WIMC_ERR_INVALID_PARA(-50)	유효하지 않은 파라메터입니다.

➲ Example

```
WIMC_INT32       errCode;
errCode = WIMC_Get_LastErr();

printf("WIMC Err> errCode is %d\r\n", errCode);
```

➲ See Also

1.1.16. WIMC_Enable_ErrBeep

➲ Syntax

```
void        WIMC_API                WIMC_Enable_ErrBeep
(
            WIMC_BOOL               bEnable
);
```

➲ Description

Error 발생 시 Beep음 발생을 설정(Enable/Disable) 합니다.

➲ Parameters

bEnable
WIMC_TRUE : Error Beep Enable
WIMC_FALSE : Error Beep Disable

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

1.1.17. WIMC_Get_EnginVer

➲ Syntax

```
void        WIMC_API                WIMC_Get_EnginVer
(
            char                    *pVerName,
            WIMC_INT32              numOfChar
);
```

➲ Description

모션엔진 버전을 확인합니다.

➲ Parameters

pVerName

버전정보를 저장할 버퍼입니다. (“V02.13 Normal_Engine X6V3”; 모션보드마다 다를 수 있습니다.)

numOfChar

읽어들일 문자 수입니다. (버전정보 최대길이는 512 문자입니다.)

➲ Return Value

Return code/value	Description

➲ Example

```
char     eng_Ver[512] = { 0, };
CString  str_ver = _T(“”);

WIMC_Get_EngineVer(eng_Ver, 512);

str_ver.Format(_T("%S"), eng_Ver);

AfxMessageBox(str_ver);
```

➲ See Also

2. System Command

2.1. 시스템 명령

시스템 명령은 전체동작의 리셋, 실행, 정지 회복, 재기동, 초기화 등의 엔진내부의 기초적인 명령들로 구성됩니다.
아래에 시스템명령군 포함된 명령들을 나열하며 ()안에 관련된 라이브러리 함수도 표기합니다.

(1) RESET 명령 (WIMC_Reset();)

(2) EMGSTOP 명령 (WIMC_Estop(TRUE); or WIMC_Estop(FALSE);)

(3) ENABLE EXE 명령 (WIMC_Enable_Exe();)

(4) ALARM RESET 명령 (WIMC_AlarmReset();)

(5) RESUME 명령 (WIMC_Resume();)

(6) CANCEL 명령 (WIMC_Cancel();)

(7) CLEAR QUEUE 명령 (WIMC_Clr_Queue();)

- 명령큐 내용을 클리어합니다. (관련 라이브러리 함수는 PC상의 소프트웨어Queue도 함께 클리어합니다)

2.1.1. WIMC_Reset

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Reset( void);
```

➲ Description

리셋명령을 전송합니다.

❶ 시스템 명령

시스템 명령은 전체동작의 리셋, 실행, 정지, 회복, 재기동, 초기화 등의 엔진내부의 기초적인 명령들로 구성됩니다.
아래에 시스템명령군 포함된 명령들을 나열하며 ()안에 관련된 라이브러리 함수도 표기합니다.

(1) RESET 명령 (WIMC_Reset();)
(2) EMGSTOP 명령 (WIMC_Estop(TRUE); or WIMC_Estop(FALSE);)
(3) ENABLE EXE 명령 (WIMC_Enable_Exe();)
(4) ALARM RESET 명령 (WIMC_AlarmReset();)
(5) RESUME 명령 (WIMC_Resume();)
(6) CANCEL 명령 (WIMC_Cancel();)
(7) CLEAR QUEUE 명령 (WIMC_Clr_Queue();)
- 명령큐 내용을 클리어합니다. (관련 라이브러리 함수는 PC상의 소프트웨어Queue도 함께 클리어합니다)

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

2.1.2. WIMC_EStop

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_EStop
(
              WIMC_BOOL               bImStop
);
```

➲ Description

EMG STOP명령(비상정지 명령에 의해 알람상태를 만들어 비상실행상태로 전환시킵니다.)

즉시 정지명령과 감속정지명령 중 선택되어 실행하는 명령입니다.

※라이브러리 함수 WIMC_EStop()함수는 내부적으로 WIMC_ImStop()과 WIMC_DecStop()함수에 의해 구성됩니다.
▶WIMC_EStop(TRUE); // WIMC_ImStop()와 동일합니다.
▶WIMC_EStop(FALSE); // WIMC_DecStop()와 동일합니다.

❶ 즉시정지 명령 [IMSTOP: Immediate Stop]

이 명령은 현재 동작을 급히 정지시키는 명령입니다. 즉시정지 명령이 실행되면 모션 엔진은 급정지 혹은 급감속정지를 실행하여 동작이 정지하며 알람이 발생되고, 비상실행상태가 됩니다.
비상실행상태에서는 명령큐의 명령은 실행되지 않으며 오직 즉시실행명령만 실행가능합니다.
즉시정지 명령에 의해서 펄스를 즉각 중단시키는 급정지방식과 1차 선형감속형식으로 감속하여 정지하는 급감속정지방식의 선택은 즉시정지 감속도값으로 결정됩니다. 이 값이 0 이면 급정지하게 되고 0이 아니면 이 감속계수값으로 선형 감속정지하게 됩니다.

즉시정지 명령에서 회복하기 위해서는 먼저 [ALARM RESET]명령을 통해서 정지명령에 의한 알람상태를 해제시킵니다. 그리고 명령큐의 실행동작을 가능하게 하기 위해 [ENABLE EXE]명령을 전송합니다.
마지막으로 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

❷ 감속 정지 명령 [DECSTOP : Deceleration Stop]

현재 모션동작의 속도모드방식으로 자연스럽게 감속시켜 초속도에 도달하면 정지시킵니다. 즉시정지방식보다 자연스럽게 정지시키지만 정지시간이 길어질 수 있어 정지거리 역시 길어집니다. 완전히 정지한 후 알람이 발생되고, 비상실행상태가 됩니다.
감속이 완료되지 않은 상태에서 [RESUME]명령이나 [CANCEL]명령을 전송하면 감속정지명령이 취소되어 다시 가속하여 명령을 정상으로 실행하게 됩니다.

※ EMG STOP명령의 라이브러리 함수는 실제 내부적으로 즉시정지명령과 감속정지명령 중 선택되어 실행하는 명령입니다. WIMC_EmgStop(TRUE) 와 같이 인자값이 TRUE이면 즉시정지명령이 전송되고 FALSE이면 감속정지명령이 전송됩니다.

감속정지 명령에서 회복방식 역시 즉시정지 명령에서의 회복방식과 동일합니다. 즉 [ALARM RESET]명령을 통해서 정지명령에 의한 알람상태를 해제시킨 후 명령큐의 실행동작을 가능하게 하기 위해 [ENABLE EXE]명령을 전송합니다. 마지막으로 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시

즉시실행명령으로 전송하여야 합니다.

➲ Parameters

bImStop

WIMC_TRUE : 즉시정지명령이 전송됩니다.

WIMC_FALSE : 감속정지명령이 전송됩니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

2.1.3. WIMC_ImStop

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_ImStop( void);
```

➲ Description

즉시정지 명령으로 즉시정지동작 실행 및 알람을 발생시키며 정지 후 비상실행상태가 됩니다.

❶ 즉시정지 명령 [IMSTOP: Immediate Stop]

이 명령은 현재 동작을 급히 정지시키는 명령입니다. 즉시정지 명령이 실행되면 모션 엔진은 급정지 혹은 급감속정지를 실행하여 동작이 정지하며 알람이 발생되고, 비상실행상태가 됩니다.
비상실행상태에서는 명령큐의 명령은 실행되지 않으며 오직 즉시실행명령만 실행가능합니다.
즉시정지 명령에 의해서 펄스를 즉각 중단시키는 급정지방식과 1차 선형감속형식으로 감속하여 정지하는 급감속정지방식의 선택은 즉시정지 감속도값으로 결정됩니다. 이 값이 0 이면 급정지하게 되고 0이 아니면 이 감속계수값으로 선형 감속정지하게 됩니다.

즉시정지 명령에서 회복하기 위해서는 먼저 [ALARM RESET]명령을 통해서 정지명령에 의한 알람상태를 해제시킵니다. 그리고 명령큐의 실행동작을 가능하게 하기 위해 [ENABLE EXE]명령을 전송합니다.
마지막으로 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

❷ 감속 정지 명령 [DECSTOP : Deceleration Stop]

현재 모션동작의 속도모드방식으로 자연스럽게 감속시켜 초속도에 도달하면 정지시킵니다. 즉시정지방식보다 자연스럽게 정지시키지만 정지시간이 길어질 수 있어 정지거리 역시 길어집니다. 완전히 정지한 후 알람이 발생되고, 비상실행상태가 됩니다.
감속이 완료되지 않은 상태에서 [RESUME]명령이나 [CANCEL]명령을 전송하면 감속정지명령이 취소되어 다시 가속하여 명령을 정상으로 실행하게 됩니다.

※ EMG STOP명령의 라이브러리 함수는 실제 내부적으로 즉시정지명령과 감속정지명령 중 선택되어 실행하는 명령입니다. WIMC_EmgStop(TRUE) 와 같이 인자값이 TRUE이면 즉시정지명령이 전송되고 FALSE이면 감속정지명령이 전송됩니다.

감속정지 명령에서 회복방식 역시 즉시정지 명령에서의 회복방식과 동일합니다. 즉 [ALARM RESET]명령을 통해서 정지명령에 의한 알람상태를 해제시킨 후 명령큐의 실행동작을 가능하게 하기 위해 [ENABLE EXE]명령을 전송합니다.
마지막으로 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

2.1.4. WIMC_DecStop

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_DecStop( void);
```

➲ Description

감속정지동작을 실행하며 알람을 발생시며 정지 후 비상실행상태가 됩니다.

❶ 즉시정지 명령 [IMSTOP: Immediate Stop]

이 명령은 현재 동작을 급히 정지시키는 명령입니다. 즉시정지 명령이 실행되면 모션 엔진은 급정지 혹은 급감속정지를 실행하여 동작이 정지하며 알람이 발생되고, 비상실행상태가 됩니다.
비상실행상태에서는 명령큐의 명령은 실행되지 않으며 오직 즉시실행명령만 실행가능합니다.
즉시정지 명령에 의해서 펄스를 즉각 중단시키는 급정지방식과 1차 선형감속형식으로 감속하여 정지하는 급감속정지방식의 선택은 즉시정지 감속도값으로 결정됩니다. 이 값이 0 이면 급정지하게 되고 0이 아니면 이 감속계수값으로 선형 감속정지하게 됩니다.

즉시정지 명령에서 회복하기 위해서는 먼저 [ALARM RESET]명령을 통해서 정지명령에 의한 알람상태를 해제시킵니다. 그리고 명령큐의 실행동작을 가능하게 하기 위해 [ENABLE EXE]명령을 전송합니다.
마지막으로 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

❶ 감속 정지 명령 [DECSTOP : Deceleration Stop]

현재 모션동작의 속도모드방식으로 자연스럽게 감속시켜 초속도에 도달하면 정지시킵니다. 즉시정지방식보다 자연스럽게 정지시키지만 정지시간이 길어질 수 있어 정지거리 역시 길어집니다. 완전히 정지한 후 알람이 발생되고, 비상실행상태가 됩니다.
감속이 완료되지 않은 상태에서 [RESUME]명령이나 [CANCEL]명령을 전송하면 감속정지명령이 취소되어 다시 가속하여 명령을 정상으로 실행하게 됩니다.

※ EMG STOP명령의 라이브러리 함수는 실제 내부적으로 즉시정지명령과 감속정지명령 중 선택되어 실행하는 명령입니다. WIMC_EmgStop(TRUE) 와 같이 인자값이 TRUE이면 즉시정지명령이 전송되고 FALSE이면 감속정지명령이 전송됩니다.

감속정지 명령에서 회복방식 역시 즉시정지 명령에서의 회복방식과 동일합니다. 즉 [ALARM RESET]명령을 통해서 정지명령에 의한 알람상태를 해제시킨 후 명령큐의 실행동작을 가능하게 하기 위해 [ENABLE EXE]명령을 전송합니다.
마지막으로 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

2.1.5. WIMC_AlarmReset

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_AlarmReset( void);
```

➲ Description

알람리셋 명령입니다.

명령을 중지시킨 비정상적인 내외부적인 원인을 제거하거나 해제시킨 후 알람리셋(ALARM RESET)명령을 전송하여 알람을 해제시킨다면 정상적으로 이전 중지된 명령의 동작을 재개시킬 수 있게 됩니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

아래에 비상실행상태와 관련된 명령입니다.

```
WIMC_EStop();
WIMC_AlarmReset();
WIMC_Enable_Exe();
WIMC_Resume();
WIMC_Cancel();
WIMC_Reset();
```

2.1.6. WIMC_Enable_Exe

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Enable_Exe( void);
```

➲ Description

명령큐의 실행을 재개하도록 합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

2.1.7. WIMC_Pause

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Pause( void);
```

➲ Description

일시정지 명령입니다.

ʊ 일시정지 명령

일시 정지 명령은 DECSTOP방식으로 감속시켜 정지하지만, 알람이 발생하지 않으며 비상실행상태로 전환됩니다.
완료되지 않은 상태에서 [RESUME]명령이나 [CANCEL]명령을 전송하면 감속정지명령이 취소되어 다시 가속하여 명령을 정상으로 실행하게 됩니다.

일시 정지 명령에서 회복방식은 즉시정지 명령에서의 회복방식에서 [ALARM RESET]과정이 없을 때와 동일합니다. 즉 [ENABLE EXE]명령을 통해서 명령큐 실행이 가능토록 한 후 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

2.1.8. WIMC_Resume

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Resume( void);
```

➲ Description

동작재개명령을 전송합니다.

동작재개명령[RESUME]은 아래의 다양한 기능을 수행합니다.
① 비상실행상태를 정상실행상태로 되돌리고 정상실행에서 중지된 명령을 재개시킵니다.
② 딜레이명령에 의한 Delay상태를 해제합니다.
③ 외부동기신호 대기명령에 의한 Wait상태를 해제합니다.
④ 스텝실행상태에서 명령큐의 명령을 하나 읽어들여 실행하도록 허락합니다.
⑤ 강제감속상태에서 완전히 정지하지 않았을 때 강제감속상태를 해제합니다.

각 상태가 중첩되어 있다면 [RESUME]명령은 ① > ② > ③ > ④ > ⑤ 의 순서로 우선 실행됩니다. 그러므로 [RESUME]명령은 항상 비상실행상태를 먼저 해제하려 합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_StepExeOn(); // STEP ON 명령전송
WIMC_StepExeOff(); // STEP OFF 명령전송
WIMC_Is_StepExe(); // 현재 스텝실행상태인지 조사
WMC_Resume();    // RESUME 명령
WIMC_Cancel();   // CANCEL명령
```

2.1.9. WIMC_Cancel

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Cancel( void);
```

➲ Description

동작 취소명령입니다.

ʊ 동작취소명령

동작취소명령 [CANCEL]명령은 [RESUME]명령과 비슷하게 각각의 실행상태에서 서로 다른 기능을 수행합니다.
① 비상실행상태를 정상실행상태로 되돌리고 정상실행에서 중지된 명령을 취소하고 명령큐의 다음명령을 실행하도록 지시합니다.
② 딜레이명령에 의한 Delay상태를 해제합니다. ([RESUME]명령과 동일)
③ 외부동기신호 대기명령에 의한 Wait상태를 해제합니다.([RESUME]명령과 동일)
④ 스텝실행상태에서 [CANCEL]명령은 [RESUME]명령과 같이 명령큐의 명령을 하나 읽어들이지만 실행하지 않고 버리도록 지시합니다.
⑤ 강제감속상태에서 완전히 정지하지 않았을 때 강제감속상태를 해제합니다.([RESUME]명령과 동일)

각 상태가 중첩되어 있다면 [RESUME]명령과 동일하게 ① > ② > ③ > ④ > ⑤ 의 순서로 우선 실행됩니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

아래에 비상실행상태와 관련된 명령입니다.

```
WIMC_EStop();
WIMC_AlarmReset();
WIMC_Enable_Exe();
WIMC_Resume();
WIMC_Cancel();
WIMC_Reset();
```

3. System State

3.1. System State

3.1.1. WIMC_Is_EStop

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_EStop( void);
```

➲ Description

비상실행 상태를 확인합니다.

ʊ 즉시정지 명령 [IMSTOP: Immediate Stop]

이 명령은 현재 동작을 급히 정지시키는 명령입니다. 즉시정지 명령이 실행되면 모션 엔진은 급정지 혹은 급감속정지를 실행하여 동작이 정지하며 알람이 발생되고, 비상실행상태가 됩니다.
비상실행상태에서는 명령큐의 명령은 실행되지 않으며 오직 즉시실행명령만 실행가능합니다.
즉시정지 명령에 의해서 펄스를 즉각 중단시키는 급정지방식과 1차 선형감속형식으로 감속하여 정지하는 급감속정지방식의 선택은 즉시정지 감속도값으로 결정됩니다. 이 값이 0 이면 급정지하게 되고 0이 아니면 이 감속계수값으로 선형 감속정지하게 됩니다.

즉시정지 명령에서 회복하기 위해서는 먼저 [ALARM RESET]명령을 통해서 정지명령에 의한 알람상태를 해제시킵니다. 그리고 명령큐의 실행동작을 가능하게 하기 위해 [ENABLE EXE]명령을 전송합니다.
마지막으로 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

ʊ 감속 정지 명령 [DECSTOP : Deceleration Stop]

현재 모션동작의 속도모드방식으로 자연스럽게 감속시켜 초속도에 도달하면 정지시킵니다. 즉시정지방식보다 자연스럽게 정지시키지만 정지시간이 길어질 수 있어 정지거리 역시 길어집니다. 완전히 정지한 후 알람이 발생되고, 비상실행상태가 됩니다.
감속이 완료되지 않은 상태에서 [RESUME]명령이나 [CANCEL]명령을 전송하면 감속정지명령이 취소되어 다시 가속하여 명령을 정상으로 실행하게 됩니다.

※ EMG STOP명령의 라이브러리 함수는 실제 내부적으로 즉시정지명령과 감속정지명령 중 선택되어 실행하는 명령입니다. WIMC_EmgStop(TRUE) 와 같이 인자값이 TRUE이면 즉시정지명령이 전송되고 FALSE이면 감속정지명령이 전송됩니다.

감속정지 명령에서 회복방식 역시 즉시정지 명령에서의 회복방식과 동일합니다. 즉 [ALARM RESET]명령을 통해서 정지명령에 의한 알람상태를 해제시킨 후 명령큐의 실행동작을 가능하게 하기 위해 [ENABLE EXE]명령을 전송합니다.
마지막으로 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	비상실행상태입니다.
WIMC_FALSE	비상실행상태가 아닙니다.

➲ Example

➲ See Also

3.1.2. WIMC_Is_Pause

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_Pause( void);
```

➲ Description

일시정지 상태를 확인합니다.

ʊ 일시정지 명령

일시 정지 명령은 DECSTOP방식으로 감속시켜 정지하지만, 알람이 발생하지 않으며 비상실행상태로 전환됩니다.
완료되지 않은 상태에서 [RESUME]명령이나 [CANCEL]명령을 전송하면 감속정지명령이 취소되어 다시 가속하여 명령을 정상으로 실행하게 됩니다.

일시 정지 명령에서 회복방식은 즉시정지 명령에서의 회복방식에서 [ALARM RESET]과정이 없을 때와 동일합니다. 즉 [ENABLE EXE]명령을 통해서 명령큐 실행이 가능토록 한 후 [RESUME]명령을 전송하면 정지된 명령은 정지시점 이후로 동작을 재개하게 됩니다. [RESUME]명령 대신 [CANCEL]명령을 전송하면 정지된 명령은 취소되고 다음의 명령큐 명령이 실행됩니다. 이때 명령들은 반드시 즉시실행명령으로 전송하여야 합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	일시정지 상태입니다.
WIMC_FALSE	일시정지 상태가 아닙니다.

➲ Example

➲ See Also

3.1.3. WIMC_Is_Alarm

.

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_Alarm( void);
```

➲ Description

모션엔진의 알람을 확인합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	알람 상태입니다.
WIMC_FALSE	알람 상태가 아닙니다.

➲ Example

➲ See Also

3.1.4. WIMC_Is_EmgExe

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_EmgExe( void);
```

➲ Description

비상실행 상태를 확인합니다.

❶ 비상실행상태란?

모션엔진은 기본적으로 정상실행상태에서 모든 동작을 실행합니다. 그러나 단순통지상태의 알람을 제외한 알람이나 일시정지동작이 실행된다면 현재 실행 동작은 정지되고 비상실행상태가 됩니다.

비상실행상태란 동작 중지형 알람이 발생되었을 때 현재 실행 동작명령을 멈추고, 명령큐의 명령실행을 차단한채 오직 즉시실행명령 통로만을 열어 상위제어기로부터 새로운 명령을 대기하는 상태를 말합니다. 현재 정지된 명령은 내부적으로 백업되어 동작이 재개될 때(resume) 복원됩니다.
비상실행상태는 WIMC_Is_EmgExe() 라이브러리 함수로써 확인할 수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	비상실행 상태입니다.
WIMC_FALSE	비상실행 상태가 아닙니다.

➲ Example

➲ See Also

3.1.5. WIMC_Is_EnableExe

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_EnableExe( void);
```

➲ Description

모션엔진의 큐명령이 실행가능한 상태임을 확인합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	큐명령 실행 가능한 상태입니다.
WIMC_FALSE	큐명령 실행 가능한 상태가 아닙니다.

➲ Example

➲ See Also

4. Main / Aux com port

4.1. Main com

4.1.1. WIMC_Set_MainBaudrate

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_MainBaudrate
(
              WIMC_INT32              baudrate
);
```

➲ Description

모션엔진 주 연결 포트의 통신속도를 설정합니다.

※ 모션엔진에 따라(ex.WIMC 엔진) 내부 명령에 의한 통신설정 변경을 허용하지 않을 수 있습니다. 관련 매뉴얼을 참고 하실 수 있습니다.

➲ Parameters

baudrate

설정가능한 속도는 115200/ 230400/ 460800/ 921600 입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

4.1.2. WIMC_Get_MainBaudrate

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_MainBaudrate( void);
```

➲ Description

모션엔진 주 연결 포트의 통신속도 설정을 읽습니다.

➲ Parameters

➲ Return Value

통신속도 115200/ 230400/ 460800/ 921600 중 하나를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

4.2. Aux com

4.2.1. WIMC_Set_AuxBaudrate

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_AuxBaudrate
(
              WIMC_INT32              baudrate
);
```

➲ Description

모션엔진 보조 연결 포트의 통신속도를 설정합니다.

※ 인터페이스 포트는 Main(주연결포트)와 Aux(보조연결포트) 2 가지로 구성되어 있지만 모션엔진에 따라(ex.WIMC 엔진) Main 포트 하나만 제공되거나 내부 명령에 의한 통신설정 변경을 허용하지 않을 수 있습니다. 관련 매뉴얼을 참고 하실 수 있습니다.

➲ Parameters

baudrate

설정가능한 속도는 115200/ 230400/ 460800/ 921600 입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

4.2.2. WIMC_Get_AuxBaudrate

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_AuxBaudrate( void);
```

➲ Description

모션엔진 보조 연결 포트의 통신속도설정을 읽습니다.

※ 인터페이스 포트는 Main(주연결포트)와 Aux(보조연결포트) 2 가지로 구성되어 있지만 모션엔진에 따라(ex.WIMC 엔진) Main 포트 하나만 제공합니다. 관련 매뉴얼을 참고 하실 수 있습니다.

➲ Parameters

➲ Return Value

통신속도 115200/ 230400/ 460800/ 921600 중 하나를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

4.3. Aux Com Options

4.3.1. WIMC_Enable_AuxPortAllCommand

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Enable_AuxPortAllCommand
(
              WIMC_BOOL                 bEnable
);
```

➲ Description

보조포트를 통한 모든 명령지령을 enable 또는 disable 하도록 합니다.

※ 인터페이스 포트는 Main(주연결포트)와 Aux(보조연결포트) 2 가지로 구성되어 있지만 모션엔진에 따라(ex.WIMC 엔진) Main 포트 하나만 제공합니다. 관련 매뉴얼을 참고 하실 수 있습니다.

(1) 보조포트 통신속도 설정
- 115,200bps, 230,400bps, 460,800bps, 921600bps 중 하나를 설정합니다.

(2) 보조포트 활성화
- 이 기능이 활성화 되어야만 보조포트를 통해서 명령을 수신할 수 있습니다.
- 이 기능은 모션마스터프로그램에서 설정되지 않습니다. 리셋시 자동으로 활성화 됩니다.

(3) 상태타이머 정보전송 활성화
- 이 기능이 활성화 되면 보조포트로도 상태타이머의 정보가 지정된 주기로 전송되게 됩니다.

(4) 상태패킷 전송 활성화
- 이 기능이 활성화 되면 보조포트로 BUSY상태패킷, COMPLETE상태패킷, INT상태패킷이 전송됩니다.

(5) 모든 명령허용
- 이 기능이 활성화 되면 보조포트를 통해 모든 명령을 전송할 수 있습니다.
- 이 기능이 비활성화되면 보조포트를 통해서 전송할 수 있는 명령은 즉시실행명령입니다.

※ 보조포트 활성화 옵션에 관계없이 보조포트가 동작되는 모션엔진은 즉시 실행명령 중 시스템명령은 항상 허용됩니다.

<리셋시 설정상태>
보조포트통신속도 = 115,200bps (엔진종류에 따라 달라집니다.)
보조포트활성화 기능 = ON
보조포트 상태타이머 정보전송기능 = OFF
보조포트 상태패킷 전송기능 = ON
보조포트 모든 명령허용기능 = ON

➲ Parameters

bEnable

WIMC_TRUE : 보조포트를 통한 모든 명령지령 enable.
WIMC_FALSE : 보조포트를 통한 모든 명령지령 disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_AuxBaudrate();           // 보조포트의 통신속도를 설정합니다.
WIMC_Enable_AuxPortStateTimer();  // 보조포트를 통한 상태타이머정보전송 기능을 설정합니다.
WIMC_Enable_AuxPortStatePacket();         // 보조포트를 통한 상태패킷전송 기능을 설정합니다.
WIMC_Enable_AuxPortAllCommand();  // 보조포트를 통한 모든 명령지령이 가능한 기능을 설정합니다.
```

4.3.2. WIMC_Enable_AuxPortStatePacket

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Enable_AuxPortStatePacket
(
             WIMC_BOOL               bEnable
);
```

➲ Description

보조포트를 통한 상태패킷전송 기능을 Enable/Disable합니다.

※ 인터페이스 포트는 Main(주연결포트)와 Aux(보조연결포트) 2 가지로 구성되어 있지만 모션엔진에 따라(ex.WIMC 엔진) Main 포트 하나만 제공합니다. 관련 매뉴얼을 참고 하실 수 있습니다.

(1) 보조포트 통신속도 설정
- 115,200bps, 230,400bps, 460,800bps, 921600bps 중 하나를 설정합니다.

(2) 보조포트 활성화
- 이 기능이 활성화 되어야만 보조포트를 통해서 명령을 수신할 수 있습니다.
- 이 기능은 모션마스터프로그램에서 설정되지 않습니다. 리셋시 자동으로 활성화 됩니다.

(3) 상태타이머 정보전송 활성화
- 이 기능이 활성화 되면 보조포트로도 상태타이머의 정보가 지정된 주기로 전송되게 됩니다.

(4) 상태패킷 전송 활성화
- 이 기능이 활성화 되면 보조포트로 BUSY상태패킷, COMPLETE상태패킷, INT상태패킷이 전송됩니다.

(5) 모든 명령허용
- 이 기능이 활성화 되면 보조포트를 통해 모든 명령을 전송할 수 있습니다.
- 이 기능이 비활성화되면 보조포트를 통해서 전송할 수 있는 명령은 즉시실행명령입니다.

※ 보조포트 활성화 옵션에 관계없이 보조포트가 동작되는 모션엔진은 즉시 실행명령 중 시스템명령은 항상 허용됩니다.

<리셋시 설정상태>
보조포트통신속도 = 115,200bps (엔진종류에 따라 달라집니다.)
보조포트활성화 기능 = ON
보조포트 상태타이머 정보전송기능 = OFF
보조포트 상태패킷 전송기능 = ON
보조포트 모든 명령허용기능 = ON

➲ Parameters

bEnable

WIMC_TRUE : 보조포트를 통한 모든 상태패킷 전송기능 Enable.
WIMC_FALSE : 보조포트를 통한 모든 상태패킷 전송기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_AuxBaudrate();           // 보조포트의 통신속도를 설정합니다.
WIMC_Enable_AuxPortStateTimer();  // 보조포트를 통한 상태타이머정보전송 기능을 설정합니다.
WIMC_Enable_AuxPortStatePacket();         // 보조포트를 통한 상태패킷전송 기능을 설정합니다.
WIMC_Enable_AuxPortAllCommand();  // 보조포트를 통한 모든 명령지령이 가능한 기능을 설정합니다.
```

4.3.3. WIMC_Enable_AuxPortStateTimer

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Enable_AuxPortStateTimer
(
             WIMC_BOOL               bEnable
);
```

➲ Description

보조포트를 통한 상태타이머정보 전송 기능을 Enable/Disable 합니다.

※ 인터페이스 포트는 Main(주연결포트)와 Aux(보조연결포트) 2 가지로 구성되어 있지만 모션엔진에 따라(ex.WIMC 엔진) Main 포트 하나만 제공합니다. 관련 매뉴얼을 참고 하실 수 있습니다.

(1) 보조포트 통신속도 설정
- 115,200bps, 230,400bps, 460,800bps, 921600bps 중 하나를 설정합니다.

(2) 보조포트 활성화
- 이 기능이 활성화 되어야만 보조포트를 통해서 명령을 수신할 수 있습니다.
- 이 기능은 모션마스터프로그램에서 설정되지 않습니다. 리셋시 자동으로 활성화 됩니다.

(3) 상태타이머 정보전송 활성화
- 이 기능이 활성화 되면 보조포트로도 상태타이머의 정보가 지정된 주기로 전송되게 됩니다.

(4) 상태패킷 전송 활성화
- 이 기능이 활성화 되면 보조포트로 BUSY상태패킷, COMPLETE상태패킷, INT상태패킷이 전송됩니다.

(5) 모든 명령허용
- 이 기능이 활성화 되면 보조포트를 통해 모든 명령을 전송할 수 있습니다.
- 이 기능이 비활성화되면 보조포트를 통해서 전송할 수 있는 명령은 즉시실행명령입니다.

※ 보조포트 활성화 옵션에 관계없이 보조포트가 동작되는 모션엔진은 즉시 실행명령 중 시스템명령은 항상 허용됩니다.

<리셋시 설정상태>
보조포트통신속도 = 115,200bps (엔진종류에 따라 달라집니다.)
보조포트활성화 기능 = ON
보조포트 상태타이머 정보전송기능 = OFF
보조포트 상태패킷 전송기능 = ON
보조포트 모든 명령허용기능 = ON

➲ Parameters

bEnable

WIMC_TRUE : 보조포트를 통한 모든 상태타이머정보 전송기능 Enable.
WIMC_FALSE : 보조포트를 통한 모든 상태타이머정보 전송기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_AuxBaudrate();           // 보조포트의 통신속도를 설정합니다.
WIMC_Enable_AuxPortStateTimer();  // 보조포트를 통한 상태타이머정보전송 기능을 설정합니다.
WIMC_Enable_AuxPortStatePacket();         // 보조포트를 통한 상태패킷전송 기능을 설정합니다.
WIMC_Enable_AuxPortAllCommand();  // 보조포트를 통한 모든 명령지령이 가능한 기능을 설정합니다.
```

4.4. Update Baudrate

4.4.1. WIMC_Update_Baudrate

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Update_Baudrate( void);
```

➲ Description

설정된 통신속도로 모션엔진 통신속도를 변경합니다.

※ 모션엔진에 따라(ex. X6V3 엔진) 내부 명령에 의한 통신설정 변경을 허용하지 않을 수 있습니다. 관련 매뉴얼을 참고 하실 수 있습니다.

➲ Parameters

baudrate

설정가능한 속도는 115200/ 230400/ 460800/ 921600 입니다.

➲ Return Value

Return code/value	Description

➲ Example

```
// 모션엔진 통신속도 변경 예
WIMC_Set_MainBaudrate(921600);
WIMC_Set_AuxBaudrate(115200);
WIMC_Update_Baudrate();
```

➲ See Also

5. Estop

5.1. Emergency Stop

5.1.1. WIMC_Set_ImStopDec

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ImStopDec
(
              WIMC_INT32              pps
);
```

➲ Description

즉시정지 감속도를 설정합니다. 정지 후 비상실행상태가 됩니다.

EStop Dec(Deceleration) = (0~65535)

알람에 의한 비상정지동작의 급정지 감속도(급감속도)값을 설정합니다. 급감속도가 0 이면 펄스를 즉시차단하는 급정지동작이 실행되고 0이 아니면 현재 속도모드에 관계없이 1차가감속방식의 감속정지동작이 실행됩니다.

`WIMC_Set_ImStopDec(dec); // dec = 0이면 급정지, dec ≠ 0 이면 dec감속도값으로 급감속정지`

※급감속도값은 모션시스템이 허용하는 최대값을 설정하는 것이 바람직합니다. 이 값이 너무 작으면 비상정지동작에서 매우 느리게 반응하게 되어 모션시스템을 위험에 빠뜨릴 수 있습니다.

<리셋시 설정>
보간종점알람 설정, WDT알람 해제, 레지스터복구기능 해제
EStop신호에 의한 즉시정지해제(감속정지방식)
비상정지 급감속계수 = 0

➲ Parameters

pps
값이 0이면 급정지, 0이 아니면 감속도값으로 급감속정지합니다.
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

5.1.2. WIMC_Get_ImStopDec

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ImStopDec( void);
```

➲ Description

즉시정지 감속도를 설정값을 읽습니다.

EStop Dec(Deceleration)

알람에 의한 비상정지동작의 급정지 감속도(급감속도)설정값입니다. 급감속도가 0 이면 펄스를 즉시차단하는 급정지동작이 실행되고 0이 아니면 현재 속도모드에 관계없이 1차가감속방식의 감속정지동작이 실행됩니다.

<리셋시 설정>
보간종점알람 설정, WDT알람 해제, 레지스터복구기능 해제
EStop신호에 의한 즉시정지해제(감속정지방식)
비상정지 급감속계수 = 0

➲ Parameters

pps

값이 0이면 급정지, 0이 아니면 감속도값으로 급감속정지합니다.
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

즉시정지 감속도 값을 반환합니다.(0~65535)

Return code/value	Description
WIMC_ERR	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

5.1.3. WIMC_Set_ExtEmgStopMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ExtEmgStopMode
(
              WIMC_INT32              mode
);
```

➲ Description

모션엔진의 외부신호선인 ESTOP신호의 활성화에 의한 정지모드를 설정한다.

❶ 모션엔진의 외부신호선인 ESTOP 신호의 활성화에 의한 정지

외부신호선 ESTOP신호에 의한 정지방식은 다음의 라이브러리 함수를 통해서 즉시정지동작과 감속정지동작 중 하나를 선택할 수 있습니다.

```
WIMC_Set_ExtEmgStopMode( WIMC_EXTEMGSTOPMODE_IMSTOP );
WIMC_Set_ExtEmgStopMode( WIMC_EXTEMGSTOPMODE_DECSTOP);
```

즉시정지동작을 선택한다면 WIMC_Set_ImStopDec()함수에 의해 급정지와 급감속정지방식을 선택할 수 있습니다.

감속정지동작은 현재 실행중인 동작의 속도모드 방식으로 그대로 감속하여 정지하게 되므로 안정되게 정지할 수 있지만 정지거리가 즉시정지동작에 비해 커질 수 있습니다.

정지 후 비상실행상태가 됩니다.

➲ Parameters

mode

WIMC_EXTEMGSTOPMODE_IMSTOP : 즉시정지동작입니다.

WIMC_EXTEMGSTOPMODE_DECSTOP : 감속정지동작입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

5.1.4. WIMC_Get_ExtEmgStopMode

➲ Syntax

```
WIMC_INT32      WIMC_API                WIMC_Get_ExtEmgStopMode( void);
```

➲ Description

모션엔진의 외부신호선인 ESTOP신호의 활성화에 의한 정지모드를 설정한다.

❶ 모션엔진의 외부신호선인 ESTOP 신호의 활성화에 의한 정지

외부신호선 ESTOP신호에 의한 정지방식은 다음의 라이브러리 함수를 통해서 즉시정지동작과 감속정지동작 중 하나를 선택할 수 있습니다.

```
WIMC_Set_ExtEmgStopMode( WIMC_EXTEMGSTOPMODE_IMSTOP );
WIMC_Set_ExtEmgStopMode( WIMC_EXTEMGSTOPMODE_DECSTOP);
```

즉시정지동작을 선택한다면 WIMC_Set_ImStopDec()함수에 의해 급정지와 급감속정지방식을 선택할 수 있습니다.

감속정지동작은 현재 실행중인 동작의 속도모드 방식으로 그대로 감속하여 정지하게 되므로 안정되게 정지할 수 있지만 정지거리가 즉시정지동작에 비해 커질 수 있습니다.

정지 후 비상실행상태가 됩니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_EXTEMGSTOPMODE_IMSTOP	즉시정지 모드입니다.
WIMC_EXTEMGSTOPMODE_DECSTOP	감속정지 모드입니다.
WIMC_ERR	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

6. Command ID

6.1. Command ID

6.1.1. WIMC_Set_CmdID

➲ Syntax

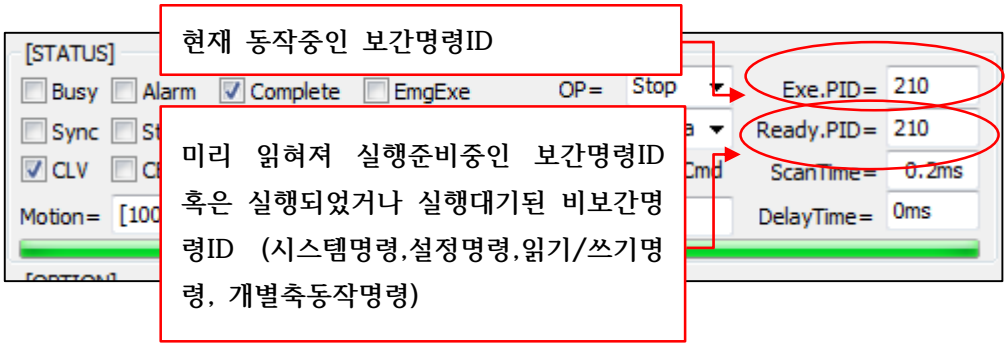
```
WIMC_BOOL     WIMC_API                WIMC_Set_CmdID
(
              WIMC_INT32              id
);
```

➲ Description

명령 ID를 설정합니다.

❶ 명령 ID 설정

모션엔진에 전달되는 모든 명령은 내부적으로 8비트의 명령ID값을 갖고 있습니다.
이 명령ID값은 아래의 모션마스터프로그램의 모니터링화면에서 확인할 수 있습니다.



(상태타이머에 의해 명령의 실행상태가 실시간으로 모니터링 되고 있는 것으로 가정합니다.)

아래의 함수를 통해서 다음에 이어지는 명령의 ID값이 설정되게 됩니다.
WIMC_Set_CmdID(id);
… 이후 명령들은 모두 id값으로 명령ID가 설정됩니다.

모니터링화면의 Exe.PID (Packet ID)값은 현재 동작중인 보간동작명령에 대한 명령ID값으로 WIMC_Get_ExeCmdID() 함수로 읽을 수 있습니다.
Ready.PID로 표시되는 값은 아래 둘 중의 하나에 대한 명령ID값입니다.
  보간명령들 중 다음 실행을 위해 미리읽혀져 실행준비중인 명령ID
  보간명령을 제외한 명령들 중 실행되었거나 실행보류중인 명령ID
Ready.PID값은 WIMC_Get_ReadyCmdID() 함수로 읽을 수 있습니다.

※ 명령 ID 값은 함수의 인자형과 관계없이 8 비트만을 사용합니다. 그러므로 0~255 번까지 사용할 수 있습니다. 그러나 201 번부터는 내부적으로 특별한 신호의미로 사용되도록 예약되었으므로 임의적으로 사용할 수 있는 명령 ID 값은 최종 0~200 까지입니다.

➲ Parameters

id

설정할 명령 ID입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

6.1.2. WIMC_Get_CmdID

➲ Syntax

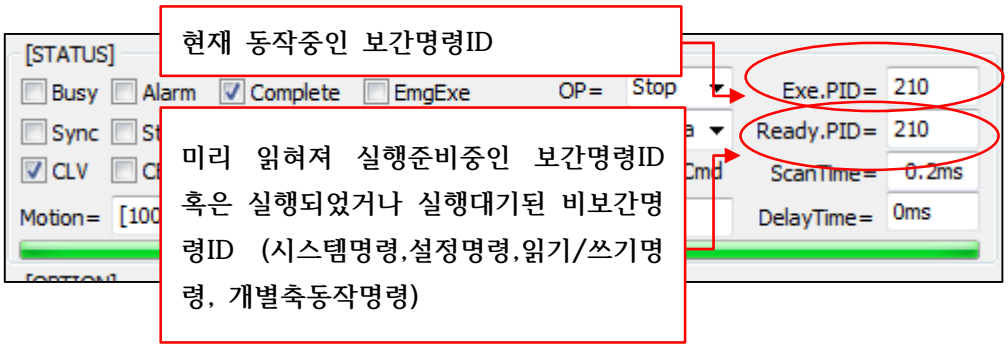
```
WIMC_INT32    WIMC_API                WIMC_Get_CmdID( void);
```

➲ Description

명령 ID를 읽습니다.

❶ 명령 ID 설정

모션엔진에 전달되는 모든 명령은 내부적으로 8비트의 명령ID값을 갖고 있습니다.
이 명령ID값은 아래의 모션마스터프로그램의 모니터링화면에서 확인할 수 있습니다.



(상태타이머에 의해 명령의 실행상태가 실시간으로 모니터링 되고 있는 것으로 가정합니다.)

아래의 함수를 통해서 다음에 이어지는 명령의 ID값이 설정되게 됩니다.
WIMC_Set_CmdID(id);
… 이후 명령들은 모두 id값으로 명령ID가 설정됩니다.

모니터링화면의 Exe.PID (Packet ID)값은 현재 동작중인 보간동작명령에 대한 명령ID값으로 WIMC_Get_ExeCmdID() 함수로 읽을 수 있습니다.
Ready.PID로 표시되는 값은 아래 둘 중의 하나에 대한 명령ID값입니다.
  보간명령들 중 다음 실행을 위해 미리읽혀져 실행준비중인 명령ID
  보간명령을 제외한 명령들 중 실행되었거나 실행보류중인 명령ID
Ready.PID값은 WIMC_Get_ReadyCmdID() 함수로 읽을 수 있습니다.

※ 명령 ID 값은 함수의 인자형과 관계없이 8 비트만을 사용합니다. 그러므로 0~255 번까지 사용할 수 있습니다. 그러나 201 번부터는 내부적으로 특별한 신호의미로 사용되도록 예약되었으므로 임의적으로 사용할 수 있는 명령 ID 값은 최종 0~200 까지입니다.

➲ Parameters

➲ Return Value

명령 ID를 반환합니다.

Return code/value	Description
WIMC_ERR	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

6.1.3. WIMC_Get_ExeCmdID

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_ExeCmdID( void);
```

➲ Description

모션엔진의 현재실행중인 모션동작명령ID값을 표시합니다. (0 ~255)

➲ Parameters

➲ Return Value

모션엔진의 현재실행중인 모션동작명령ID값을 반환합니다. (0 ~255)

Return code/value	Description
WIMC_ERR	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

6.1.4. WIMC_Get_ReadyCmdID

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ReadyCmdID( void);
```

➲ Description

모션엔진이 실행준비를 마치고 연속동작을 위해 대기하고 있는 명령의 ID값을 읽습니다.
그 외 축을 움직이는 모션동작이 아닌 명령들에 대한 실행ID값도 포함됩니다.

➲ Parameters

➲ Return Value

모션엔진이 실행준비를 마치고 연속동작을 위해 대기하고 있는 명령의 ID값을 반환합니다.

Return code/value	Description
WIMC_ERR	실패했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

7. Queue

7.1. Command Queue

7.1.1. WIMC_Enter_QueueCmd

➲ Syntax

```
void          WIMC_API                  WIMC_Enter_QueueCmd( void);
```

➲ Description

명령큐 형식의 명령으로 전달하도록 합니다.

DLL 라이브러리 함수를 이용한 명령전송방법은 아래와 같습니다.
WIMC_Enter_QueueCmd();
...명령나열 ;// 여기에 존재하는 명령은 명령큐형식의 명령으로 전달하게 됩니다.
WIMC_Leave_QueueCmd()

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

WIMC_Enter_ImCmd();
WIMC_Leave_ImCmd();
WIMC_Leave_QueueCmd();

7.1.2. WIMC_Leave_QueueCmd

➲ Syntax

```
void          WIMC_API                  WIMC_Leave_QueueCmd( void);
```

➲ Description

명령큐 실행 명령 방식을 해제합니다.
이전 실행 방식으로 돌아갑니다.

```
WIMC_Enter_QueueCmd();
...명령나열  ;// 여기에 존재하는 명령은 명령큐형식의 명령으로 전달하게 됩니다.
WIMC_Leave_QueueCmd()
```

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

7.1.3. WIMC_Enter_ImCmd

➲ Syntax

```
void          WIMC_API                  WIMC_Enter_ImCmd( void);
```

➲ Description

즉시 실행형식의 명령으로 전달하도록 합니다.

```
WIMC_Enter_ImCmd();
..명령나열 ;// 여기에 존재하는 명령은 즉시실행형식의 명령으로 전달하게 됩니다.
WIMC_Leave_ImCmd();
```

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Leave_ImCmd();
WIMC_Enter_QueueCmd();
WIMC_Leave_QueueCmd();
```

7.1.4. WIMC_Leave_ImCmd

➲ Syntax

```
void          WIMC_API                  WIMC_Leave_ImCmd( void);
```

➲ Description

즉시실행 명령 방식을 해제합니다.
이전 실행 방식으로 돌아갑니다.

```
WIMC_Enter_ImCmd();
..명령나열 ;// 여기에 존재하는 명령은 즉시실행형식의 명령으로 전달하게 됩니다.
WIMC_Leave_ImCmd();
```

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

7.1.5. WIMC_Reallocate_SoftQueue

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Reallocate_SoftQueue
(
              WIMC_INT32              queueSize
);
```

➲ Description

소프트웨어 명령큐의 메모리 용량을 조절합니다.

WIMC 엔진은 내부적으로 수KBYTE용량의 명령큐버퍼를 갖고 있습니다. 일반적으로 100~200개 내외의 명령을 저장할 수 있습니다. 그러나 DLL라이브러리는 엔진내부의 큐와 더불어 소프트웨어 명령큐기능을 통합하고 있어 PC의 메모리가 허용하는 만큼의 명령큐를 사용할 수 있습니다.

WIMC DLL라이브러리를 초기화 하게 되면 기본적으로 10Mbyte 용량의 명령큐를 초기할당하게 됩니다.
현재 할당된 명령큐의 메모리 용량은 WIMC_Get_SoftQueueSize() 함수로써 byte단위로 확인할 수 있습니다.

➲ Parameters

queueSize
명령큐 size입니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	명령큐 할당에 성공했습니다.
WIMC_FALSE	명령큐 할당에 실패했습니다.

➲ Example

➲ See Also

WIMC_Get_SoftQueueSize()

7.1.6. WIMC_Get_SoftQueueSize

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_SoftQueueSize( void);
```

➲ Description

현재 할당된 명령큐의 메모리 용량을 byte단위로 읽습니다.

라이브러리를 초기화 하게 되면 기본적으로 10Mbyte 용량의 명령큐를 초기할당하게 됩니다.

WIMC_Reallocate_SoftQueue() 함수를 사용하여 소프트웨어 명령큐의 메모리용량을 조절할 수 있습니다.

➲ Parameters

➲ Return Value

할당된 명령큐의 메모리 용량을 byte 단위로 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

WIMC_Reallocate_SoftQueue();

7.1.7. WIMC_Get_CmdQueueLeftCmd

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_CmdQueueLeftCmd( void);
```

➲ Description

엔진큐와 소프트웨어큐를 합친 명령큐에 남아 있는 명령수를 읽습니다.

➲ Parameters

➲ Return Value

엔진큐와 소프트웨어큐를 합친 명령큐에 남아 있는 명령수를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Get_EngineQueueLeftCmd();    // 엔진큐에 남아있는 명령수
WIMC_Get_SoftQueueLeftCmd();      // 소프트웨어큐에 남아있는 명령수
WIMC_Get_CmdQueueLeftCmd();       // 엔진큐와 소프트웨어큐를 합친 명령큐에 남아 있는 명령수
```

7.1.8. WIMC_Get_SoftQueueLeftCmd

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_SoftQueueLeftCmd( void);
```

➲ Description

소프트웨어큐에 남아있는 명령수를 읽습니다.

➲ Parameters

➲ Return Value

소프트웨어큐에 남아있는 명령수를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Get_EngineQueueLeftCmd();    // 엔진큐에 남아있는 명령수
WIMC_Get_SoftQueueLeftCmd();      // 소프트웨어큐에 남아있는 명령수
WIMC_Get_CmdQueueLeftCmd();       // 엔진큐와 소프트웨어큐를 합친 명령큐에 남아 있는 명령수
```

7.1.9. WIMC_Get_EngineQueueLeftCmd

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_EngineQueueLeftCmd( void);
```

➲ Description

엔진큐에 남아있는 명령수를 읽습니다.

➲ Parameters

➲ Return Value

엔진큐에 남아있는 명령수를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Get_EngineQueueLeftCmd();    // 엔진큐에 남아있는 명령수
WIMC_Get_SoftQueueLeftCmd();      // 소프트웨어큐에 남아있는 명령수
WIMC_Get_CmdQueueLeftCmd();       // 엔진큐와 소프트웨어큐를 합친 명령큐에 남아 있는 명령수
```

7.1.10. WIMC_Clr_Queue

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Clr_Queue( void);
```

➲ Description

명령큐 내용을 클리어합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

PC상의 소프트웨어 Queue도 함께 클리어합니다

➲ See Also

시스템 명령은 전체동작의 리셋, 실행, 정지, 회복, 재기동, 초기화 등의 엔진내부의 기초적인 명령들로 구성됩니다.
아래에 시스템명령군 포함된 명령들을 나열하며 ()안에 관련된 라이브러리 함수도 표기합니다.

(1) RESET 명령 (WIMC_Reset();)
(2) EMGSTOP 명령 (WIMC_Estop(TRUE); or WIMC_Estop(FALSE);)
(3) ENABLE EXE 명령 (WIMC_Enable_Exe();)
(4) ALARM RESET 명령 (WIMC_AlarmReset();)
(5) RESUME 명령 (WIMC_Resume();)
(6) CANCEL 명령 (WIMC_Cancel();)
(7) CLEAR QUEUE 명령 (WIMC_Clr_Queue();)

8. State Packet

8.1. Busy

8.1.1. WIMC_Enable_BusyStatePacket

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_BusyStatePacket
(
              WIMC_BOOL               bEnable
);
```

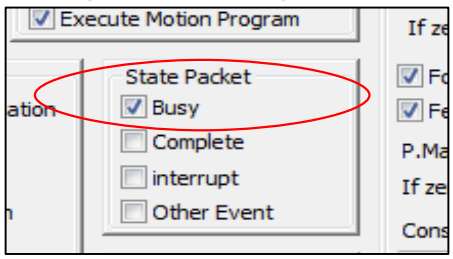
➲ Description

Busy 상태 패킷전송을 Enable/Disable 합니다.

Busy State Packet

BUSY상태패킷은 BUSY신호가 발생될 때 상위제어기에 보내는 상태패킷입니다. 상태타이머에 의해 Busy상태가 주기적으로 전달되지만, 상태타이머의 동작주파수를 25Hz이하로 낮게 설정하였다면 Busy상태패킷전송을 활성화하여 상위제어기가 빠르게 Busy상태를 확인할 수 있도록 도와줄 수 있습니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>

기능 OFF

➲ Parameters

bEnable

WIMC_TRUE : busy 상태패킷 전송기능 Enable.

WIMC_FALSE : busy 상태패킷 전송기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_BusyStatePacket();    // Busy상태패킷 전송기능 설정
WIMC_Get_BusyStatePacketCount();  // Busy상태패킷의 수신카운터(횟수) 읽기
WIMC_Set_BusyStatePacketCount();  // Busy상태패킷의 수신카운터(횟수) 설정
```

8.1.2. WIMC_Is_BusyStatePacket

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_BusyStatePacket( void);
```

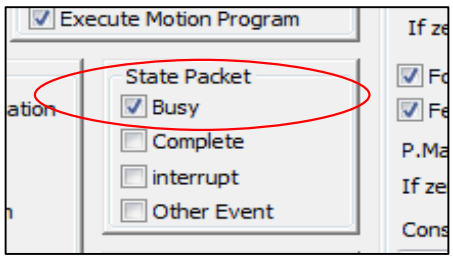
➲ Description

Busy 상태 패킷전송을 설정(Enable/Disable)을 읽습니다

Busy State Packet

BUSY상태패킷은 BUSY신호가 발생될 때 상위제어기에 보내는 상태패킷입니다. 상태타이머에 의해 Busy상태가 주기적으로 전달되지만, 상태타이머의 동작주파수를 25Hz이하로 낮게 설정하였다면 Busy상태패킷전송을 활성화하여 상위제어기가 빠르게 Busy상태를 확인할 수 있도록 도와줄 수 있습니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>

기능 OFF

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	busy 상태패킷 전송기능 Enable 설정 상태입니다.
WIMC_FALSE	busy 상태패킷 전송기능 Disable 설정 상태입니다.

➲ Example

➲ See Also

```
WIMC_Enable_BusyStatePacket();    // Busy상태패킷 전송기능 설정
WIMC_Get_BusyStatePacketCount();  // Busy상태패킷의 수신카운터(횟수) 읽기
WIMC_Set_BusyStatePacketCount();  // Busy상태패킷의 수신카운터(횟수) 설정
```

8.1.3. WIMC_Set_BusyStatePacketCount

➲ Syntax

```
void        WIMC_API                WIMC_Set_BusyStatePacketCount
(
            WIMC_UINT32             cnt
);
```

➲ Description

Busy 상태패킷의 수신카운터(횟수)를 설정합니다.

Busy State Packet

BUSY상태패킷은 BUSY신호가 발생될 때 상위제어기에 보내는 상태패킷입니다. 상태타이머에 의해 Busy상태가 주기적으로 전달되지만, 상태타이머의 동작주파수를 25Hz이하로 낮게 설정하였다면 Busy상태패킷전송을 활성화하여 상위제어기가 빠르게 Busy상태를 확인할 수 있도록 도와줄 수 있습니다.

<리셋시 설정상태>
기능 OFF

➲ Parameters

cnt

Busy 상태패킷의 수신 카운터값입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_BusyStatePacket();    // Busy상태패킷 전송기능 설정
WIMC_Get_BusyStatePacketCount();  // Busy상태패킷의 수신카운터(횟수) 읽기
WIMC_Set_BusyStatePacketCount();  // Busy상태패킷의 수신카운터(횟수) 설정
```

8.1.4. WIMC_Get_BusyStatePacketCount

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Get_BusyStatePacketCount( void);
```

➲ Description

Busy 상태 패킷의 수신 횟수를 읽습니다.

Busy 상태패킷의 수신카운터(횟수)를 설정합니다.

Busy State Packet

BUSY상태패킷은 BUSY신호가 발생될 때 상위제어기에 보내는 상태패킷입니다. 상태타이머에 의해 Busy상태가 주기적으로 전달되지만, 상태타이머의 동작주파수를 25Hz이하로 낮게 설정하였다면 Busy상태패킷전송을 활성화하여 상위제어기가 빠르게 Busy상태를 확인할 수 있도록 도와줄 수 있습니다.

<리셋시 설정상태>
기능 OFF

➲ Parameters

➲ Return Value

Busy 상태 패킷의 수신 횟수를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_BusyStatePacket();    // Busy상태패킷 전송기능 설정
WIMC_Get_BusyStatePacketCount();  // Busy상태패킷의 수신카운터(횟수) 읽기
WIMC_Set_BusyStatePacketCount();  // Busy상태패킷의 수신카운터(횟수) 설정
```

8.2. Complete

8.2.1. WIMC_Enable_CompleteStatePacket

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_CompleteStatePacket
(
              WIMC_BOOL               bEnable
);
```

➲ Description

Complete 상태패킷 전송기능을 설정합니다.

Complete State Packet

COMPLETE상태패킷은 모션동작명령이 완료되어 COMPLETE신호가 발생되었들 때 상위제어기로 보내는 상태패킷입니다. 라이브러리의 모션명령함수들은 모달리스함수와 같이 명령을 전송하고 바로 리턴하게 됩니다. 만약 모션명령이 축의 움직임을 발생하는 동작형 명령이다면 명령이 완료되기까지 특정시간이 요구될 것입니다. 보간동작이 완료되기를 기다려야 한다면 COMPLETE상태패킷을 활성시켜야만 합니다. 아래에 이전 보간명령이 완료되기를 기다리는 형태의 명령구성 예를 보여줍니다.

```
INT32 P1[8]={1000,1000};          // P1의 위치는 1000,1000 으로 설정합니다.
WIMC_Set_CmdID(100);              //  이후 명령의 ID는 100으로 설정합니다.
WIMC_Line (WIMC_XY, P1);          //  목표위치 P1으로 XY축의 직선보간명령을 전송합니다.
WIMC_WaitMotion(100, 1000);       // 모션명령ID가 100의 값을 명령이 완료되기를 기다립니다.
                                  // 최대 1000msec (1초)동안만 대기합니다.
                                  // 1초가 경과된 동안 명령완료패킷이 전송되지 못하면 FALSE값을
                                  // 리턴합니다. 정상적으로 대기완료되면 TRUE값을 리턴합니다.
```

모션완료상태패킷(Motion Complete State Packe) 은 축을 움직이는 동작명령에만 유효합니다. 속도를 설정하거나 옵션들을 설정하는 등의 명령에서는 아무런 반응이 없습니다.

※ **스핀들 회전명령에는 COMPLETE 응답이 없습니다. 스핀들 회전명령은 특정 위치각도까지 회전하는 것이 아니라 계속 회전하는 명령이므로 명령처리는 회전시작과 동시에 종료되어 일반적인 읽기/쓰기명령과 동일하기 때문입니다.**

초당 수백회 이상의 짧은 보간명령들을 연속으로 실행하면 모션완료상태패킷도 초당 수백회 이상 보낼 수 있습니다. 만약 통신속도가 적절하지 못할 경우 통신부하로 인해 모션완료패킷의 일부를 전송하지 못할 수도 있습니다.

※ WIMC 엔진의 통신은 내부버스와 같은 무결성통신에 기반해야만 합니다. 모션컨트롤러의 특징상 단 하나의 명령유실도 완전히 신뢰되지 못하는 결과를 발생하기 때문에 상위제어기간의 통신은 반드시 안정된 상태를 유지해야만 합니다.

<리셋시 설정상태>
기능 ON

➲ Parameters

bEnable

```
WIMC_TRUE : Enable.
WIMC_FALSE : Disable.
```

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_CompleteStatePacket();                    // Complete상태패킷 전송기능 설정
WIMC_Is_CompleteStatePacket();                  // Complete상태패킷 전송기능 설정여부 읽기
WIMC_Get_CompleteStatePacketCount();            // Complete상태패킷 수신카운터(횟수)읽기
WIMC_Set_CompleteStatePacketCount();            // Complete상태패킷 수신카운터(횟수)설정
```

8.2.2. WIMC_Is_CompleteStatePacket

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_CompleteStatePacket( void);
```

➲ Description

Complete 상태패킷 전송기능 설정을 확인합니다.

Complete State Packet

COMPLETE상태패킷은 모션동작명령이 완료되어 COMPLETE신호가 발생되었들 때 상위제어기로 보내는 상태패킷입니다. 라이브러리의 모션명령함수들은 모달리스함수와 같이 명령을 전송하고 바로 리턴하게 됩니다. 만약 모션명령이 축의 움직임을 발생하는 동작형 명령이다면 명령이 완료되기까지 특정시간이 요구될 것입니다. 보간동작이 완료되기를 기다려야 한다면 COMPLETE상태패킷을 활성시켜야만 합니다. 아래에 이전 보간명령이 완료되기를 기다리는 형태의 명령구성 예를 보여줍니다.

```
INT32 P1[8]={1000,1000};          // P1의 위치는 1000,1000 으로 설정합니다.
WIMC_Set_CmdID(100);              //  이후 명령의 ID는 100으로 설정합니다.
WIMC_Line (WIMC_XY, P1);          //  목표위치 P1으로 XY축의 직선보간명령을 전송합니다.
WIMC_WaitMotion(100, 1000);                 // 모션명령ID가 100의 값을 명령이 완료되기를 기다립니다.
                                  // 최대 1000msec (1초)동안만 대기합니다.
                                  // 1초가 경과된 동안 명령완료패킷이 전송되지 못하면 FALSE값을
                                  // 리턴합니다. 정상적으로 대기완료되면 TRUE값을 리턴합니다.
```

모션완료상태패킷(Motion Complete State Packe) 은 축을 움직이는 동작명령에만 유효합니다. 속도를 설정하거나 옵션들을 설정하는 등의 명령에서는 아무런 반응이 없습니다.

※ **스핀들 회전명령에는 COMPLETE 응답이 없습니다. 스핀들 회전명령은 특정 위치각도까지 회전하는 것이 아니라 계속 회전하는 명령이므로 명령처리는 회전시작과 동시에 종료되어 일반적인 읽기/쓰기명령과 동일하기 때문입니다.**

초당 수백회 이상의 짧은 보간명령들을 연속으로 실행하면 모션완료상태패킷도 초당 수백회 이상 보낼 수 있습니다. 만약 통신속도가 적절하지 못할 경우 통신부하로 인해 모션완료패킷의 일부를 전송하지 못할 수도 있습니다.

※ WIMC 엔진의 통신은 내부버스와 같은 무결성통신에 기반해야만 합니다. 모션컨트롤러의 특징상 단 하나의 명령유실도 완전히 신뢰되지 못하는 결과를 발생하기 때문에 상위제어기간의 통신은 반드시 안정된 상태를 유지해야만 합니다.

<리셋시 설정상태>
기능 ON

➲ Parameters

cnt

Complete 상태패킷 수신 카운터(횟수) 값을 설정합니다.

➲ Return Value

Complete 상태패킷 전송기능 설정여부 상태를 반환합니다

Return code/value	Description
WIMC_TRUE	Complete 상태패킷 전송기능 설정상태입니다.
WIMC_FALSE	Complete 상태패킷 전송기능 설정상태가 아닙니다.

➲ Example

➲ See Also

```
WIMC_Enable_CompleteStatePacket();                  // Complete상태패킷 전송기능 설정
WIMC_Is_CompleteStatePacket();                  // Complete상태패킷 전송기능 설정여부 읽기
WIMC_Get_CompleteStatePacketCount();            // Complete상태패킷 수신카운터(횟수)읽기
WIMC_Set_CompleteStatePacketCount();            // Complete상태패킷 수신카운터(횟수)설정
```

8.2.3. WIMC_Set_CompleteStatePacketCount

➲ Syntax

```
void        WIMC_API                WIMC_Set_CompleteStatePacketCount
(
            WIMC_UINT32             cnt
);
```

➲ Description

Complete 상태패킷 수신카운터(횟수)를 설정합니다.

Complete State Packet

COMPLETE상태패킷은 모션동작명령이 완료되어 COMPLETE신호가 발생되었들 때 상위제어기로 보내는 상태패킷입니다. 라이브러리의 모션명령함수들은 모달리스함수와 같이 명령을 전송하고 바로 리턴하게 됩니다. 만약 모션명령이 축의 움직임을 발생하는 동작형 명령이다면 명령이 완료되기까지 특정시간이 요구될 것입니다. 보간동작이 완료되기를 기다려야 한다면 COMPLETE상태패킷을 활성시켜야만 합니다. 아래에 이전 보간명령이 완료되기를 기다리는 형태의 명령구성 예를 보여줍니다.

```
INT32 P1[8]={1000,1000};          // P1의 위치는 1000,1000 으로 설정합니다.
WIMC_Set_CmdID(100);              //  이후 명령의 ID는 100으로 설정합니다.
WIMC_Line (WIMC_XY, P1);          //  목표위치 P1으로 XY축의 직선보간명령을 전송합니다.
WIMC_WaitMotion(100, 1000);       // 모션명령ID가 100의 값을 명령이 완료되기를 기다립니다.
                                  // 최대 1000msec (1초)동안만 대기합니다.
                                  // 1초가 경과된 동안 명령완료패킷이 전송되지 못하면 FALSE값을
                                  // 리턴합니다. 정상적으로 대기완료되면 TRUE값을 리턴합니다.
```

모션완료상태패킷(Motion Complete State Packe) 은 축을 움직이는 동작명령에만 유효합니다. 속도를 설정하거나 옵션들을 설정하는 등의 명령에서는 아무런 반응이 없습니다.

※ **스핀들 회전명령에는 COMPLETE 응답이 없습니다. 스핀들 회전명령은 특정 위치각도까지 회전하는 것이 아니라 계속 회전하는 명령이므로 명령처리는 회전시작과 동시에 종료되어 일반적인 읽기/쓰기명령과 동일하기 때문입니다.**

초당 수백회 이상의 짧은 보간명령들을 연속으로 실행하면 모션완료상태패킷도 초당 수백회 이상 보낼 수 있습니다. 만약 통신속도가 적절하지 못할 경우 통신부하로 인해 모션완료패킷의 일부를 전송하지 못할 수도 있습니다.

※ WIMC 엔진의 통신은 내부버스와 같은 무결성통신에 기반해야만 합니다. 모션컨트롤러의 특징상 단 하나의 명령유실도 완전히 신뢰되지 못하는 결과를 발생하기 때문에 상위제어기간의 통신은 반드시 안정된 상태를 유지해야만 합니다.

<리셋시 설정상태>
기능 ON

➲ Parameters

cnt

Complete 상태패킷 수신 카운터(횟수) 값을 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_CompleteStatePacket(); // Complete상태패킷 전송기능 설정
WIMC_Get_CompleteStatePacketCount(); // Complete상태패킷 수신카운터(횟수)읽기
WIMC_Set_CompleteStatePacketCount(); // Complete상태패킷 수신카운터(횟수)설정
```

8.2.4. WIMC_Get_CompleteStatePacketCount

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Get_CompleteStatePacketCount( void);
```

➲ Description

Complete 상태패킷 수신카운터(횟수)를 읽습니다.

❶ Complete State Packet

COMPLETE상태패킷은 모션동작명령이 완료되어 COMPLETE신호가 발생되었들 때 상위제어기로 보내는 상태패킷입니다. 라이브러리의 모션명령함수들은 모달리스함수와 같이 명령을 전송하고 바로 리턴하게 됩니다. 만약 모션명령이 축의 움직임을 발생하는 동작형 명령이다면 명령이 완료되기까지 특정시간이 요구될 것입니다. 보간동작이 완료되기를 기다려야 한다면 COMPLETE상태패킷을 활성시켜야만 합니다. 아래에 이전 보간명령이 완료되기를 기다리는 형태의 명령구성 예를 보여줍니다.

```
INT32 P1[8]={1000,1000};          // P1의 위치는 1000,1000 으로 설정합니다.
WIMC_Set_CmdID(100);              //  이후 명령의 ID는 100으로 설정합니다.
WIMC_Line (WIMC_XY, P1);          //  목표위치 P1으로 XY축의 직선보간명령을 전송합니다.
WIMC_WaitMotion(100, 1000);       // 모션명령ID가 100의 값을 명령이 완료되기를 기다립니다.
                                  // 최대 1000msec (1초)동안만 대기합니다.
                                  // 1초가 경과된 동안 명령완료패킷이 전송되지 못하면 FALSE값을
                                  // 리턴합니다. 정상적으로 대기완료되면 TRUE값을 리턴합니다.
```

모션완료상태패킷(Motion Complete State Packe) 은 축을 움직이는 동작명령에만 유효합니다. 속도를 설정하거나 옵션들을 설정하는 등의 명령에서는 아무런 반응이 없습니다.

※ **스핀들 회전명령에는 COMPLETE 응답이 없습니다. 스핀들 회전명령은 특정 위치각도까지 회전하는 것이 아니라 계속 회전하는 명령이므로 명령처리는 회전시작과 동시에 종료되어 일반적인 읽기/쓰기명령과 동일하기 때문입니다.**

초당 수백회 이상의 짧은 보간명령들을 연속으로 실행하면 모션완료상태패킷도 초당 수백회 이상 보낼 수 있습니다. 만약 통신속도가 적절하지 못할 경우 통신부하로 인해 모션완료패킷의 일부를 전송하지 못할 수도 있습니다.

※ WIMC 엔진의 통신은 내부버스와 같은 무결성통신에 기반해야만 합니다. 모션컨트롤러의 특징상 단 하나의 명령유실도 완전히 신뢰되지 못하는 결과를 발생하기 때문에 상위제어기간의 통신은 반드시 안정된 상태를 유지해야만 합니다.

<리셋시 설정상태>
기능 ON

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_CompleteStatePacket(); // Complete상태패킷 전송기능 설정
WIMC_Get_CompleteStatePacketCount(); // Complete상태패킷 수신카운터(횟수)읽기
WIMC_Set_CompleteStatePacketCount(); // Complete상태패킷 수신카운터(횟수)설정
```

8.3. Interrupt

8.3.1. WIMC_Enable_InterruptStatePacket

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_InterruptStatePacket
(
              WIMC_BOOL               bEnable
);
```

➲ Description

Interrupt 상태 패킷 전송 기능을 Enable/Disable 합니다.

인터럽트신호의 발생옵션은 아래의 두가지 상황중 하나를 선택할 수 있습니다.
① [R01]칩상태레지스터의 하위 1바이트 값 변화될 때
② [R01]칩상태레지스터의 하위 1워드 값이 변화될 때입니다.

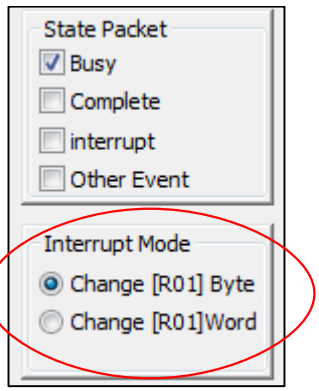
[R01] 칩상태레지스터의 하위 1바이트 정보는
- Busy신호 유무
- Complete신호 유무
- Sync신호 유무
- Step실행상태 유무
- Emg실행상태 유무
- Enable Exe 큐명령 실행가능상태 유무
- MotionOp 모션동작 실행중 유무
- Alarm 발생유무

입니다.

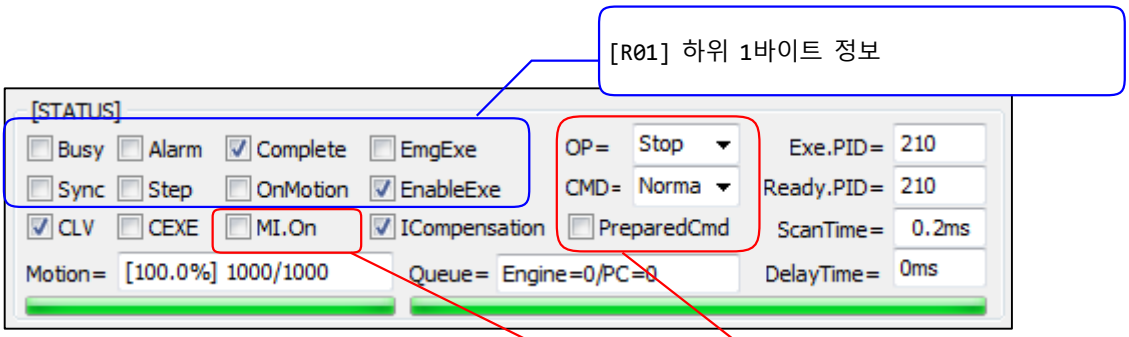
[R01] 칩상태레지스터의 하위 1워드에 추가된 정보는
- 현재 보간동작의 가감속모드
- 현재 보간동작의 가감속상태
- 현재 명령상태 : Normal명령상태/Sync명령상태/Wait명령상태/Delay명령상태
- 다중보간 실행상태 유무
- 실행준비를 마치고 대기하는 명령존재 유무

입니다.

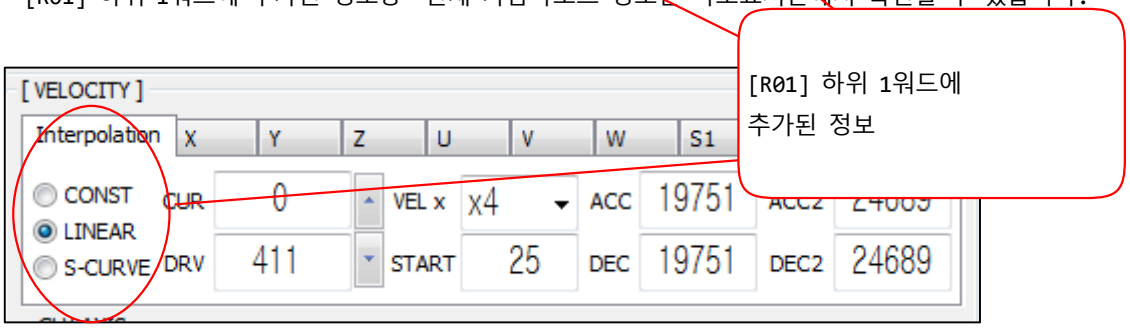
인터럽트신호발생옵션은 아래의 “모션마스터프로그램”의 엔진구성설정내에 포함되어 있습니다.



[R01]칩상태레지스터에 대한 상태값들은 아래의"모션마스터프로그램"의 모니터링화면의 엔진주요상태란[STATUS]에서 확인할 수 있습니다.



[R01] 하위 1워드에 추가된 정보중 현재 가감속모드 정보는 속도표시란에서 확인할 수 있습니다.



<리셋시 설정상태>

기능 OFF

➲ Parameters

bEnable

WIMC_TRUE : Enable.
WIMC_FALSE : Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

8.3.2. WIMC_Is_InterruptStatePacket

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_InterruptStatePacket( void);
```

➲ Description

Interrupt 상태 패킷 전송 기능 설정을 확인합니다.(Enable/Disable)

인터럽트신호의 발생옵션은 아래의 두가지 상황중 하나를 선택할 수 있습니다.
① [R01]칩상태레지스터의 하위 1바이트 값 변화될 때
② [R01]칩상태레지스터의 하위 1워드 값이 변화될 때입니다.

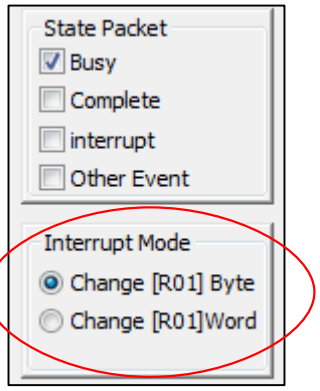
[R01] 칩상태레지스터의 하위 1바이트 정보는
- Busy신호 유무
- Complete신호 유무
- Sync신호 유무
- Step실행상태 유무
- Emg실행상태 유무
- Enable Exe 큐명령 실행가능상태 유무
- MotionOp 모션동작 실행중 유무
- Alarm 발생유무

입니다.

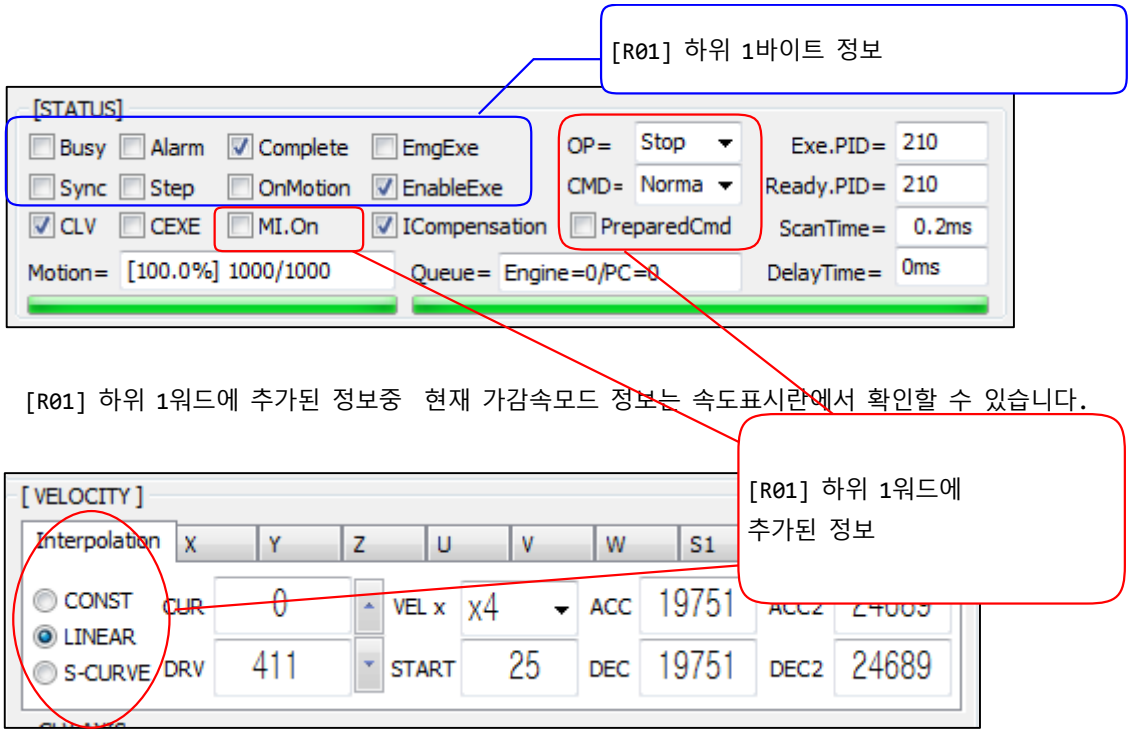
[R01] 칩상태레지스터의 하위 1워드에 추가된 정보는
- 현재 보간동작의 가감속모드
- 현재 보간동작의 가감속상태
- 현재 명령상태 : Normal명령상태/Sync명령상태/Wait명령상태/Delay명령상태
- 다중보간 실행상태 유무
- 실행준비를 마치고 대기하는 명령존재 유무

입니다.

인터럽트신호발생옵션은 아래의 “모션마스터프로그램”의 엔진구성설정내에 포함되어 있습니다.



[R01]칩상태레지스터에 대한 상태값들은 아래의“모션마스터프로그램”의 모니터링화면의 엔진주요상태란[STATUS]에서 확인할 수 있습니다.



[R01] 하위 1워드에 추가된 정보중 현재 가감속모드 정보는 속도표시란에서 확인할 수 있습니다.

<리셋시 설정상태>

기능 OFF

➲ Parameters

➲ Return Value

Interrupt 상태 패킷 전송 기능 설정을 반환합니다.

Return code/value	Description
WIMC_TRUE	Enable 상태입니다.
WIMC_FALSE	Disalbe 상태입니다.

➲ Example

➲ See Also

```
WIMC_Set_InterruptMode( WIMC_INTERRUPTMODE_CHANGE_BYTE_R01);
WIMC_Set_InterruptMode( WIMC_INTERRUPTMODE_CHANGE_WORD_R01);
WIMC_Enable_InterruptStatePacket(); // 인터럽트상태패킷 발생 기능설정
WIMC_Is_InterruptStatePacket(); // 인터럽트상태패킷 발생 기능설정 여부 읽기
WIMC_Set_InterruptStatePacketCount(); // 인터럽트상태패킷 수신카운터 설정
WIMC_Get_InterruptStatePacketCount(); // 인터럽트상태패킷 수신카운터 읽기
```

8.3.3. WIMC_Set_InterruptStatePacketCount

➲ Syntax

```
void        WIMC_API                WIMC_Set_InterruptStatePacketCount
(
            WIMC_UINT32             cnt
);
```

➲ Description

Interrupt상태패킷 수신카운터(횟수)를 설정합니다.

인터럽트신호의 발생옵션은 아래의 두가지 상황중 하나를 선택할 수 있습니다.

① [R01]칩상태레지스터의 하위 1바이트 값 변화될 때

② [R01]칩상태레지스터의 하위 1워드 값이 변화될 때입니다.

[R01] 칩상태레지스터의 하위 1바이트 정보는

- Busy신호 유무
- Complete신호 유무
- Sync신호 유무
- Step실행상태 유무
- Emg실행상태 유무
- Enable Exe 큐명령 실행가능상태 유무
- MotionOp 모션동작 실행중 유무
- Alarm 발생유무

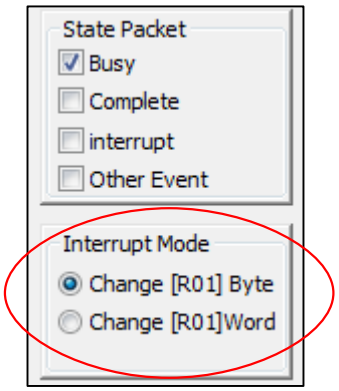
입니다.

[R01] 칩상태레지스터의 하위 1워드에 추가된 정보는

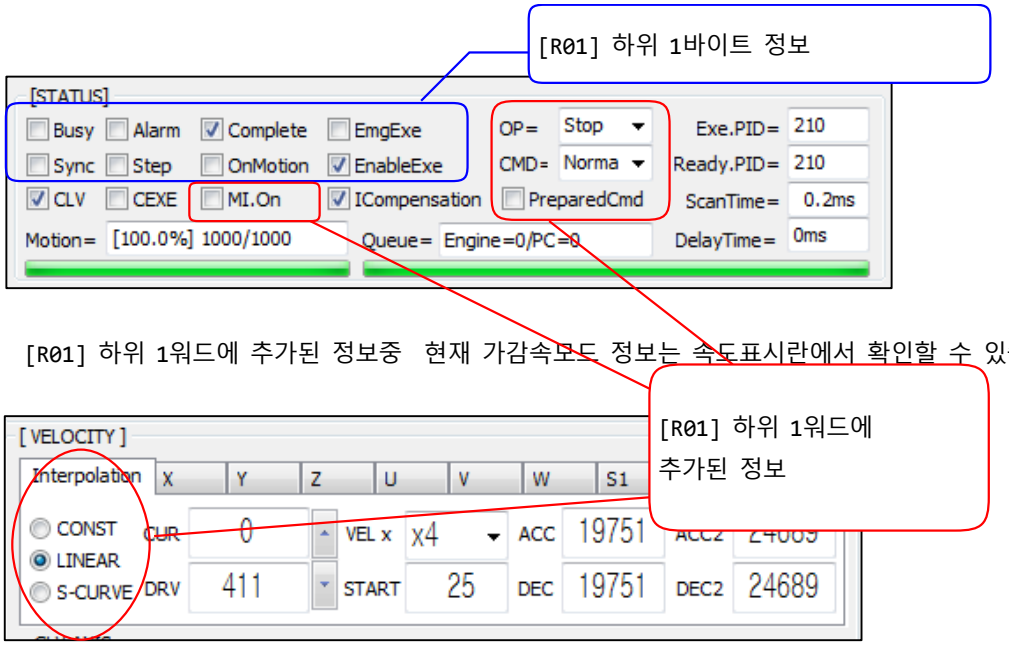
- 현재 보간동작의 가감속모드
- 현재 보간동작의 가감속상태
- 현재 명령상태 : Normal명령상태/Sync명령상태/Wait명령상태/Delay명령상태
- 다중보간 실행상태 유무
- 실행준비를 마치고 대기하는 명령존재 유무

입니다.

인터럽트신호발생옵션은 아래의 “모션마스터프로그램”의 엔진구성설정내에 포함되어 있습니다.



[R01]칩상태레지스터에 대한 상태값들은 아래의“모션마스터프로그램”의 모니터링화면의 엔진주요상태란[STATUS]에서 확인할 수 있습니다.



[R01] 하위 1워드에 추가된 정보중 현재 가감속모드 정보는 속도표시란에서 확인할 수 있습니다.

<리셋시 설정상태>
기능 OFF

➲ Parameters

cnt

Interrupt상태패킷 수신카운터(횟수) 입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_InterruptStatePacket(TRUE); // Interrupt상태패킷 전송기능 설정
WIMC_Enable_InterruptStatePacket(FALSE); // Interrupt상태패킷 전송기능 해제
WIMC_Get_InterruptStatePacketCount(); // Interrupt상태패킷 수신카운터(횟수)읽기
WIMC_Set_InterruptStatePacketCount(); // Interrupt상태패킷 수신카운터(횟수)설정
```

8.3.4. WIMC_Get_InterruptStatePacketCount

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Get_InterruptStatePacketCount( void);
```

➲ Description

Interrupt상태패킷 수신카운터(횟수)를 읽습니다.

인터럽트신호의 발생옵션은 아래의 두가지 상황중 하나를 선택할 수 있습니다.
① [R01]칩상태레지스터의 하위 1바이트 값 변화될 때
② [R01]칩상태레지스터의 하위 1워드 값이 변화될 때입니다.

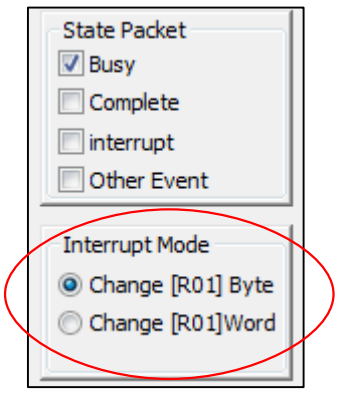
[R01] 칩상태레지스터의 하위 1바이트 정보는
- Busy신호 유무
- Complete신호 유무
- Sync신호 유무
- Step실행상태 유무
- Emg실행상태 유무
- Enable Exe 큐명령 실행가능상태 유무
- MotionOp 모션동작 실행중 유무
- Alarm 발생유무

입니다.

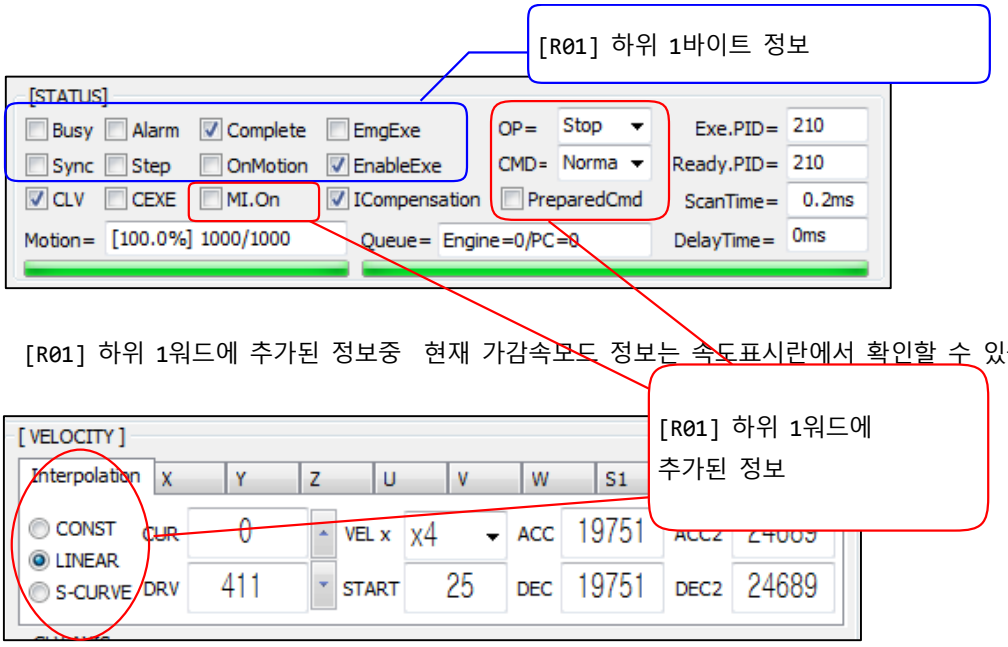
[R01] 칩상태레지스터의 하위 1워드에 추가된 정보는
- 현재 보간동작의 가감속모드
- 현재 보간동작의 가감속상태
- 현재 명령상태 : Normal명령상태/Sync명령상태/Wait명령상태/Delay명령상태
- 다중보간 실행상태 유무
- 실행준비를 마치고 대기하는 명령존재 유무

입니다.

인터럽트신호발생옵션은 아래의 “모션마스터프로그램”의 엔진구성설정내에 포함되어 있습니다.



[R01]칩상태레지스터에 대한 상태값들은 아래의“모션마스터프로그램”의 모니터링화면의 엔진주요상태란[STATUS]에서 확인할 수 있습니다.



[R01] 하위 1워드에 추가된 정보중 현재 가감속모드 정보는 속도표시란에서 확인할 수 있습니다.

<리셋시 설정상태>
기능 OFF

Parameters

Return Value

Interrupt상태패킷 수신카운터(횟수)를 반환합니다.

Return code/value	Description

Example

See Also

```
WIMC_Enable_InterruptStatePacket(TRUE); // Interrupt상태패킷 전송기능 설정
WIMC_Enable_InterruptStatePacket(FALSE); // Interrupt상태패킷 전송기능 해제
WIMC_Get_InterruptStatePacketCount(); // Interrupt상태패킷 수신카운터(횟수)읽기
WIMC_Set_InterruptStatePacketCount(); // Interrupt상태패킷 수신카운터(횟수)설정
```

8.4. Event

8.4.1. WIMC_Enable_EventStatePacket

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_EventStatePacket
(
              WIMC_BOOL               bEnable
);
```

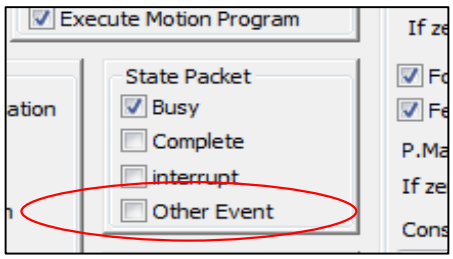
➲ Description

이벤트 상태패킷 기능을 Enable/Disable 합니다.

❶ 이벤트상태패킷

이벤트상태패킷은 엔진내부의 위치 비교기와 같은 특별한 제어장치에 의해 발생되는 통신패킷입니다.
위치비교기능을 사용할 경우 이 기능을 활성화 시킵니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
기능 ON

➲ Parameters

bEnable

WIMC_TRUE : 이벤트 상태패킷 기능 Enable.
WIMC_FALSE : 이벤트 상태패킷 기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_EventStatePacket(); // 이벤트상태패킷 발생 기능설정
WIMC_Is_EventStatePacket(); // 이벤트상태패킷 발생 기능설정 여부 읽기
WIMC_Set_EventStatePacketCount(); // 이벤트상태패킷 수신카운터 설정
WIMC_Get_EventStatePacketCount();// 이벤트상태패킷 수신카운터 읽기
```

8.4.2. WIMC_Is_EventStatePacket

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_EventStatePacket( void);
```

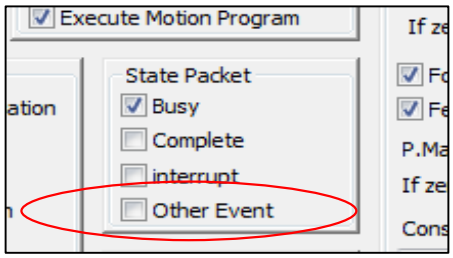
➲ Description

이벤트 상태패킷 기능의 설정 상태(Enable/Disable)을 확인합니다.

❶ 이벤트상태패킷

이벤트상태패킷은 엔진내부의 위치 비교기와 같은 특별한 제어장치에 의해 발생되는 통신패킷입니다.
위치비교기능을 사용할 경우 이 기능을 활성화 시킵니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
기능 ON

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	Enable 상태입니다.
WIMC_FALSE	Disable 상태입니다.

➲ Example

➲ See Also

WIMC_Enable_EventStatePacket(); // 이벤트상태패킷 발생 기능설정
WIMC_Is_EventStatePacket(); // 이벤트상태패킷 발생 기능설정 여부 읽기
WIMC_Set_EventStatePacketCount(); // 이벤트상태패킷 수신카운터 설정
WIMC_Get_EventStatePacketCount();// 이벤트상태패킷 수신카운터 읽기

8.4.3. WIMC_Set_EventStatePacketCount

`

➲ Syntax

```
void        WIMC_API                WIMC_Set_EventStatePacketCount
(
            WIMC_UINT32             cnt
);
```

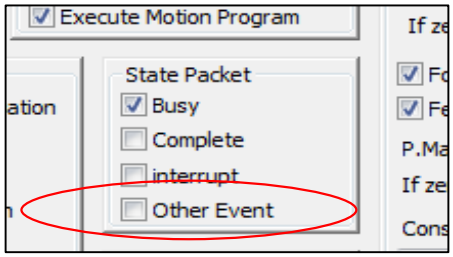
➲ Description

이벤트 상태패킷 수신카운터를 설정합니다.

ʊ 이벤트 상태패킷

이벤트상태패킷은 엔진내부의 위치 비교기와 같은 특별한 제어장치에 의해 발생되는 통신패킷입니다.
위치비교기능을 사용할 경우 이 기능을 활성화 시킵니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
기능 ON

➲ Parameters

cnt
설정할 이벤트 상태패킷 수신카운터 값입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

WIMC_Enable_EventStatePacket(); // 이벤트상태패킷 발생 기능설정
WIMC_Is_EventStatePacket(); // 이벤트상태패킷 발생 기능설정 여부 읽기

```
WIMC_Set_EventStatePacketCount(); // 이벤트상태패킷 수신카운터 설정
WIMC_Get_EventStatePacketCount();// 이벤트상태패킷 수신카운터 읽기
```

8.4.4. WIMC_Get_EventStatePacketCount

➲ Syntax

```
WIMC_UINT32   WIMC_API                  WIMC_Get_EventStatePacketCount( void);
```

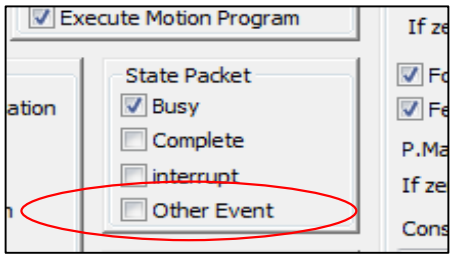
➲ Description

이벤트 상태패킷 수신카운터를 읽습니다.

❶ 이벤트상태패킷

이벤트상태패킷은 엔진내부의 위치 비교기와 같은 특별한 제어장치에 의해 발생되는 통신패킷입니다.
위치비교기능을 사용할 경우 이 기능을 활성화 시킵니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
기능 ON

➲ Parameters

➲ Return Value

이벤트 상태패킷 수신카운터 값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_EventStatePacket(); // 이벤트상태패킷 발생 기능설정
WIMC_Is_EventStatePacket(); // 이벤트상태패킷 발생 기능설정 여부 읽기
WIMC_Set_EventStatePacketCount(); // 이벤트상태패킷 수신카운터 설정
WIMC_Get_EventStatePacketCount();// 이벤트상태패킷 수신카운터 읽기
```

9. Settings

9.1. Settings

9.1.1. WIMC_MainLinkAlarmOn

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_MainLinkAlarmOn( WIMC_BOOL on);
```

➲ Description

모션보드와의 통신이 안될 경우(3초 이내) 모션보드 스스로가 알람 상태가 되도록 설정합니다.

➲ Parameters

on

WIMC_TRUE : 알람을 활성화 합니다.

WIMC_FALSE : 비활성화 합니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	설정 OK
WIMC_FALSE	설정 NG

➲ Example

```
WIMC_MainLinkAlarmOn(WIMC_TRUE);
```

➲ See Also

9.1.2. WIMC_Is_MainLinkAlarmOn

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_MainLinkAlarmOn( void);
```

➲ Description

모션보드와의 통신이 안될 경우(3초 이내) 모션보드 스스로가 알람 상태가 되도록 하는 설정을 확인합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	활성
WIMC_FALSE	비활성

➲ Example

➲ See Also

9.1.3. WIMC_Set_AxisPulseWidth

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Set_AxisPulseWidth( WIMC_UINT32 pulseWidth);
```

➲ Description

모션보드 축펄스 폭을 설정합니다.

➲ Parameters

pulseWidth

모션보드 축펄스 폭(uSec)을 설정합니다.

0 : 기본값 2.0 uSec
1 : 3.0 uSec
2 : 4.0 uSec
3 : 5.0 uSec
4 : 7.5 uSec
5 : 10.0 uSec
6 : MaxSpeed(모션보드 내부 설정)

➲ Return Value

Return code/value	Description
WIMC_TRUE	설정 OK
WIMC_FALSE	설정 NG

➲ Example

```
WIMC_Set_AxisPulseWidth(0);
```

➲ See Also

9.1.4. WIMC_Get_AxisPulseWidth

➲ Syntax

```
WIMC_UINT32   WIMC_API                  WIMC_Get_AxisPulseWidth(void);
```

➲ Description

모션보드 축펄스 폭을 설정을 확인합니다.

➲ Parameters

pulseWidth

모션보드 축펄스 폭(uSec)을 설정합니다.

0 : 기본값 2.0 uSec
1 : 3.0 uSec
2 : 4.0 uSec
3 : 5.0 uSec
4 : 7.5 uSec
5 : 10.0 uSec
6 : MaxSpeed(모션보드 내부 설정)

➲ Return Value

Return code/value	Description
0	2uSec
1	
2	
3	
4	
5	
6	

➲ Example

```
WIMC_Set_AxisPulseWidth(0);
```

➲ See Also

9.1.5. WIMC_Set_InportMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_InportMode
(
              WIMC_INT32              mode
);
```

➲ Description

입력포트의 내부 Pull-Up 저항 연결상태를 설정합니다.

➲ Parameters

mode

WIMC_INPORTMODE_NOPULLUP : 입력신호 풀업연결 안함

WIMC_INPORTMODE_PULLUP : 입력신호 풀업 연결

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

9.1.6. WIMC_Get_InportMode

➲ Syntax

```
WIMC_INT32          WIMC_API                    WIMC_Get_InportMode( void);
```

➲ Description

입력포트의 내부 Pull-Up 저항 연결상태를 확인합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_INPORTMODE_NOPULLUP	입력신호 풀업연결 안함
WIMC_INPORTMODE_PULLUP	입력신호 풀업 연결

➲ Example

➲ See Also

9.1.7. WIMC_Set_OutportMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_OutortMode
(
              WIMC_INT32              mode
);
```

➲ Description

출력포트 회로특성 모드를 설정합니다.

모드 종류로는 Open-drain 출력과 totempole 출력 및 내부 Pull-up 저항 연결유무를 선택할 수 있습니다.

➲ Parameters

mode

WIMC_OUTPORTMODE_OPENDRAIN : 출력신호 Open drain
WIMC_OUTPORTMODE_OPENDRAIN_PULLUP : 출력신호 Open drain 및 풀업
WIMC_OUTPORTMODE_TOTEMPOLE : 출력신호 Totempole
WIMC_OUTPORTMODE_TOTEMPOLE_PULLUP : 출력신호 Totempole 및 풀업

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

9.1.8. WIMC_Get_OutportMode

➲ Syntax

```
WIMC_INT32          WIMC_API                    WIMC_Get_OutportMode( void);
```

➲ Description

출력포트 회로특성 모드를 설정합니다.

모드 종류로는 Open-drain 출력과 totempole 출력 및 내부 Pull-up 저항 연결유무가 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_OUTPORTMODE_OPENDRAIN	출력신호 Open drain
WIMC_OUTPORTMODE_OPENDRAIN_PULLUP	출력신호 Open drain 및 풀업
WIMC_OUTPORTMODE_TOTEMPOLE	출력신호 Totempole
WIMC_OUTPORTMODE_TOTEMPOLE_PULLUP	출력신호 Totempole 및 풀업

➲ Example

➲ See Also

9.1.9. WIMC_Set_PosDispMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_PosDispMode
(
              WIMC_INT32              mode
);
```

➲ Description

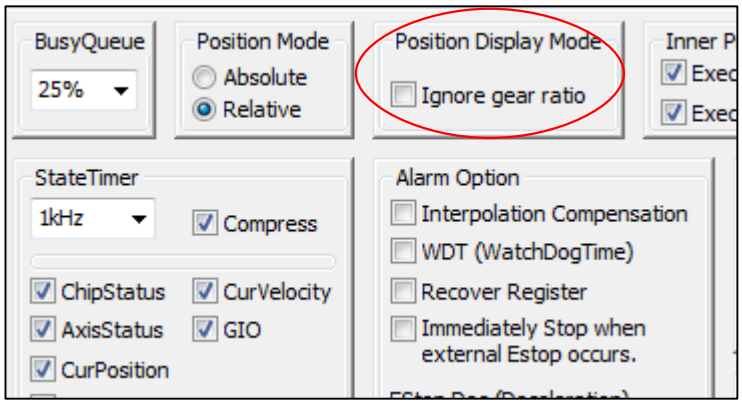
전자기어비가 적용된 위치표시모드를 설정합니다.

❶ 위치표시모드

일반적으로 전자기어비를 적용한 현재위치를 표시합니다. 전자기어비가 모든 축에 대해서 일정한 값으로 설정되었다면(즉 전자기어비가 적용되지 않은 것과 동일) 이 옵션의 설정여부에 대한 차이가 없습니다. 그러나 전자기어비가 각 축마다 서로 다르게 설정되었다면(전자기어비가 적용된 경우) 항상 전자기어비를 적용한 위치를 표시하는 것이 바람직합니다.

※ 전자기어비의 조성비율을 임의적으로 조절한 후 전자기어비적용을 무시한 위치를 표시하면 원호보간이 타원형상으로 보여집니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>

전자기어비가 적용된 현재위치표시모드(WIMC_POSDISPMODE_APPLY_GEAR)

➲ Parameters

mode

WIMC_POSDISPMODE_APPLY_GEAR : 전자기어비가 적용된 현재위치를 표시합니다.

WIMC_POSDISPMODE_IGNORE_GEAR : 전자기어비가 적용되지 않은 현재위치를 표시합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_PosDispMode( WIMC_POSDISPMODE_APPLY_GEAR);        // 전자기어비가 적용된 현재위치표시
WIMC_Set_PosDispMode( WIMC_POSDISPMODE_IGNORE_GEAR)        ;// 전자기어비가 적용되지 않은 현재위치표시
```

9.1.10. WIMC_Get_PosDispMode

➲ Syntax

```
WIMC_INT32          WIMC_API                    WIMC_Get_PosDispMode( void);
```

➲ Description

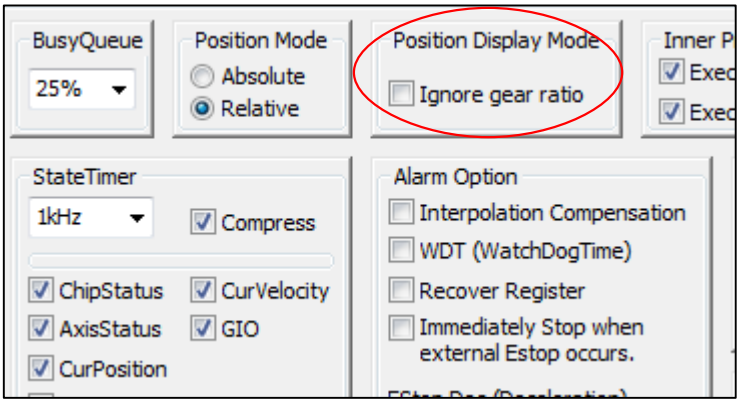
전자기어비가 적용된 위치표시모드 설정상태인지을 확인합니다.

ʊ 위치표시모드

일반적으로 전자기어비를 적용한 현재위치를 표시합니다. 전자기어비가 모든 축에 대해서 일정한 값으로 설정되었다면(즉 전자기어비가 적용되지 않은 것과 동일) 이 옵션의 설정여부에 대한 차이가 없습니다. 그러나 전자기어비가 각 축마다 서로 다르게 설정되었다면(전자기어비가 적용된 경우) 항상 전자기어비를 적용한 위치를 표시하는 것이 바람직합니다.

※ 전자기어비의 조성비율을 임의적으로 조절한 후 전자기어비적용을 무시한 위치를 표시하면 원호보간이 타원형상으로 보여집니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>

전자기어비가 적용된 현재위치표시모드(WIMC_POSDISPMODE_APPLY_GEAR)

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_POSDISPMODE_APPLY_GEAR	전자기어비가 적용된 현재위치표시 모드입니다.
WIMC_POSDISPMODE_IGNORE_GEAR	전자기어비가 적용되지 않은 현재위치 표시모드입니다.
WIMC_ERR	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

```
WIMC_Set_PosDispMode( WIMC_POSDISPMODE_APPLY_GEAR);        // 전자기어비가 적용된 현재위치표시
WIMC_Set_PosDispMode( WIMC_POSDISPMODE_IGNORE_GEAR);       // 전자기어비가 적용되지 않은 현재위치표시
```

9.1.11. WIMC_Set_PosMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_PosMode
(
              WIMC_INT32              mode
);
```

➲ Description

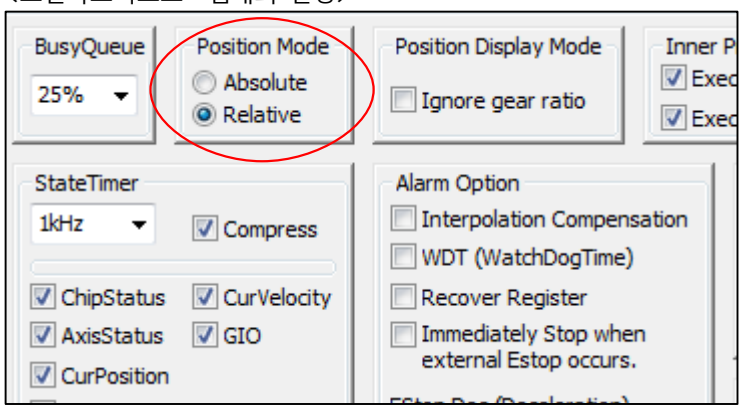
위치제어의 목적지 위치에 대한 정보를 표현하는 방식을 설정합니다. 상대위치를 기본적으로 사용하며 절대위치로 절환시 연속실행상태가 아닐 때만 안전합니다.

❶ 위치모드

WIMC 엔진은 부호있는 정수 32비트를 위치값으로 유지하며 모든 동작명령은 정수값으로 위치를 지령해야 합니다.
위치를 지령하는 방식에는 절대위치지령과 상대위치지령 2가지가 있습니다.
WIMC 엔진모션엔진은 기본적으로 상대위치지령을 사용합니다.
상대위치지령이란 입력된 위치값을 이동해야하는 변위값(상대적이동량)으로 인식하는 것입니다.
모션엔진코어는 기본적으로 상대위치지령에 의해 동작됩니다. 절대위치지령으로 모션동작을 실행시키더라도 명령을 내린 시점의 현재 위치값 기준으로 목표위치값까지의 상대위치를 계산하여 이동변위값으로 바꾸게 됩니다. 그러므로 별다른 동작이 필요로 하지 않는다면 위치모드는 항상 상대위치 지령방식으로 설정합니다.
보간동작명령은 상대위치지령과 절대위치지령방식을 구분하지만, 독립이송 등의 비보간동작명령들은 모든 목표위치설정값을 상대위치지령으로 취급합니다. WIMC라이브러리의 응용함수들도 대부분 상대위치지령에 기반하여 동작됩니다.

만약 절대위치 지령방식으로 사용한다면 아래의 경우에 유의하여 사용하셔야 합니다.
- 절대위치지령방식을 연속실행동작에서는 사용할 경우 이전동작의 실행을 완료하기 전에 현재위치에 대한 목표위치간의 상대위치가 계산되기 때문에 목표위치에 대한 상대적 변위량 계산에 오차가 발생합니다. 그러므로 절대위치지령방식을 사용할 경우 비연속실행동작에서 사용할 것을 권합니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
상대위치지령모드

➲ Parameters

mode

WIMC_POSMODE_REL : 상대위치지령 모드입니다.
WIMC_POSMODE_ABS : 절대위치지령 모드입니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

9.1.12. WIMC_Get_PosMode

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_PosMode( void);
```

➲ Description

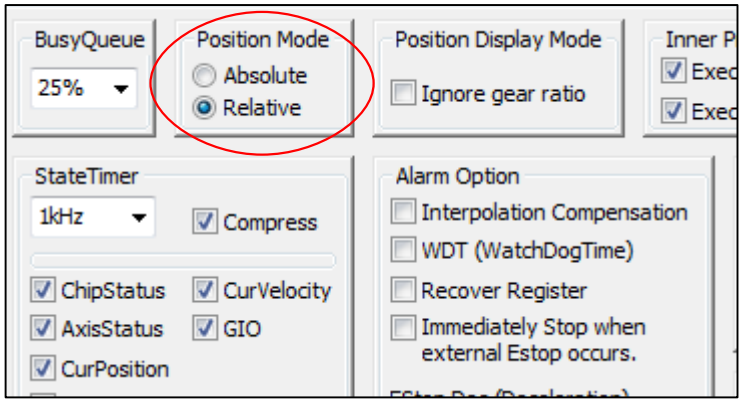
위치모드 설정상태를 확인합니다.

❶ 위치모드

WIMC 엔진은 부호있는 정수 32비트를 위치값으로 유지하며 모든 동작명령은 정수값으로 위치를 지령해야 합니다.
위치를 지령하는 방식에는 절대위치지령과 상대위치지령 2가지가 있습니다.
WIMC 엔진모션엔진은 기본적으로 상대위치지령을 사용합니다.
상대위치지령이란 입력된 위치값을 이동해야하는 변위값(상대적이동량)으로 인식하는 것입니다.
모션엔진코어는 기본적으로 상대위치지령에 의해 동작됩니다. 절대위치지령으로 모션동작을 실행시키더라도 명령을 내린 시점의 현재 위치값 기준으로 목표위치값까지의 상대위치를 계산하여 이동변위값으로 바꾸게 됩니다. 그러므로 별다른 동작이 필요로 하지 않는다면 위치모드는 항상 상대위치 지령방식으로 설정합니다.
보간동작명령은 상대위치지령과 절대위치지령방식을 구분하지만, 독립이송 등의 비보간동작명령들은 모든 목표위치설정값을 상대위치지령으로 취급합니다. WIMC라이브러리의 응용함수들도 대부분 상대위치지령에 기반하여 동작됩니다.

만약 절대위치 지령방식으로 사용한다면 아래의 경우에 유의하여 사용하셔야 합니다.
- 절대위치지령방식을 연속실행동작에서는 사용할 경우 이전동작의 실행을 완료하기 전에 현재위치에 대한 목표위치간의 상대위치가 계산되기 때문에 목표위치에 대한 상대적 변위량 계산에 오차가 발생합니다. 그러므로 절대위치지령방식을 사용할 경우 비연속실행동작에서 사용할 것을 권합니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
상대위치지령모드

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_POSMODE_REL	상대위치지령 모드입니다.
WIMC_POSMODE_ABS	절대위치지령 모드입니다.

➲ Example

➲ See Also

9.1.13. WIMC_Set_BusySize

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_BusySize
(
              WIMC_INT32              percent
);
```

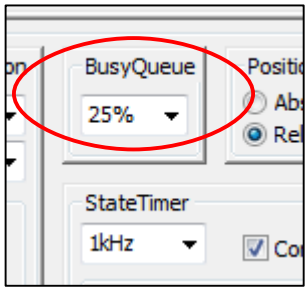
➲ Description

명령큐의 명령들이 차지하는 비중(%)에 따라 Busy 신호를 발생시킵니다.

BUSY 신호

BUSY신호는 WIMC 엔진의 내부 명령큐의 총 (바이트)크기 중에서 수신되었지만 처리되지 않고 남아있는 명령큐의 명령들이 차지하는 비중이 지정된 %보다 클 때 발생됩니다. 이 지정은 25%, 50%, 75%, 100% 중 4가지를 지정할 수 있습니다. 일반적으로 25%~50%를 설정합니다.
WIMC 라이브러리의 명령전송은 BUSY신호에 의해 자동 조절됩니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
25%

➲ Parameters

percent

25%, 50%, 75%, 100% 중 4가지를 지정할 수 있습니다. 일반적으로 25%~50%를 설정합니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

```
WIMC_Set_BusySize( busySize );    // busySize는 0, 25, 75, 100 중 하나를 선택합니다.
WIMC_Get_BusySize();
WIMC_Is_Busy();                   // 현재 Busy상태를 확인
```

9.1.14. WIMC_Get_BusySize

Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_BusySize( void);
```

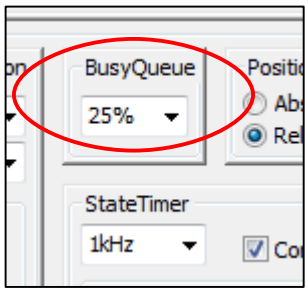
Description

명령큐의 명령들이 차지하는 비중(%)에 따라 Busy 신호를 발생시킵니다. 설정된 비중(%)을 확인합니다.

BUSY 신호

BUSY신호는 WIMC 엔진의 내부 명령큐의 총 (바이트)크기 중에서 수신되었지만 처리되지 않고 남아있는 명령큐의 명령들이 차지하는 비중이 지정된 %보다 클 때 발생됩니다. 이 지정은 25%, 50%, 75%, 100% 중 4가지를 지정할 수 있습니다. 일반적으로 25%~50%를 설정합니다.
WIMC 라이브러리의 명령전송은 BUSY신호에 의해 자동 조절됩니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
25%

Parameters

Return Value

0, 25%, 50%, 75%, 100% 를 반환합니다.

Return code/value	Description

Example

See Also

```
WIMC_Set_BusySize( busySize );    // busySize는 0, 25, 75, 100 중 하나를 선택합니다.
WIMC_Get_BusySize();
WIMC_Is_Busy();                   // 현재 Busy상태를 확인
```

9.1.15. WIMC_Set_InterruptMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_InterruptMode
(
              WIMC_INT32              mode
);
```

➲ Description

인터럽트 발생모드를 설정합니다.

인터럽트신호의 발생옵션은 아래의 두가지 상황중 하나를 선택할 수 있습니다.

① [R01]칩상태레지스터의 하위 1바이트 값 변화될 때

② [R01]칩상태레지스터의 하위 1워드 값이 변화될 때입니다.

[R01] 칩상태레지스터의 하위 1바이트 정보는

- Busy신호 유무
- Complete신호 유무
- Sync신호 유무
- Step실행상태 유무
- Emg실행상태 유무
- Enable Exe 큐명령 실행가능상태 유무
- MotionOp 모션동작 실행중 유무
- Alarm 발생유무

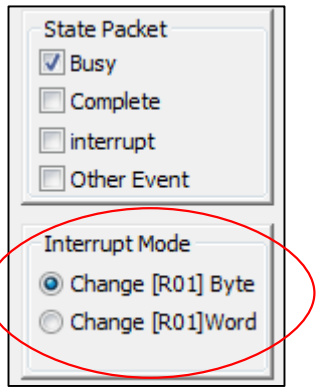
입니다.

[R01] 칩상태레지스터의 하위 1워드에 추가된 정보는

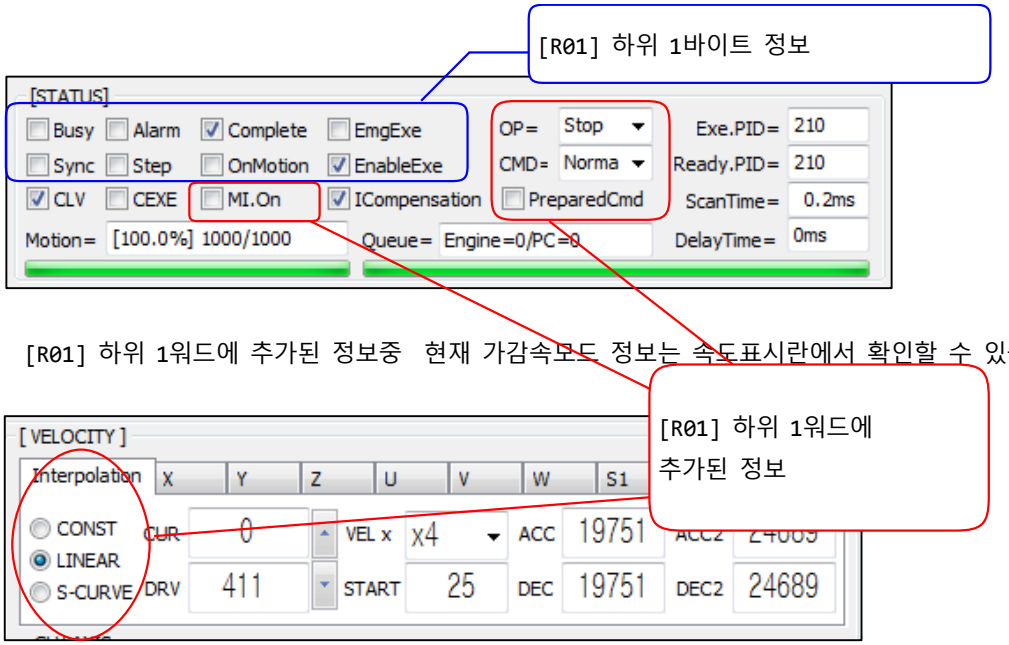
- 현재 보간동작의 가감속모드
- 현재 보간동작의 가감속상태
- 현재 명령상태 : Normal명령상태/Sync명령상태/Wait명령상태/Delay명령상태
- 다중보간 실행상태 유무
- 실행준비를 마치고 대기하는 명령존재 유무

입니다.

인터럽트신호발생옵션은 아래의 “모션마스터프로그램”의 엔진구성설정내에 포함되어 있습니다.



[R01]칩상태레지스터에 대한 상태값들은 아래의“모션마스터프로그램”의 모니터링화면의 엔진주요상태란[STATUS]에서 확인할 수 있습니다.



[R01] 하위 1워드에 추가된 정보중 현재 가감속모드 정보는 속도표시란에서 확인할 수 있습니다.

➲ Parameters

mode

WIMC_INTERRUPTMODE_CHANGE_BYTE_R01 : [R01]칩상태레지스터의 하위 1바이트 값이 변화될 때로 설정합니다.

WIMC_INTERRUPTMODE_CHANGE_WORD_R01 : [R01]칩상태레지스터의 하위 1워드 값이 변화될 때로 설정합니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

```
WIMC_Set_InterruptMode( WIMC_INTERRUPTMODE_CHANGE_BYTE_R01);
WIMC_Set_InterruptMode( WIMC_INTERRUPTMODE_CHANGE_WORD_R01);
WIMC_Enable_InterruptStatePacket(); // 인터럽트상태패킷 발생 기능설정
WIMC_Is_InterruptStatePacket(); // 인터럽트상태패킷 발생 기능설정 여부 읽기
WIMC_Set_InterruptStatePacketCount(); // 인터럽트상태패킷 수신카운터 설정
WIMC_Get_InterruptStatePacketCount(); // 인터럽트상태패킷 수신카운터 읽기
```

9.1.16. WIMC_Get_InterruptMode

➲ Syntax

```
WIMC_INT32          WIMC_API                    WIMC_Get_InterruptMode( void);
```

➲ Description

설정된 인터럽트 발생모드를 확인합니다.

인터럽트신호의 발생옵션은 아래의 두가지 상황중 하나를 선택할 수 있습니다.
① [R01]칩상태레지스터의 하위 1바이트 값 변화될 때
② [R01]칩상태레지스터의 하위 1워드 값이 변화될 때입니다.

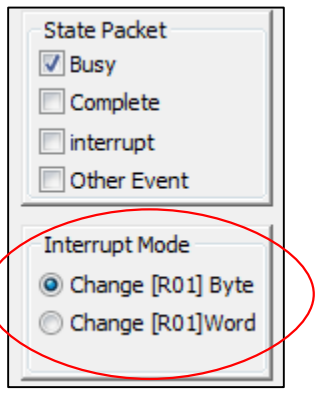
[R01] 칩상태레지스터의 하위 1바이트 정보는
- Busy신호 유무
- Complete신호 유무
- Sync신호 유무
- Step실행상태 유무
- Emg실행상태 유무
- Enable Exe 큐명령 실행가능상태 유무
- MotionOp 모션동작 실행중 유무
- Alarm 발생유무

입니다.

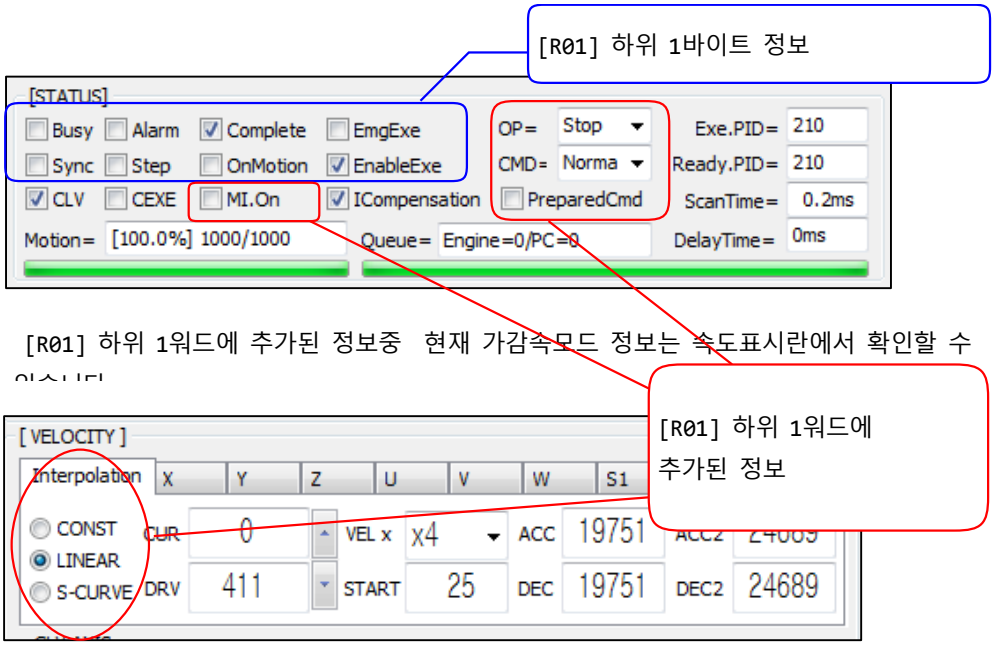
[R01] 칩상태레지스터의 하위 1워드에 추가된 정보는
- 현재 보간동작의 가감속모드
- 현재 보간동작의 가감속상태
- 현재 명령상태 : Normal명령상태/Sync명령상태/Wait명령상태/Delay명령상태
- 다중보간 실행상태 유무
- 실행준비를 마치고 대기하는 명령존재 유무

입니다.

인터럽트신호발생옵션은 아래의 “모션마스터프로그램”의 엔진구성설정내에 포함되어 있습니다.



[R01]칩상태레지스터에 대한 상태값들은 아래의“모션마스터프로그램”의 모니터링화면의 엔진주요상태란[STATUS]에서 확인할 수 있습니다.



[R01] 하위 1워드에 추가된 정보중 현재 가감속모드 정보는 속도표시란에서 확인할 수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_INTERRUPTMODE_CHANGE_BYTE_R01	[R01]칩상태레지스터의 하위 1바이트 값이 변화될 때로 설정되어 있습니다.
WIMC_INTERRUPTMODE_CHANGE_WORD_R01	[R01]칩상태레지스터의 하위 1워드 값이 변화될 때로 설정되어 있습니다.

➲ Example

➲ See Also

```
WIMC_Set_InterruptMode( WIMC_INTERRUPTMODE_CHANGE_BYTE_R01);
WIMC_Set_InterruptMode( WIMC_INTERRUPTMODE_CHANGE_WORD_R01);
WIMC_Enable_InterruptStatePacket(); // 인터럽트상태패킷 발생 기능설정
WIMC_Is_InterruptStatePacket(); // 인터럽트상태패킷 발생 기능설정 여부 읽기
WIMC_Set_InterruptStatePacketCount(); // 인터럽트상태패킷 수신카운터 설정
WIMC_Get_InterruptStatePacketCount(); // 인터럽트상태패킷 수신카운터 읽기
```

9.1.17. WIMC_Set_ProcCmdNo

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ProcCmdNo
(
              WIMC_INT32              numOfCmd
);
```

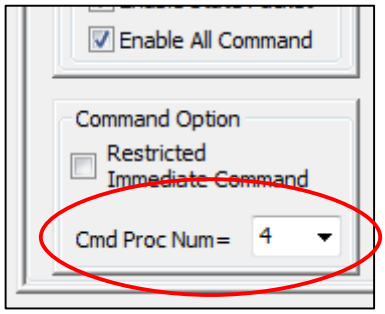
➲ Description

명령처리수를 지정합니다.

❶ 명령처리수 지정

1회 스캔타임동안 한번에 처리할 모션명령 수를 지정합니다. 1~16 중 하나의 수를 지정합니다.
너무 큰수를 지정하면 스캔타임을 길게 만들어 오히려 전체적인 모션명령의 실행속도를 낮출 수도 있습니다.
일반적으로 1~4 이내의 값을 설정합니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
명령처리수 = 1 로 설정

➲ Parameters

numOfCmd

1~16 중 하나의 수를 지정합니다.
1회 스캔타임동안 한번에 처리할 모션 명령 수를 지정합니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

```
WIMC_Set_ProcCmdNo();
WIMC_Get_ProcCmdNo();
```

9.1.18. WIMC_Get_ProcCmdNo

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ProcCmdNo( void);
```

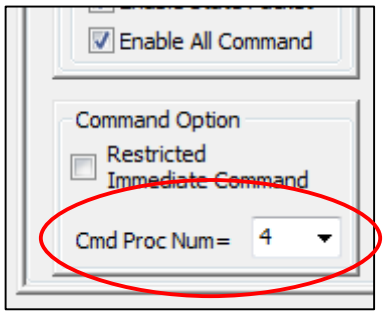
➲ Description

지정된 명령처리수를 확인합니다.

❶ 명령처리수 지정

1회 스캔타임동안 한번에 처리할 모션명령 수를 지정합니다. 1~16 중 하나의 수를 지정합니다.
너무 큰수를 지정하면 스캔타임을 길게 만들어 오히려 전체적인 모션명령의 실행속도를 낮출 수도 있습니다.
일반적으로 1~4 이내의 값을 설정합니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
명령처리수 = 1 로 설정

➲ Parameters

➲ Return Value

1~16 중 하나의 수를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

WIMC_Set_ProcCmdNo();
WIMC_Get_ProcCmdNo();

9.1.19. WIMC_Enable_RestrictedImCmd

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Enable_RestrictedImCmd
(
              WIMC_BOOL                 bEnable
);
```

➲ Description

즉시 실행명령의 엄격한 제한기능을 Enable/Disable 합니다.

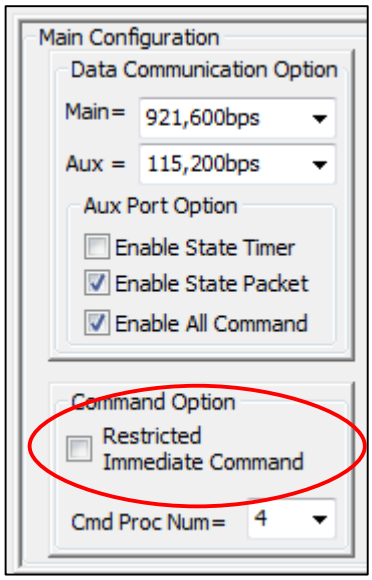
❶ 즉시실행명령의 엄격한 제한기능

이 기능이 설정되면 보간명령 및 IMOVE(독립이송명령), CMOVE(연속회전명령), 스핀들명령(SPINDLE) 등 축을 움직이는 명령등의 다수명령들을 즉시실행명령으로 실행할 수 없게 됩니다.

즉시실행명령의 엄격한 제한기능에 의해 실행 불가능한 명령들을 나열하면 아래와 같습니다.

- 원점복귀명령 (HOME, IHOME)
- 직선보간명령 (LINE)
- 원호보간명령 (CWARC, CCWARC)
- 비트패턴보간명령 (BITPATTERN)
- 독립이송명령 (IMOVE)
- 연속이송명령 (CMOVE)
- 스핀들명령 (SPINDLE)
- 조건이송명령 (XMOVE)

<모션마스터프로그램내의 설정>



<리셋시 설정상태>

기능 OFF

➲ Parameters

bEnable

WIMC_TRUE : Enable.
WIMC_FALSE : Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

WIMC_Enable_RestrictedImCmd(); // 엄격한 즉시실행명령모드 설정 및 해제합니다.
WIMC_Is_RestrictedImCmd(); // 현재설정상태를 읽습니다.

9.1.20. WIMC_Is_RestrictedImCmd

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_RestrictedImCmd( void);
```

➲ Description

설정된 즉시 실행명령의 엄격한 제한기능을 확인 합니다.

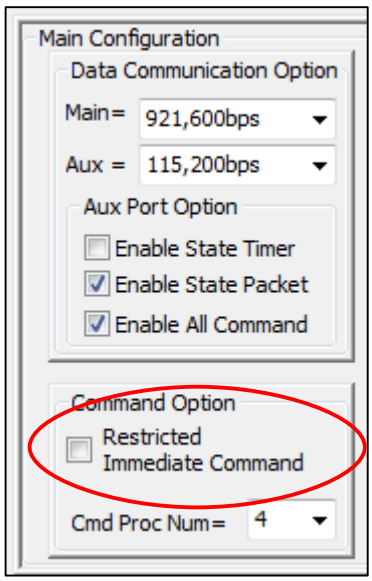
❶ 즉시실행명령의 엄격한 제한기능

이 기능이 설정되면 보간명령 및 IMOVE(독립이송명령), CMOVE(연속회전명령), 스핀들명령(SPINDLE) 등 축을 움직이는 명령등의 다수명령들을 즉시실행명령으로 실행할 수 없게 됩니다.

즉시실행명령의 엄격한 제한기능에 의해 실행 불가능한 명령들을 나열하면 아래와 같습니다.

- 원점복귀명령 (HOME, IHOME)
- 직선보간명령 (LINE)
- 원호보간명령 (CWARC, CCWARC)
- 비트패턴보간명령 (BITPATTERN)
- 독립이송명령 (IMOVE)
- 연속이송명령 (CMOVE)
- 스핀들명령 (SPINDLE)
- 조건이송명령 (XMOVE)

<모션마스터프로그램내의 설정>



<리셋시 설정상태>

기능 OFF

➲ Parameters

bEnable

WIMC_TRUE : Enable.
WIMC_FALSE : Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_RestrictedImCmd(); // 엄격한 즉시실행명령모드 설정 및 해제합니다.
WIMC_Is_RestrictedImCmd(); // 현재설정상태를 읽습니다.
```

9.1.21. WIMC_Enable_IEndComp

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_IEndComp
(
              WIMC_BOOL               bEnable
);
```

➲ Description

보간 보상기능을 Enable/Disable 합니다.

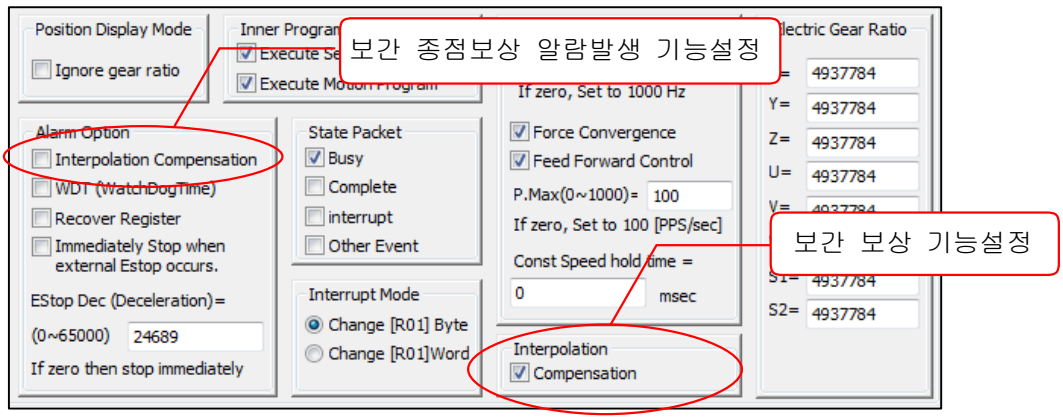
❶ 보간 보상 (Interpolation Compensation) 동작

직선보간명령은 내부연산의 오차없이 항상 목표위치로 도달하게 됩니다.
그러나 원호보간명령의 경우 지령된 목표위치값과 보간동작의 결과 목표위치값이 불일치 할 수도 있습니다. WIMC 엔진은 정수체계로써 위치를 입력받기 때문 종점위치에 대한 오차가 1펄스 이내에서 발생될 수 있습니다. 또한 원호보간의 중심위치와 종점위치가 바르게 입력되지 않았다면 원호보간의 결과는 1펄스를 초과하는 종점불일치가 발생될 수 있습니다.

이럴 경우 WIMC 엔진은 원호보간의 동작처리 후에 강제로 지령된 종점위치로 이동시키는 옵션이 보간보상동작옵션입니다 (보간종점보상).
WIMC 엔진은 기본적으로 상대위치지령에 의해 동작되므로 1펄스등의 오차가 계속 누적되게 되면 최종 모션동작의 결과가 허용되지 않을 만큼의 큰 오차를 포함할 수 있게 됩니다. 그러므로 하나의 명령에 의한 오차량은 비교적 작기 때문에 하나의 명령이 종료된 후 최종 지령위치와 비교하여 동일하지 않을 때에 강제로 지령된 종점위치로 수렴시킴으로써 상대위치지령으로 인한 오차누적을 없앨 수 있게 됩니다.

WIMC 엔진은 종점보상이 2펄스 이상 발생하게 되면 알람을 발생시킬 수 있어 오차를 허용하지 않을 경우 동작을 정지시켜 해당 보간명령을 디버깅할 수 있게 도와줍니다.

<모션마스터프로그램내의 설정>



<리셋시 설정>

기능 ON

➲ Parameters

bEnable

WIMC_TRUE : Enable.
WIMC_FALSE : Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_IEndComp (TRUE);                                  // 보간보상기능 설정
WIMC_Enable_IEndComp (FALSE);                                 // 보간보상기능 해제
WIMC_Set_AlarmOption( WIMC_ALARMOPTION_IENDCOMP); // 보간종점보상 알람발생 기능 설정
WIMC_Clr_AlarmOption( WIMC_ALARMOPTION_IENDCOMP); // 보간종점보상 알람발생 기능 해제
```

9.1.22. WIMC_Is_IEndComp

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_IEndComp( void);
```

➲ Description

모션엔진의 보간보상동작기능 설정을 확인합니다.

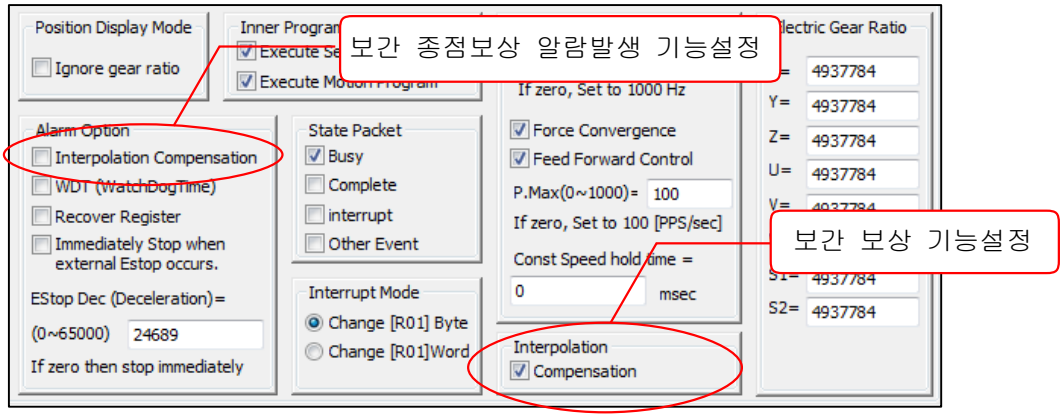
❶ 보간 보상 (Interpolation Compensation) 동작

직선보간명령은 내부연산의 오차없이 항상 목표위치로 도달하게 됩니다.
그러나 원호보간명령의 경우 지령된 목표위치값과 보간동작의 결과 목표위치값이 불일치 할 수도 있습니다. WIMC 엔진은 정수체계로써 위치를 입력받기 때문 종점위치에 대한 오차가 1펄스 이내에서 발생될 수 있습니다. 또한 원호보간의 중심위치와 종점위치가 바르게 입력되지 않았다면 원호보간의 결과는 1펄스를 초과하는 종점불일치가 발생될 수 있습니다.

이럴 경우 WIMC 엔진은 원호보간의 동작처리 후에 강제로 지령된 종점위치로 이동시키는 옵션이 보간보상동작옵션입니다 (보간종점보상).
WIMC 엔진은 기본적으로 상대위치지령에 의해 동작되므로 1펄스등의 오차가 계속 누적되게 되면 최종 모션동작의 결과가 허용되지 않을 만큼의 큰 오차를 포함할 수 있게 됩니다. 그러므로 하나의 명령에 의한 오차량은 비교적 작기 때문에 하나의 명령이 종료된 후 최종 지령위치와 비교하여 동일하지 않을 때에 강제로 지령된 종점위치로 수렴시킴으로써 상대위치지령으로 인한 오차누적을 없앨 수 있게 됩니다.

WIMC 엔진은 종점보상이 2펄스 이상 발생하게 되면 알람을 발생시킬 수 있어 오차를 허용하지 않을 경우 동작을 정지시켜 해당 보간명령을 디버깅할 수 있게 도와줍니다.

<모션마스터프로그램내의 설정>



<리셋시 설정>

기능 ON

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	보간보상동작 기능 설정 상태입니다.
WIMC_FALSE	보간보상동작 기능 설정 상태가 아닙니다.

➲ Example

➲ See Also

```
WIMC_Enable_IEndComp (TRUE);                              // 보간보상기능 설정
WIMC_Enable_IEndComp (FALSE);                             // 보간보상기능 해제
WIMC_Set_AlarmOption( WIMC_ALARMOPTION_IENDCOMP); // 보간종점보상 알람발생 기능 설정
WIMC_Clr_AlarmOption( WIMC_ALARMOPTION_IENDCOMP); // 보간종점보상 알람발생 기능 해제
```

10. StateTimer

10.1. 상태타이머 (State Timer)

WIMC 엔진은 상태타이머기능을 통해서 상위제어기(PC)의 상태읽기 명령 없이 지정된 시간간격마다 정해진 정보들을 자동으로 전송하게 됩니다. 상태타이머에 의해서 모션컨트롤러와 상위제어기(PC)는 실시간으로 연결됩니다.

상태타이머의 동작은 Stop, 25Hz, 50Hz, 100Hz, 200Hz, 250Hz, 500Hz, 1000Hz 로 총 8가지 이며 리셋시 25Hz로 설정됩니다. 상태타이머의 동작이 빠를수록 통신속도를 높여 주어야만 원활하게 동작할 수 있습니다. .

상태타이머의 동작이 빠르면 스캔타임에 영향을 주기 때문에 200Hz이상의 고속모니터링을 사용할 경우 통신속도를 높여주어야 하며, 모션마스터프로그램을 통해서 스캔타임을 확인해 보는 것이 안전합니다.

상태타이머를 통해 전송되는 정보들은 선택할 수 있으며, 전송방식을 압축형태로 전송할 것인지에 대해서도 선택할 수 있습니다. 정보를 압축형태로 보내면 정보들의 변화를 감지하여, 선택적으로 전송되며 동일한 여러 정보들을 하나의 블록형태로 전송하게 됩니다. 압축전송방식을 사용하면 빠른 실시간 연결시에도 통신부하가 적어 스캔타임에 유리합니다.

기본적인 전송정보 이외에 모션엔진의 내부레지스터 중 하나를 임의적으로 전송할 수 있는 User Select정보 전송기능이 있습니다. 레지스터번호 Rxx(xx는 0~31)까지 선택할 수 있으며 각 레지스터번호에 대한 정보의 종류는 아래와 같습니다.

레지스터		정보설명	레지스터		정보설명
번호	이름		번호	이름	
R00	CHIPSET	모션엔진 주요 구성설정	R16	STARTVEL	시작속도(초속도) 정보설정
R01	CHIPSTATUS	모션엔진 주요 동작,신호상태	R17	DRIVEVEL	목표속도(구동속도) 정보설정
R02	AXISSET	모션엔진 축 구성설정	R18	CURVEL	현재속도 정보
R03	AXISSTATUS	모션엔진 축 동작,신호상태	R19	ACC	가속도 설정
R04	AXISAUXSET	모션엔진 축 부가설정	R20	DEC	감속도 설정
R05	GEARRATIO	전자기어비 정보	R21	ACC2	2차 가속도 설정
R06	CURPOS	현재위치 카운터정보	R22	DEC2	2차 감속도 설정
R07	CAL0	명령파라메터 0	R23	FEEDBACK	엔코더z카운터 및 서보피드백정보
R08	CAL1	명령파라메터 1	R24	HOMERETURN	원점복귀 동작 구성설정
R09	CAL2	명령파라메터 2	R25	COMARE	비교기 동작 구성설정
R10	CAL3	명령파라메터 3	R26	AUTOCAL	자동연산기능 구성설정
R11	PULSEOFFSET	펄스동작 설정정보	R27	PROGRAM	현재 사용하지 않음.
R12	ENCPOS	현재엔코더 카운터정보	R28	SERVO	서보입출력 구성설정
R13	SLIMITL	소프트리미트 하한설정값	R29	GIO	범용입출력 정보
R14	SLIMITH	소프트리미트 상한설정값	R30	EIO	확장입출력 정보
R15	VELAUXSET	속도 부가구성설정 정보	R31	CUSTOM	PID 와 같은 부가 컨틀로장치 정보

이들 정보는 아래의 기본적인 전송 정보와 겹치는 영역이 있으므로 아래의 정보이외의 다른 정보와 관련된 레지스터를 지정하는 것이 유용합니다.

아래에 상태타이머와 관련된 기능을 요약합니다.

항목	설정	설명 (관련 DLL함수포함)
전송속도	STOP, 25Hz, 50Hz 100Hz,200Hz,250Hz 500Hz, 1000Hz	WIMC_Set_StateTimer(WIMC_STATETIMER_xx); WIMC_Clr_StateTimer(WIMC_STATETIMER_xx); WIMC_Get_StateTimer(WIMC_STATETIMER_HZ); xx = STOP,25Hz,50Hz,100Hz,200Hz,250Hz,500Hz,1000Hz 중 하나
전송압축	Yes/No	WIMC_Set_StateTimer(WIMC_STATETIMER_COMPRESS); WIMC_Clr_StateTimer(WIMC_STATETIMER_COMPRESS); WIMC_Is_StateTimer(WIMC_STATETIMER_COMPRESS);
전송 정보	ChipStatus	모션엔진의 주요상태정보를 전송 WIMC_Set_StateTimer(WIMC_STATETIMER_CHIPSTATUS); WIMC_Clr_StateTimer(WIMC_STATETIMER_CHIPSTATUS); WIMC_Is_StateTimer(WIMC_STATETIMER_CHIPSTATUS);
	AxisStatus	모션엔진의 축상태정보를 전송 WIMC_Set_StateTimer(WIMC_STATETIMER_AXISSTATUS); WIMC_Clr_StateTimer(WIMC_STATETIMER_AXISSTATUS); WIMC_Is_StateTimer(WIMC_STATETIMER_AXISSTATUS);
	CurPosition	모션엔진의 현재위치정보를 전송 WIMC_Set_StateTimer(WIMC_STATETIMER_CURPOSITION); WIMC_Clr_StateTimer(WIMC_STATETIMER_CURPOSITION); WIMC_Is_StateTimer(WIMC_STATETIMER_CURPOSITION);
	EncPosition	모션엔진의 현재 엔코더정보를 전송 WIMC_Set_StateTimer(WIMC_STATETIMER_ENCPOSITION); WIMC_Clr_StateTimer(WIMC_STATETIMER_ENCPOSITION); WIMC_Is_StateTimer(WIMC_STATETIMER_ENCPOSITION);
	CurVelocity	모션엔진의 현재속도정보를 전송 WIMC_Set_StateTimer(WIMC_STATETIMER_CURVELOCITY); WIMC_Clr_StateTimer(WIMC_STATETIMER_CURVELOCITY); WIMC_Is_StateTimer(WIMC_STATETIMER_CURVELOCITY);
	GIO	모션엔진의 GIO혹은 EIO정보를 전송 WIMC_Set_StateTimer(WIMC_STATETIMER_GIO); WIMC_Clr_StateTimer(WIMC_STATETIMER_GIO); WIMC_Is_StateTimer(WIMC_STATETIMER_GIO);
	UserSelect	모션엔진의 기타 사용자 선택정보를 전송 WIMC_Set_StateTimer(WIMC_STATETIMER_USER_Rxx); WIMC_Clr_StateTimer(WIMC_STATETIMER_USER_Rxx); WIMC_Is_StateTimer(WIMC_STATETIMER_USER_Rxx); WIMC_Get_StateTimer(WIMC_STATETIMER_USER_REG);

<리셋시 설정상태>

상태타이머 Hz : 25Hz

상태타이머 압축모드 : Off

상태타이머 전송정보 : ChipStatus, AxisStatus, CurPosition, CurVelocity

10.1.1. WIMC_Set_StateTimer

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_StateTimer
(
              WIMC_UINT32             nFlags
);
```

➲ Description

상태타이머기능을 활성화 합니다.

관련 내용 및 설명은 (10.1 절)을 참고 하실수 있습니다..

➲ Parameters

nFlags

WIMC_STATETIMER_xx (xx = STOP,25Hz,50Hz,100Hz,200Hz,250Hz,500Hz,1000Hz 중 하나)
ex) WIMC_STATETIMER_25Hz
WIMC_STATETIMER_COMPRESSED
WIMC_STATETIMER_CHIPSTATUS
WIMC_STATETIMER_AXISSTATUS
WIMC_STATETIMER_CURPOSITION
WIMC_STATETIMER_ENCPOSITION
WIMC_STATETIMER_CURVELOCITY
WIMC_STATETIMER_GIO
WIMC_STATETIMER_USER_Rxx (xx=00 ~ 31 중 하나)
ex) WIMC_STATETIMER_USER_R01

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

WIMC_Set_StateTimer
WIMC_Get_StateTimer
WIMC_Clr_StateTimer
WIMC_Is_StateTimer

10.1.2. WIMC_Clr_StateTimer

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Clr_StateTimer
(
```

```
            WIMC_UINT32             nFlags
);
```

➲ Description

상태타이머 기능을 비활성화 합니다.

관련 내용 및 설명은 (10.1 절)을 참고 하실수 있습니다..

➲ Parameters

nFlags

WIMC_STATETIMER_xx (xx = STOP,25Hz,50Hz,100Hz,200Hz,250Hz,500Hz,1000Hz 중 하나)
ex) WIMC_STATETIMER_25Hz
WIMC_STATETIMER_COMPRESSED
WIMC_STATETIMER_CHIPSTATUS
WIMC_STATETIMER_AXISSTATUS
WIMC_STATETIMER_CURPOSITION
WIMC_STATETIMER_ENCPOSITION
WIMC_STATETIMER_CURVELOCITY
WIMC_STATETIMER_GIO
WIMC_STATETIMER_USER_Rxx (xx=00 ~ 31 중 하나)
ex) WIMC_STATETIMER_USER_R01

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

WIMC_Set_StateTimer
WIMC_Get_StateTimer
WIMC_Clr_StateTimer
WIMC_Is_StateTimer

10.1.3. WIMC_Get_StateTimer

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_StateTimer
(
              WIMC_UINT32             nFlags
);
```

➲ Description

WIMC_STATETIMER_HZ 플래그를 통해 전송속도를 읽습니다.

WIMC_STATETIMER_USER_REG 플래그를 통해 사용자 레지스터 번호(0~31)를 읽습니다.

관련 내용 및 설명은 (10.1 절)을 참고 하실수 있습니다..

➲ Parameters

nFlags

WIMC_STATETIMER_HZ
WIMC_STATETIMER_USER_REG

➲ Return Value

전송 속도를 반환합니다.(Hz = 0/25/50/100/200/250/500/1000)

사용자 레지스터 번호를 반환합니다.(0~31)

Return code/value	Description
WIMC_ERR	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

WIMC_Set_StateTimer
WIMC_Get_StateTimer
WIMC_Clr_StateTimer
WIMC_Is_StateTimer

10.1.4. WIMC_Is_StateTimer

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_StateTimer
(
              WIMC_UINT32             nFlags
);
```

➲ Description

상태타이머 설정 상태를 읽습니다.

관련 내용 및 설명은 (10.1 절)을 참고 하실수 있습니다..

➲ Parameters

nFlags

WIMC_STATETIMER_COMPRESSED
WIMC_STATETIMER_CHIPSTATUS
WIMC_STATETIMER_AXISSTATUS
WIMC_STATETIMER_CURPOSITION
WIMC_STATETIMER_ENCPOSITION
WIMC_STATETIMER_CURVELOCITY
WIMC_STATETIMER_GIO
WIMC_STATETIMER_USER_Rxx (xx=00 ~ 31 중 하나)
ex) WIMC_STATETIMER_USER_R01

➲ Return Value

nFlags 에 해당하는 상태타이머 설정 상태를 반환합니다.

Return code/value	Description
WIMC_TRUE	WIMC_STATETIMER_COMPRESSED : 전송압축 WIMC_STATETIMER_CHIPSTATUS : 모션엔진의 주요상태정보 전송 WIMC_STATETIMER_AXISSTATUS : 모션엔진의 축상태정보 전송 WIMC_STATETIMER_CURPOSITION : 모션엔진의 현재위치정보 전송 WIMC_STATETIMER_ENCPOSITION : 모션엔진의 현재 엔코더정보 전송 WIMC_STATETIMER_CURVELOCITY : 모션엔진의 현재속도정보 전송 WIMC_STATETIMER_GIO : 모션엔진의 GIO혹은 EIO정보 전송 WIMC_STATETIMER_USER_REG : 모션엔진의 사용자 선택정보 전송 가 Set 상태입니다.
WIMC_FALSE	WIMC_STATETIMER_COMPRESSED : 전송압축 WIMC_STATETIMER_CHIPSTATUS : 모션엔진의 주요상태정보 전송 WIMC_STATETIMER_AXISSTATUS : 모션엔진의 축상태정보 전송 WIMC_STATETIMER_CURPOSITION : 모션엔진의 현재위치정보 전송 WIMC_STATETIMER_ENCPOSITION : 모션엔진의 현재 엔코더정보 전송 WIMC_STATETIMER_CURVELOCITY : 모션엔진의 현재속도정보 전송 WIMC_STATETIMER_GIO : 모션엔진의 GIO혹은 EIO정보 전송 WIMC_STATETIMER_USER_REG : 모션엔진의 사용자 선택정보 전송 가 비활성화(Clear) 상태입니다.

➲ Example

➲ See Also

```
WIMC_Set_StateTimer
WIMC_Get_StateTimer
WIMC_Clr_StateTimer
WIMC_Is_StateTimer
```

11. Progress

11.1. Progress

11.1.1. WIMC_Get_CurCmdPulse

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_CurCmdPulse( void);
```

➲ Description

현재 실행중에 소요된 보간명령 펄스 수를 읽습니다.

➲ Parameters

➲ Return Value

현재 실행중에 소요된 보간명령 펄스 수를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Get_CmdPulseMax();          // 현재 실행중인 보간명령에서 소요되는 총 펄스수
WIMC_Get_CurCmdPulse();          // 현재 실행중에 소요된 보간명령 펄스수
```

11.1.2. WIMC_Get_CmdPulseMax

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_CmdPulseMax( void);
```

➲ Description

현재 실행중인 보간명령에서 소요되는 총 펄스수를 읽습니다.

➲ Parameters

➲ Return Value

현재 실행 중인 보간명령에서 소요되는 총 펄스수를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Get_CmdPulseMax();          // 현재 실행중인 보간명령에서 소요되는 총 펄스수
WIMC_Get_CurCmdPulse();          // 현재 실행중에 소요된 보간명령 펄스수
```

12. Scan Time

12.1. Scan Time

12.1.1. WIMC_Get_ScanTime

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ScanTime( void);
```

➲ Description

모션엔진의 처리성능을 나타내는 스캔타임값을 읽습니다. 0.1milliseconds 단위입니다.

ʊ 스캔타임 (Scan Time)

스캔타임이란 주처리 동작이 1회 실행되는데 소요했던 시간기록으로 단위는 0.1msec(100usec)입니다.
스캔타임은 모션엔진이 초당 실행할 수 있는 명령수와 직접적으로 관련됩니다. 모션엔진이 어떤 동작도 하지 않는 대기상태에 있다면 스캔타임은 일반적으로 0.1~0.3msec(평균 0.2msec)의 범위를 유지하여 1/2000초 이하의 속도로 명령을 처리할 수 있게 됩니다.
그러나 주변의 다양한 동작이 실행중이다면 스캔타임이 길어질 수 있으며, 명령의 처리속도는 스캔타임에 직접영향을 받게 됩니다. 스캔타임이 너무 커서 명령실행속도가 느려졌다면 1스캔타임내에서 명령을 처리하는 회수를 1~16회까지 임의적으로 늘일 수 있습니다. 이를 활용하여 명령처리속도를 (엔진의 동작한계만큼만) 높일 수도 있습니다.
원활한 동작을 위해서 스캔타임은 2msec이하로 유지해 주시는 것이 바람직합니다.

결국 스캔타임은 전체적인 엔진의 동작 부하량을 지시합니다. 그러므로 스캔타임이 커지는 것은 모션엔진의 코어(core)에 해당하는 마이크로프로세서의 연산량이 많아 부하를 많이 받고 있다는 것을 알려줍니다.
WDT(Watch Dog Time) 알람을 활성화 하면 내부적으로 스캔타임의 평균을 구하여 특정값(엔진내부에서 임의적으로 정해놓은값)보다 클 때 알람을 발생시켜 비상정지동작을 수행할 수 있습니다.

➲ Parameters

➲ Return Value

스캔타임값을 0.1 milliseconds 단위로 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

13. CExe

13.1. 연속실행모드와 비연속 실행모드

WIMC 엔진의 연속실행모드와 비연속실행모드는 입력된 모든 보간명령들의 속도연속성 유지하기 위한 내부동작 유무를 말합니다.
비연속실행모드에서는 모든 명령이 각각 종료된 후에 다음 명령이 실행됩니다.
비연속실행모드에서는 명령큐의 명령실행이 하나씩 순차적으로 실행됨으로 상기 명령큐에 의한 보간명령과 설정명령들간의 실행순서 바뀜현상이 발생될 우려가 없습니다.

WIMC 엔진의 연속실행동작과 비연속실행동작을 설정하기 위해서 아래의 함수를 사용할 수 있습니다.
- 연속실행모드 : WIMC_Enable_CExe(TRUE);
- 비연속실행모드 : WIMC_Enable_CExe(FALSE);

모션시스템이 동작전체가 연속실행동작이 주로 이루어지며, 중간 중간에 설정명령이나 혹은 다른 이동명령들은 이전명령이 완료된 후에 실행하게 됩니다. 연속실행모드에서 특정 순간에 잠시 명령의 동기성을 필요로 한다면 비연속실행모드의 전환없이 간단히 SYNC명령을 사용하여 이전 모션동작이 완료될 때까지 명령큐의 모든 명령들을 실행대기시킬 수 있습니다. 이 명령을 통해 연속실행모드를 유지하면서 필요할 때마다 설정명령들을 동기화 시킬 수 있게 됩니다. 이와 관련된 DLL함수는 WIMC_Sync()입니다.

13.1.1. WIMC_Enable_Cexe

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_CExe
(
              WIMC_BOOL               bEnable
);
```

➲ Description

연속실행 모드를 설정(Enable/Disable) 합니다.

관련 내용 및 설명은 (13.1 절)을 참고 하십시오.

➲ Parameters

bEnable

WIMC_TRUE : 연속실행모드로 설정합니다.

WIMC_FALSE : 비연속 실행모드로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

WIMC_CExeOff

13.1.2. WIMC_Is_CExe

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_CExe( void);
```

➲ Description

연속실행 모드를 확인합니다.

관련 내용 및 설명은 (13.1 절)을 참고 하십시오.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	연속실행 모드 상태입니다.
WIMC_FALSE	연속실행 모드 상태가 아닙니다.

➲ Example

➲ See Also

14. Step Exe

14.1. 스텝실행

스텝실행

정상실행상태에서 명령큐를 실행하는 방법 중 스텝실행방식이 있습니다.
스텝실행은 STEP ON명령을 전송함으로써 실행됩니다. 스텝실행이란 명령큐로부터 명령을 하나씩 실행하고 일시정지하는 실행을 말합니다. 일시정지된 상태에서 다음 명령큐 하나를 실행하려면 RESUME명령을 사용합니다. CANCEL명령을 사용하면 다음 명령큐를 실행하지 않고 버리게 됩니다.
이 기능은 일련의 모션명령동작들을 한 스텝씩 디버깅 할 수 있도록 허용합니다.
스텝실행을 OFF하려면 STEP OFF명령을 전송하면 됩니다. STEP OFF명령이 전송됨과 동시에 명령큐의 명령들이 연속적으로 실행됩니다.

아래에 스텝실행과 관련된 라이브러리 함수들을 나열합니다.

```
WIMC_Enable_StepExe(TRUE); // STEP 실행 시작 명령전송
WIMC_Enable_StepExe(FALSE); // STEP 실행 종료 명령전송
WIMC_Is_StepExe(); // 스텝실행 여부 읽기
WMC_Resume(); // RESUME 명령
WIMC_Cancel(); // CANCEL명령
```

스텝실행 명령

스텝실행명령을 실행하면 모션엔진은 스텝실행모드로 전환합니다.
스텝실행모드가 되면 명령큐의 명령을 하나씩 읽어들여 실행 한 후 정지하게 됩니다. 이때 [RESUME]명령을 즉시실행명령으로 전송하게 되면 다음 명령큐의 명령을 하나를 읽어들여 실행한 후 다시 정지하게 됩니다. 만약 [RESUME]명령 대신 [CANCEL]을 즉시실행명령으로 실행하게 되면 다음 명령큐의 명령을 읽어들인 후 실행하지 않고 버려진 후 다시 대기상태가 됩니다.
스텝실행모드를 해제하는 즉시 명령큐의 명령들은 연속적으로 실행하게 됩니다.

14.1.1. WIMC_Enable_StepExe

➲ Syntax

```
WIMC_BOOL    WIMC_API                    WIMC_Enable_StepExe
(
             WIMC_BOOL                   bEnable
);
```

➲ Description

스텝실행모드를 Enable/Disable 합니다.

관련 내용 및 설명은 (14.1절)을 참고 하실수 있습니다..

➲ Parameters

bEnable
WIMC_TRUE : Enable.
WIMC_FALSE : Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

14.1.2. WIMC_Is_StepExe

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_StepExe( void);
```

➲ Description

스탭실행상태인지 확인합니다.

관련 내용 및 설명은 (14.1절)을 참고 하실수 있습니다..

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	스텝실행 상태입니다.
WIMC_FALSE	스텝실행 상태가 아닙니다.

➲ Example

➲ See Also

15. Delay

15.1. 동작지연명령

동작지연명령 [DELAY]

동작지연명령은 msec단위로 설정된 시간지연값 만큼 명령실행을 중지하도록 지시합니다. [DELAY] 명령상태에서도 즉시실행명령은 실행가능함으로 엄밀히 말해서 명령실행을 중지하지는 못합니다. [DELAY]명령은 즉시실행명령으로는 실행되지 않습니다.
[DELAY]명령이 실행되면 모션엔진은 Delay실행상태가 되며 Delay타이머가 작동됩니다. Delay타이머의 작동은 msec단위 이며 현재타이머값을 읽어들 일 수 있습니다. 딜레이시간지정은 32비트인데 비해 16비트까지만 읽어들 수 있습니다.
Delay실행상태를 중간에 정지시키고 다음 명령으로 넘어가려면 [RESUME]명령이나 [CANCEL]명령을 사용할 수 있습니다.
※즉시실행명령방식일 경우 아무런 기능을 하지 않습니다.

조건지연 (XDelay : eXtenstion Delay)

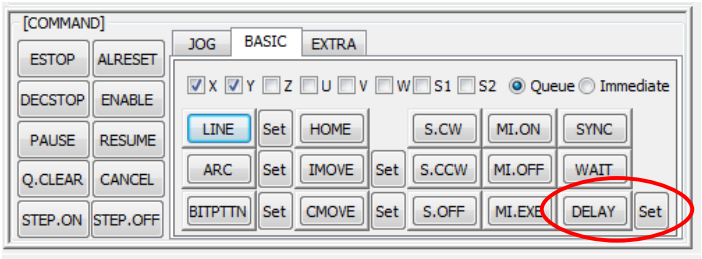
XDelay명령은 위치제어명령은 아니지만 모션제어와 관련되어 자주 사용될 수 있는 명령으로 특정 조건이 만족될 때까지 지연하는 동작을 실행합니다. 일반적인 Delay명령에 조건을 부여한 명령입니다.
내부적으로 Delay명령의 확장으로 만들어져 XDelay형식의 명령이름을 취하지만으로 동작은 조건기능을 갖는 대기(Wait)명령으로 이해할 수 있습니다.

조건은 아래의 총 6가지입니다.
(1) Sync신호입력 대기
(2) 원점복귀완료 대기 (지정된 모든축)
(3) 동작완료(Complete신호) 대기 (지정된 모든축)
(4) 입력포트매칭 대기 (입력포트값의 1값과 매칭)
(5) 동작속도 안정 대기 (지정된 모든축)
(6) 값비교 대기

이들 조건상황이 지정된 시간에 만족되지 않으면 그냥 통과시키거나 알람을 발생시킬 수 있습니다.
이때 알람은 COMMAND알람과 PROGRAM알람으로 설정됩니다.
여러조건을 동시에 지정하면 OR된 조건결과로 DELAY동작이 실행됩니다.

※주> SYNC, WAIT, DELAY 와 같은 대기형 명령들은 즉시실행명령에서는 아무런 효과가 없으며 오직 명령큐에서 실행될 때만 제 기능을 실행합니다.

<모션마스터프로그램내의 명령>



<관련 라이브러리함수>

WIMC_Delay(); WIMC_XDelay(…); // 다양한 조건에 의해 종료될 수 있는 시간지연함수.

15.1.1. WIMC_Delay

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Delay
(
              WIMC_INT32                msec
);
```

➲ Description

동작지연 명령입니다.

관련 내용 및 설명은 (15.1절)을 참고 하실수 있습니다..

➲ Parameters

msec

milliseconds 단위입니다.
DELAY시간값은 msec단위로 지정이 가능합니다.
최대 대략 2,147,483초로써 대략 25일(day)에 해당하는 시간지연 값입니다

➲ Return Value

Return code/value	Description

➲ Example

```
WIMC_Enter_QueueCmd();
WIMC_Sync();
WIMC_Delay( 500);            // 500 mSec 동작 지연
WIMC_Leave_QueueCmd();
```

➲ See Also

15.1.2. WIMC_XDelay

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_XDelay
(
              WIMC_INT32              msec,
              WIMC_UINT32             cond,
              WIMC_UINT8              axis,
              WIMC_UINT32             comparePortMask,
              WIMC_INT32              compareVal,
              WIMC_INT32              compareValH
);
```

➲ Description

XDelay명령은 위치제어명령은 아니지만 모션제어와 관련되어 자주 사용될 수 있는 명령으로 특정 조건이 만족될 때까지 지연하는 동작을 실행합니다. 일반적인 Delay명령에 조건을 부여한 명령입니다.
내부적으로 Delay명령의 확장으로 만들어져 XDelay형식의 명령이름을 취하지만으로 동작은 조건기능을 갖는 대기(Wait)명령으로 이해할 수 있습니다.

<표> 조건 플랙 32비트 (31~20)

31	30	29	28	27	26	25	24	23	22	21~20
조건 사용 유무	알람 발생	SYNC 대기	원점 복귀 완료 대기	축동작 완료 대기	입력 포트 매칭	입력 포트 레벨 반전	입력 포트 조건 연산	속도 안정 대기	예약	비교 동작
0=사용 안함 1=사용	0=발생 안함 1=발생	0=대기 안함 1=대기	0=대기 안함 1=대기	0=대기 안함 1=대기	0=대기 안함 1=대기	0=H 1=L	0 = OR 1 = AND	0=대기 안함 1=대기	-	00=동작안함 01=GE(>=) 10=LE(<=) 11=SCOPE (범위)

<표> 조건 플랙 32비트 (19~0)
19-0비트의 내용은 WIMC 엔진의 코어내부 레지스터에 관련된 내용이므로 다소 복잡할 수 있습니다. 그러나 비트정의된 매크로를 사용하면 간단하게 설정할 수 있으므로 여기서는 단순히 비트들의 의미만을 살펴봅니다.

19~18	17~16	15~8	7~0
비교값 데이터형	비교대상 데이터형	비교값 레지스터 지정	비교대상 레지스터 지정
00=바이트 01=하위워드 10=상위워드 11=더블워드	00=바이트 01=하위워드 10=상위워드 11=더블워드	레지스터 지정바이트구조참조	레지스터 지정바이트 구조참조

- 레지스터지정바이트구조

7	6	5	4	3	2	1	0
레지스터번호				축번호			
R00~R31 = 0~31로 지정				000=X 001=Y			

각 레지스터의 정의는 WIMC_Def.h 파일을 참조하세요	010=Z 011=U 100=V 101=W 110=S1 111=S2

이들 비트들은 아래와 같이 매크로로 정의되어 있습니다.

<조건플래그 정의 매크로명>

```
WIMC_XDELAY_ENABLE                  //조건동작 허가
WIMC_XDELAY_ENABLE_ALARM            //조건이 만족되지 않을 때 알람발생 허가
WIMC_XDELAY_COND_CLEAR              //모든 조건 클리어
WIMC_XDELAY_COND_SYNC               //외부 SYNC입력대기 조건
WIMC_XDELAY_COND_HOME               //원점복귀 완료대기 조건
WIMC_XDELAY_COND_COMPLETE           //동작완료 대기조건
WIMC_XDELAY_COND_GIN                //GIO입력값 비교조건
WIMC_XDELAY_COND_GIN_INV_LEVEL      //GIO입력값 반전
WIMC_XDELAY_COND_GIN_OR             //GIO입력포트의 해당 비트값 OR비교
WIMC_XDELAY_COND_GIN_AND            //GIO입력포트의 해당 비트값 AND비교
WIMC_XDELAY_COND_VELSTEADY          //속도안정화 대기조건

// 이하 비교값 조건을 사용하는 XDelay()를 위한 매크로정의입니다.
// 동작방식 GE/LE/SCOPE
WIMC_XDELAY_COND_COMPARE_GE         //값비교조건(비교대상 >= 비교값) 동작
WIMC_XDELAY_COND_COMPARE_LE         //값비교조건(비교대상 <= 비교값) 동작
WIMC_XDELAY_COND_COMPARE_SCOPE      //값비교조건(하한비교값<=비교대상<=상한비교값) 동작허가
// 비교대상
WIMC_XDELAY_COND_COMPARE_POS        //위치값 비교허가 (▶비교축정의와 함께사용)
WIMC_XDELAY_COND_COMPARE_ENC        //엔코더값 비교허가 (▶비교축정의와 함께사용)
WIMC_XDELAY_COND_COMPARE_VEL        //속도값 비교허가
WIMC_XDELAY_COND_COMPARE_VELAXIS    //축속도값 비교허가 (▶비교축정의와 함께사용)
WIMC_XDELAY_COND_COMPARE_GIN        //입력포트값 비교허가
WIMC_XDELAY_COND_COMPARE_ADC0       //ADC0값 비교허가
WIMC_XDELAY_COND_COMPARE_ADC1       //ADC1값 비교허가
WIMC_XDELAY_COND_COMPARE_ADC2       //ADC2값 비교허가
WIMC_XDELAY_COND_COMPARE_ADC3       //ADC3값 비교허가
WIMC_XDELAY_COND_COMPARE_GIOENC0    //GIO포트용 엔코드0값 비교허가
WIMC_XDELAY_COND_COMPARE_GIOENC1    //GIO포트용 엔코드1값 비교허가
WIMC_XDELAY_COND_COMPARE_GIOCNT0    //GIO포트용 카운터0값 비교허가
WIMC_XDELAY_COND_COMPARE_GIOCNT1    //GIO포트용 카운터1값 비교허가
// 비교대상축 : POS, ENC, VELAXIS 의 비교조건사용시 아래의 매크로를 사용하여 축을 지정합니다.
WIMC_XDELAY_COND_COMPARE_X          // X축 (0)
WIMC_XDELAY_COND_COMPARE_Y          // Y축 (1)
WIMC_XDELAY_COND_COMPARE_Z          // Z축 (2)
WIMC_XDELAY_COND_COMPARE_U          // U축 (3)
WIMC_XDELAY_COND_COMPARE_V          // V축 (4)
WIMC_XDELAY_COND_COMPARE_W          // W축 (5)
```

```
WIMC_XDELAY_COND_COMPARE_S1          // S1축 (6)
WIMC_XDELAY_COND_COMPARE_S2          // S2축 (7)
```

※ 동작완료(COMPLETE) 조건과 속도안정화(VELSTEADY)조건 동작시 축지정값 axis 가 0 혹은 WIMC_NONE 으로 지정했을 경우 개별축동작이 아닌 전체보간동작의 동작완료상태와 속도안정화상태에 의해 조건동작이 실행됩니다.

➲ Parameters

delayTime
: 조건 대기 최대시간

axis
: 비교값 동작이외의 대기동작에 관련되는 축을 설정.

conditions
: 조건 종류 설정 플랙 32비트 (아래의 표와 함께 상세내용들을 참조하세요)

comparePortMask
: 조건동작 될 GIN 포트값이 1(H레벨)이 되는 비트들에 대한 비트마스크(16비트)

compareValue
: 값비교조건일 때 비교기준값 (값범위비교동작일 때에는 하한비교값)

compareValueH
: 값범위비교동작일 때 상한비교값

➲ Return Value

Return code/value	Description

➲ Example

➲ Example

➲ 코드 예 1> ADC 0 의 입력값이 10000 보다 클 때까지 최대 10 분간 대기하며 지정된 시간내에 조건이 만족되지 않으면 알람이 발생하는 코드.

```
UINT32 cond;
cond = WIMC_XDELAY_COND_CLEAR;          // 조건 클리어
cond |= WIMC_XDELAY_ENABLE;                    // 조건 동작 가능
```

```
cond |= WIMC_XDELAY_ENABLE_ALARM; // 조건 불만족시 알람 발생
cond |= WIMC_XDELAY_COND_COMPARE_GE;       // >= 값비교 조건 동작
cond |= WIMC_XDELAY_COND_COMPARE_ADC0; // 비교대상 ADC0
WIMC_XDelay(
                10 * 60 * 1000, // 10분->msec
                cond, // 조건동작 설정
                0x00, // 축지정 필요 없음
                0x00, // 입력포트마스크 없음
                10000, // 비교값 10000 이상일 때 조건만족
                0  // 범위(SCOPE)비교가 아니므로 비교상한값 사용 안함 );
```

➲ 코드 예 2> Z 축이 원점복귀 완료될 때까지 최대 30 초간 대기하고, 지정된 시간 초과시 알람이 발생하는 코드.

```
UINT32 cond;

cond = WIMC_XDELAY_COND_CLEAR;
cond |= WIMC_XDELAY_ENABLE;
cond |= WIMC_XDELAY_ENABLE_ALARM;
cond |= WIMC_XDELAY_COND_HOME;
WIMC_XDelay(
                30 * 1000, // 30초->msec
                cond,    // 조건동작 설정
                WIMC_Z, // Z축 지정
                0X00, // 입력포트 마스크 없음
                0, // 비교값 사용안함
                0, // 비교상한값 사용안함
                );
```

15.1.3. WIMC_Get_CurDelayTime

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_CurDelayTime( void);
```

➲ Description

모션엔진이 Delay명령에 의해 대기상태일 때 현재 대기진행시간을 msec단위로 읽습니다.

➲ Parameters

➲ Return Value

현재 대기진행시간을 msec단위로 반환합니다.

Return code/value	Description
WIMC_ERR	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

15.1.4. WIMC_Sleep

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Sleep
(
              WIMC_INT32                msec
);
```

➲ Description

메시지 루프 실행하면서(윈도우 메시지 처리) 프로그램 지연(Delay)을 합니다.

내부 구현은 아래와 같습니다.

```
MSG msg;
DWORD dwStart;
dwStart = GetTickCount();

while(GetTickCount() - dwStart < msec)
{
        Sleep(1);
        while(PeekMessage(&msg,NULL,0,0,PM_REMOVE))
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
}
```

➲ Parameters

msec

milliseconds 단위입니다.

DELAY시간값은 msec단위로 지정이 가능합니다.

➲ Return Value

Return code/value	Description

➲ Example

```
WIMC_Enter_QueueCmd();
WIMC_Sync();
WIMC_Delay( 500);          // 500 mSec 동작 지연
WIMC_Leave_QueueCmd();
```

➲ See Also

16. Sync

16.1. 명령 동기

명령 동기명령[SYNC]은 현재실행중이 명령이 종료되기를 기다리는 명령입니다. 이 명령은 현재실행중인 명령을 완전히 완료한 후에 명령큐의 다음명령을 실행하도록 허락합니다. 이 명령은 연속실행상태일 때 명령간의 일시적인 동기실행상태를 만들어줍니다. 그러므로 연속실행상태에서 명령들의 동기를 위한 비연속실행상태 명령이 잠시 개입될 때 SYNC명령을 선두에 두어 간단하게 처리할 수 있게 됩니다.

※주> SYNC, WAIT, DELAY 와 같은 대기형 명령들은 즉시실행명령에서는 아무런 효과가 없으며 오직 명령큐에서 실행될 때만 제 기능을 실행합니다.

※연속실행상태에서 명령들 간의 동기성을 맞추는 방식 중 하나가 SYNC 명령에 의한 명령동기방식이며 다른 방식이 명령바인딩에 의한 동기방식입니다.

동기신호(Sync signal)

모션엔진이 대기명령(WAIT COMMAND)을 실행하면 동기신호(Sync Signal)가 Low 레벨 상태가 될 때까지 대기합니다. 동기신호가 Low 레벨 상태가 되면 대기명령(WAIT COMMAND)은 실행이 종료되고 다음명령들을 실행합니다.
동기신호는 다른 장비와 동작시점을 맞추는 등의 목적에 사용됩니다. 동기신호의 검출속도는 명령실행주기를 표시하는 스캔타임(ScanTime)과 관련됩니다. 스캔타임은 현재 모션엔진의 동작상태에 따라 달라지며, 대부분 1msec이하에서 동작하는 것이 안전합니다. 만약 모션엔진이 다른 실행동작 없이 대기하고 있다면 스캔타임은 200 usec 내외로 실행될 수 있어 외부기기간의 동기시간을 1/1000초 이하로 유지될 수 있습니다. 그렇지 않다면 모션엔진이 감지할 수 있는 충분한 시간(스캔타임의 최소 2배)동안 Low 레벨을 유지해 주어야 합니다. (스캔타임은 "모션마스터" 프로그램에서 모니터링 할 수 있습니다.)

16.1.1. WIMC_Sync

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Sync( void);
```

➲ Description

Sync 명령을 전송합니다.

관련 내용 및 설명은 (16.1절)을 참고 하실수 있습니다..

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

16.1.2. WIMC_Is_Sync

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_Sync( void);
```

➲ Description

모션엔진의 SYNC 신호핀의 상태를 확인합니다.

관련 내용 및 설명은 (16.1절)을 참고 하실수 있습니다..

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	SYNC 신호가 발생했습니다.
WIMC_FALSE	SYNC 신호가 발생하지 않았습니다.

➲ Example

➲ See Also

17. Wait

17.1. 대기명령

[WAIT]명령은 외부의 동기신호(Sync)가 발생하기 기다리는 명령입니다. 이 명령은 다수의 기기들간에 동기동작을 위해 준비된 명령입니다. WAIT명령이 실행되면 모션엔진은 WAIT명령상태가 되어 모션엔진의 Sync신호핀의 활성화 상태를 기다리게 됩니다. 이 상태에서 명령큐의 명령들만 대기하므로 즉시실행명령의 실행은 가능합니다. 만약 외부신호가 오랫동안 발생되지 않아 [RESUME] 혹은 [CANCEL]명령을 즉시실행으로 전송하면 Wait상태에서 빠져나오게 됩니다.

※주> SYNC, WAIT, DELAY 와 같은 대기형 명령들은 즉시실행명령에서는 아무런 효과가 없으며 오직 명령큐에서 실행될 때만 제 기능을 실행합니다.

동기신호(Sync signal)

모션엔진이 대기명령(WAIT COMMAND)을 실행하면 동기신호(Sync Signal)가 Low 레벨 상태가 될 때까지 대기합니다. 동기신호가 Low 레벨 상태가 되면 대기명령(WAIT COMMAND)은 실행이 종료되고 다음명령들을 실행합니다.
동기신호는 다른 장비와 동작시점을 맞추는 등의 목적에 사용됩니다. 동기신호의 검출속도는 명령실행주기를 표시하는 스캔타임(ScanTime)과 관련됩니다. 스캔타임은 현재 모션엔진의 동작상태에 따라 달라지며, 대부분 1msec이하에서 동작하는 것이 안전합니다. 만약 모션엔진이 다른 실행동작 없이 대기하고 있다면 스캔타임은 200 usec 내외로 실행될 수 있어 외부기기간의 동기시간을 1/1000초 이하로 유지될 수 있습니다. 그렇지 않다면 모션엔진이 감지할 수 있는 충분한 시간(스캔타임의 최소 2배)동안 Low 레벨을 유지해 주어야 합니다. (스캔타임은 "모션마스터" 프로그램에서 모니터링 할 수 있습니다.)

17.1.1. WIMC_Wait

➲ Syntax

```
void          WIMC_API                  WIMC_Wait( void);
```

➲ Description

Wait 명령을 전송합니다.

[WAIT]명령은 외부의 동기신호(Sync)가 발생하기 기다리는 명령입니다. 이 명령은 다수의 기기들간에 동기동작을 위해 준비된 명령입니다. WAIT명령이 실행되면 모션엔진은 WAIT명령상태가 되어 모션엔진의 Sync신호핀의 활성화 상태를 기다리게 됩니다. 이 상태에서 명령큐의 명령들만 대기하므로 즉시실행명령의 실행은 가능합니다. 만약 외부신호가 오랫동안 발생되지 않아 [RESUME] 혹은 [CANCEL]명령을 즉시실행으로 전송하면 Wait상태에서 빠져나오게 됩니다.

※주> SYNC, WAIT, DELAY 와 같은 대기형 명령들은 즉시실행명령에서는 아무런 효과가 없으며 오직 명령큐에서 실행될 때만 제 기능을 실행합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

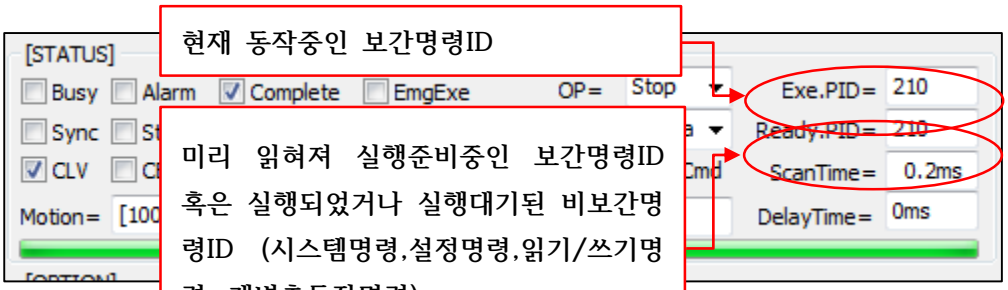
➲ Example

➲ See Also

18. Wait Motion

18.1. 명령 ID 설정과 Wait Motion

WIMC 엔진에 전달되는 모든 명령은 내부적으로 8비트의 명령ID값을 갖고 있습니다.
이 명령ID값은 아래의 모션마스터프로그램의 모니터링화면에서 확인할 수 있습니다.



(상태타이머에 [illegible]고 있는 것으로 가정합니다.)

아래의 함수를 통해서 다음에 이어지는 명령의 ID값이 설정되게 됩니다.
WIMC_Set_CmdID(idVal);
… 이후 명령들은 모두 idVal값으로 명령ID가 설정됩니다.

모니터링화면의 Exe.PID (Packet ID)값은 현재 동작중인 보간동작명령에 대한 명령ID값으로
WIMC_Get_ExeCmdID() 함수를 읽을 수 있습니다.
Ready.PID로 표시되는 값은 아래 둘 중의 하나에 대한 명령ID값입니다.

(1) 보간명령들 중 다음 실행을 위해 미리읽혀져 실행준비중인 명령ID
(2) 보간명령을 제외한 명령들 중 실행되었거나 실행보류중인 명령ID

Ready.PID값은 WIMC_Get_ReadyCmdID() 함수로 읽을 수 있습니다.

※ WIMC 엔진의 명령 ID 값은 함수의 인자형과 관계없이 8 비트만을 사용합니다. 그러므로 0~255 번까지 사용할 수 있습니다. 그러나 201 번부터는 내부적으로 특별한 신호의미로 사용되도록 예약되었으므로 임의적으로 사용할 수 있는 명령 ID 값은 최종 0~200 까지입니다.

WIMC 엔진과 관련된 함수들은 기본적으로 자신의 명령이 실행완료될 때까지 대기하지 않고 바로 리턴하게됩니다. 대부분의 일반적인 모션명령들은 명령큐를 사용할 것이며 명령큐의 명령들이 동기를 유지하기 위해서 WIMC_Sync()명령을 사용하거나 비연속실행모드를 사용함으로써 순차적으로 실행할 수 있게 됩니다.

그러나 특정 모션동작이 완료된 후 WIMC 엔진의 동작과 무관한 다른동작을 수행하려하면 반드시 현재모션동작의 완료를 대기하여야 합니다.

이러한 경우에 명령ID와 WIMC_WaitMotion(id, msec)함수를 조합하여 사용할 수 있습니다.

아래에 간략한 사용예를 제시합니다.(명령큐실행으로 가정합니다.)

```
WIMC_Enable_Notify_Complete(TRUE);        // 동작완료신호를 보낼수 있게 설정합니다.
WIMC_Set_CmdID(100);  // 다음의 명령들은 ID값을 100으로 설정됩니다.
WIMC_Line( WIMC_XY, xyPos ); // XY평면상에 ID값이 100인 Line명령이 실행됩니다.
WIMC_Set_CmdID(110); // 다음명령은 ID 110값으로 설정됩니다.
```

```
WIMC_Line( WIMC_Z, zPos); // Z축을 이동시킵니다.
WIMC_WaitMotion(100, 60000); // ID 100 인 모션동작의 완료를 최대 60000msec(=60초)동안 대기합니다.
WIMC_Enable_Notify_Complete(FALSE); //  더 이상 동작완료신호를 보내지 않도록 설정합니다.
```

※ WIMC_WaitMotion()함수는 내부적으로 윈도우메시지처리동작을 수행함으로써 일반적인 윈도우의 이벤트는 그대로 실행되면서 대기합니다.

동작의 완료상태를 응답받기 가장 먼저 WIMC_Enable_Notify_Complete(TRUE)함수를 통해서 모션명령 동작완료시 정보를 전송하도록 허가해야 합니다.
그러나 이 옵션을 항상 허가하게 되면 초당 수백번 이상의 짧은 거리를 연속해서 움직일 때 매 순간 동작완료상태를 전송하기 때문에 갑작스러운 통신부하 및 스캔타임지연을 가져올 수 있습니다. 그러므로 더 이상 완료상태전송이 필요치 않을 경우 WIMC_Enable_Notify_Complete(FALSE)함수로써 완료정보전송을 금지하는 것이 바람직합니다.

18.1.1. WIMC_Enable_Notify_Complete

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Enable_Notify_Complete
(
              WIMC_BOOL                 bEnable
);
```

➲ Description

동작 완료신호를 보낼 수 있게 설정(Enable/Disable)합니다.

관련 내용 및 설명은 (18.1절)을 참고 하실수 있습니다..

➲ Parameters

bEnable

WIMC_TRUE : 동작 완료 신호를 보낼 수 있도록 합니다.

WIMC_FALSE : 동작 완료 신호를 보낼 수 없도록 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

18.1.2. WIMC_Notify_Complete

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Notify_Complete
(
              WIMC_INT32              cmdId
);
```

➲ Description

모션명령 ID 값(cmdId)으로 동작완료 상태를 전송시킵니다.

관련 내용 및 설명은 (18.1절)을 참고 하실수 있습니다..

➲ Parameters

cmdId

명령 ID 값입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

18.1.3. WIMC_WaitMotion

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_WaitMotion
(
              WIMC_UINT8              motionId,
              WIMC_INT32              timeout
);
```

➲ Description

지정 대기시간(timeout) 이내에 모션명령 ID 명령(motionId)이 완료되기를 기다립니다.

지정시간이 경과된 동안 명령완료패킷이 전송되지 못하면 FALSE값을 리턴합니다. 정상적으로 대기완료되면 TRUE값을 리턴합니다.

관련 내용 및 설명은 (18.1절)을 참고 하실수 있습니다..

➲ Parameters

motionId
모션명령 ID

timeout
대기시간입니다. milliseconds 단위입니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	명령완료패킷이 전송되었습니다.
WIMC_FALSE	명령완료패킷의 전송되지 않았습니다.

➲ Example

➲ See Also

18.1.4. WIMC_Register_Complete_Message

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Register_Complete_Message
(
              HWND                    h,
              WIMC_UINT32             message
);
```

➲ Description

모션동작이 완료될 때 특정윈도우에 메시지를 발생할 수 있습니다.

사용자 메시지응답함수내의 메시지정보 중 아래의 항목에서 동작 완료된 명령id값을 얻을 수 있습니다.

```
UINT8 completeCmdID;
MSG msg;                          // 메시지 구조체
completeCmdID = msg.lParam ;      // 하위 8비트 값이 동작완료된 명령 id값이 됩니다.
```

동작완료 메시지 등록을 삭제하려면 아래의 함수를 사용할 수 있습니다.

```
WIMC_Unregister_Complete_Message();
```

➲ Parameters

h

윈도우 핸들 값입니다.

message

발생시키고자 하는 메세지를 WIMC_UINT32 타입으로 명시합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

18.1.5. WIMC_Unregister_Complete_Message

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Unregister_Complete_Message( void);
```

➲ Description

동작완료 메시지 등록을 삭제합니다.

WIMC_Register_Complete_Message() 함수를 참고 하실 수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

19. Command State

19.1. 명령 실행 상태

19.1.1. WIMC_Get_CmdState

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_CmdState( void);
```

➲ Description

모션엔진의 현재명령실행상태를 읽습니다.

➲ Parameters

➲ Return Value

모션엔진의 현재 명령실행상태를 반환합니다.

Return code/value	Description
WIMC_CMDSTATE_NORMAL	현재명령이 없거나 일반적인 명령의 실행상태입니다.
WIMC_CMDSTATE_SYNC	SYNC명령에 의해서 이전 모션동작이 완료되기를 기다리고 있는 상태입니다.
WIMC_CMDSTATE_WAIT	WAITt명령에 의해서 외부 SYNC신호를 대기하고 있는 상태입니다.
WIMC_CMDSTATE_DELAY	DELAY명령에 의해서 대기하고 있는 상태입니다.
WIMC_ERR	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

19.1.2. WIMC_Is_PreparedCmd

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_PreparedCmd( void);
```

➲ Description

모션엔진이 현재 선행처리되어 실행준비를 마친 모션동작명령이 존재함을 확인합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	명령이 존재합니다.
WIMC_FALSE	명령이 존재하지 않습니다.

➲ Example

➲ See Also

20. Motion State

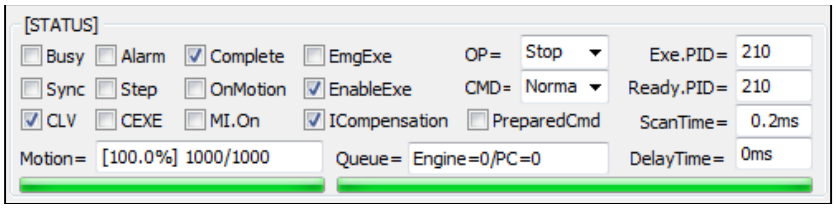
20.1. WIMC 엔진의 주요상태값들

WIMC 엔진은 상태타이머의 자동 정보전송 기능에 의해 모션엔진의 다양한 상태값들을 실시간으로 읽을 수 있습니다.
이들 상태값들은 아래와 같이 3가지 종류로 나눌 수 있습니다.

① 모션엔진 전체기능에 대한 주요한 동작 상태값들
② 각 축별 동작 상태값들
③ 범용입출력과 부가장치기능들에 대한 동작 상태값들

여기서는 ①번의 모션엔진 전체기능에 대한 주요한 동작 상태값들을 살펴보고 나머지 상태값들은 각 기능별 설명편에서 다룹니다.

<모션마스터프로그램내의 엔진 주요상태값확인>



20.1.1. WIMC_Is_Busy

➲ Syntax

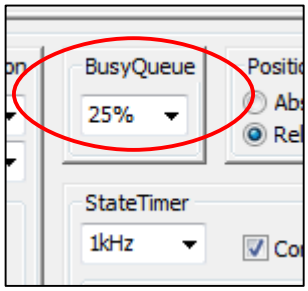
```
WIMC_BOOL     WIMC_API                WIMC_Is_Busy( void);
```

➲ Description

Busy 상태를 확인합니다.

BUSY신호는 WIMC 엔진의 내부 명령큐의 총 (바이트)크기 중에서 수신되었지만 처리되지 않고 남아있는 명령큐의 명령들이 차지하는 비중이 지정된 %보다 클 때 발생됩니다. 이 지정은 25%, 50%, 75%, 100% 중 4가지를 지정할 수 있습니다. 일반적으로 25%~50%를 설정합니다.
WIMC 라이브러리의 명령전송은 BUSY신호에 의해 자동 조절됩니다.

<모션마스터프로그램내의 설정>



<리셋시 설정상태>
25%

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	Busy 상태입니다.
WIMC_FALSE	Busy 상태가 아닙니다.

➲ Example

➲ See Also

```
WIMC_Set_BusySize();
WIMC_Get_BusySize();
WIMC_Is_Busy();          // 현재 Busy상태를 확인
```

20.1.2. WIMC_Is_OnMotion

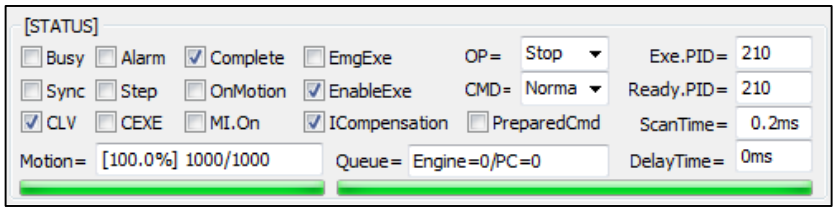
➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_OnMotion( void);
```

➲ Description

모션엔진이 모션동작 중임을 확인합니다.

<모션마스터프로그램내의 엔진 주요상태값확인>



➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	모션동작중입니다.
WIMC_FALSE	모션동작중이 아닙니다.

➲ Example

➲ See Also

20.1.3. WIMC_Is_CompleteAxis

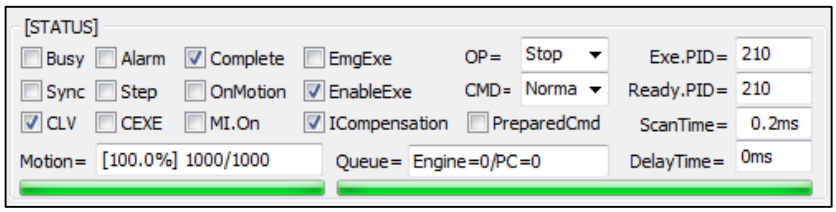
➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_CompleteAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

모션엔진의 특정축 모션명령 완료상태를 확인합니다.

<모션마스터프로그램내의 엔진 주요상태값확인>



➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	모션명령 완료 상태입니다.
WIMC_FALSE	모션명령 완료 상태가 아닙니다.

➲ Example

➲ See Also

20.1.4. WIMC_Is_Complete

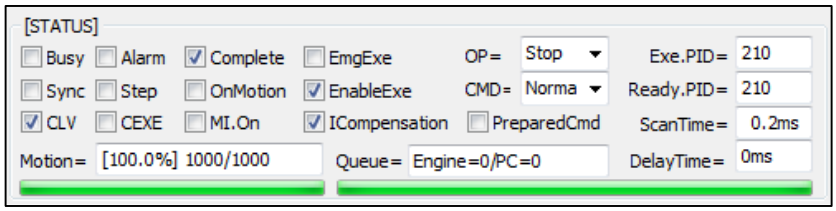
➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_Complete( void);
```

➲ Description

모션엔진의 모션명령 완료상태를 확인합니다.

<모션마스터프로그램내의 엔진 주요상태값확인>



➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	모션명령 완료 상태입니다.
WIMC_FALSE	모션명령 완료 상태가 아닙니다.

➲ Example

➲ See Also

20.1.5. WIMC_Get_VelState

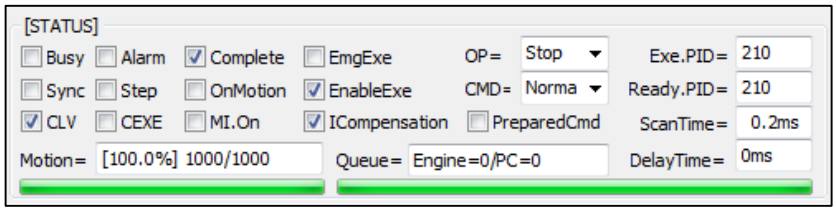
➲ Syntax

```
WIMC_UINT8    WIMC_API                WIMC_Get_VelState( void);
```

➲ Description

모션엔진의 현재 보간동작상태를 확인합니다. (보간명령만을 대상으로 합니다.)

<모션마스터프로그램내의 엔진 주요상태값확인>



OP=Stop / Acc / Steady / Dec

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_VELSTATE_STOP	정지상태
WIMC_VELSTATE_ACC	가속상태
WIMC_VELSTATE_STEADY	정속유지상태
WIMC_VELSTATE_DEC	감속상태

➲ Example

➲ See Also

20.1.6. WIMC_Get_VelStateAxis

➲ Syntax

```
WIMC_UINT8    WIMC_API                WIMC_Get_VelStateAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

모션엔진의 특정축 보간동작상태를 확인합니다. (보간명령만을 대상으로 합니다.)

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_VELSTATE_STOP	정지상태
WIMC_VELSTATE_ACC	가속상태
WIMC_VELSTATE_STEADY	정속유지상태
WIMC_VELSTATE_DEC	감속상태

➲ Example

➲ See Also

21. Axis Pulse

21.1. 축이동 펄스신호 기능설정

WIMC 엔진은 아래의 3가지 방식의 펄스신호를 발생할 수 있습니다.

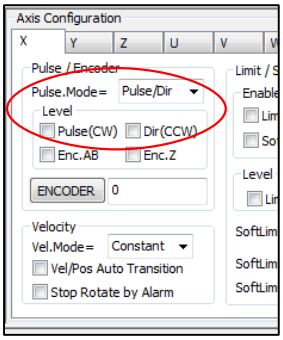
(1) 1 PULSE 방식
PULSE - DIR방식으로 불립니다. 축을 움직이는 2개의 신호 중 하나는 이동펄스신호이고 다른 하나는 이동방향신호입니다.

(2) 2 PULSE 방식
CW-CCW PULSE 방식으로 불립니다. 축을 움직이는 2개의 신호 중 하나는 CW회전방향의 이동펄스신호이고 다른 하나는 CCW회전방향의 이동펄스신호입니다.

(3) DISABLE
엔진내부의 모든 연산동작은 수행하지만 그 결과인 펄스를 출력하지 않는 기능입니다. 모션동작의 디버깅 및 시뮬레이션 기능으로 사용될 수 있습니다.

각 축이동 펄스신호들의 활성화 레벨의 반전이 가능합니다.

<모션마스터프로그램내의 설정>



<관련 라이브러리함수>

```
// 펄스출력모드 설정 및 설정값 읽기
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_PULSEDIR);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_DISABLE);
WIMC_Get_PulseMode(WIMC_X);

// 펄스출력레벨 설정 및 설정값 읽기
WIMC_Set_PulseLevel(WIMC_XYZ, WIMC_LEVEL_H);
WIMC_Get_PulseLevel(WIMC_X);
```

<리셋시 설정값>

1 PULSE 방식

H 활성화(액티브)레벨

21.1.1. WIMC_Set_PulseMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_PulseMode
(
              WIMC_UINT8              axis,
              WIMC_INT32              mode
);
```

➲ Description

특정축의 펄스모드를 설정합니다.

관련 내용 및 설명은 (21.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

mode

WIMC_PULSEMODE_DISABLE : Disable
WIMC_PULSEMODE_PULSEDIR : 1 Pulse 방식입니다.
WIMC_PULSEMODE_CWCCW : 2 Pulse 방식입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
// 펄스출력모드 설정 및 설정값 읽기
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_PULSEDIR);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_DISABLE);
WIMC_Get_PulseMode(WIMC_X);

// 펄스출력레벨 설정 및 설정값 읽기
WIMC_Set_PulseLevel(WIMC_XYZ, WIMC_LEVEL_H);
WIMC_Get_PulseLevel(WIMC_X);
```

21.1.2. WIMC_Get_PulseMode

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_PulseMode
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 펄스모드를 확인합니다.

관련 내용 및 설명은 (21.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

펄스모드를 반환합니다.

Return code/value	Description
WIMC_PULSEMODE_DISABLE	엔진내부의 모든 연산동작은 수행하지만 그 결과인 펄스를 출력하지 않는 기능입니다. 모션동작의 디버깅 및 시뮬레이션 기능으로 사용될 수 있습니다.
WIMC_PULSEMODE_PULSEDIR	PULSE - DIR방식으로 불립니다. 축을 움직이는 2개의 신호 중 하나는 이동펄스신호이고 다른 하나는 이동방향신호입니다.
WIMC_PULSEMODE_CWCCW	CW-CCW PULSE 방식으로 불립니다. 축을 움직이는 2개의 신호 중 하나는 CW회전방향의 이동펄스신호이고 다른 하나는 CCW회전방향의 이동펄스신호입니다.

➲ Example

➲ See Also

```
// 펄스출력모드 설정 및 설정값 읽기
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_PULSEDIR);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_DISABLE);
WIMC_Get_PulseMode(WIMC_X);

// 펄스출력레벨 설정 및 설정값 읽기
WIMC_Set_PulseLevel(WIMC_XYZ, WIMC_LEVEL_H);
WIMC_Get_PulseLevel(WIMC_X);
```

21.1.3. WIMC_Set_PulseLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_PulseLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축 펄스신호들의 활성화 레벨을 설정합니다.

관련 내용 및 설명은 (21.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : 신호레벨 High로 설정합니다.

WIMC_LEVEL_L : 신호레벨 Low로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
// 펄스출력모드 설정 및 설정값 읽기
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_PULSEDIR);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_DISABLE);
WIMC_Get_PulseMode(WIMC_X);

// 펄스출력레벨 설정 및 설정값 읽기
WIMC_Set_PulseLevel(WIMC_XYZ, WIMC_LEVEL_H);
WIMC_Get_PulseLevel(WIMC_X);
```

21.1.4. WIMC_Get_PulseLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Get_PulseLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축 펄스신호들의 활성화 레벨 설정값을 읽습니다.

관련 내용 및 설명은 (21.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

펄스출력 활성화 레벨을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

```
// 펄스출력모드 설정 및 설정값 읽기
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_PULSEDIR);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_DISABLE);
WIMC_Get_PulseMode(WIMC_X);

// 펄스출력레벨 설정 및 설정값 읽기
WIMC_Set_PulseLevel(WIMC_XYZ, WIMC_LEVEL_H);
WIMC_Get_PulseLevel(WIMC_X);
```

21.1.5. WIMC_Set_DirLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_DirLevel
(
              WIMC_UINT8              axis,
              WIMC_INT32              bLevel
);
```

➲ Description

1펄스 모드 지정축의 모터회전 방향 신호를 반전 출력 설정 합니다.

```
WIMC_Set_DirLevel( WIMC_X, TRUE);  //1펄스 모드 X축의 회전방향을 반전 출력 설정합니다.
```

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_TRUE : 모터 회전방향 신호 출력을 반전합니다.

WIMC_FALSE : 기본값

➲ Return Value

Return code/value	Description

➲ Example

```
// 펄스모드
WIMC_Set_PulseMode( WIMC_X, WIMC_PULSEMODE_PULSEDIR );

// 펄스 레벨
WIMC_Set_PulseLevel(WIMC_X, WIMC_TRUE );
WIMC_Set_DirLevel(WIMC_X, WIMC_TRUE );
```

➲ See Also

21.1.6. WIMC_Get_DirLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Get_DirLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

모터회전 방향 신호 레벨 설정값을 확인합니다.(1펄스 모드 지정축)

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

레벨 설정값을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

```
// 펄스모드
WIMC_Set_PulseMode( WIMC_X, WIMC_PULSEMODE_PULSEDIR );

// 펄스 레벨
WIMC_Set_PulseLevel(WIMC_X, WIMC_TRUE );
WIMC_Set_DirLevel(WIMC_X, WIMC_TRUE );
```

21.1.7. WIMC_Set_CwLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_CwLevel
(
              WIMC_UINT8              axis,
              WIMC_INT32              bLevel
);
```

➲ Description

2펄스 모드 지정축의 CW 펄스 신호를 반전 출력 설정 합니다.

WIMC_Set_CwLevel(WIMC_X, TRUE); //2펄스 모드 X축의 CW 펄스 신호를 반전 출력 설정합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_TRUE : 신호 출력을 반전 설정합니다.

WIMC_FALSE : 기본값

➲ Return Value

Return code/value	Description

➲ Example

```
// 펄스모드(X,Y,Z 축을 CWCCW 모드로 설정
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);

// 펄스 출력 레벨
WIMC_Set_CwLevel(WIMC_XYZ, WIMC_TRUE );
WIMC_Set_CcwLevel(WIMC_XYZ, WIMC_TRUE );
```

➲ See Also

21.1.8. WIMC_Get_CwLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Get_CwLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

CW 펄스 신호 레벨 설정값을 확인합니다.(2펄스 모드 지정축)

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

레벨 설정값을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

```
// 펄스모드(X,Y,Z 축을 CWCCW 모드로 설정
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);

// 펄스 출력 레벨
WIMC_Set_CwLevel(WIMC_XYZ, WIMC_TRUE );
WIMC_Set_CcwLevel(WIMC_XYZ, WIMC_TRUE );
```

21.1.9. WIMC_Set_CcwLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_CcwLevel
(
              WIMC_UINT8              axis,
              WIMC_INT32              bLevel
);
```

➲ Description

2펄스 모드 지정축의 CCW 펄스 신호를 반전 출력 설정 합니다.

WIMC_Set_CcwLevel(WIMC_X, TRUE); //2펄스 모드 X축의 CCW 펄스 신호를 반전 출력 설정합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_TRUE : 신호 출력을 반전 설정합니다.

WIMC_FALSE : 기본값

➲ Return Value

Return code/value	Description

➲ Example

```
// 펄스모드(X,Y,Z 축을 CWCCW 모드로 설정
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);

// 펄스 출력 레벨
WIMC_Set_CwLevel(WIMC_XYZ, WIMC_TRUE );
WIMC_Set_CcwLevel(WIMC_XYZ, WIMC_TRUE );
```

➲ See Also

21.1.10. WIMC_Get_CcwLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Get_CcwLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

CCW 펄스 신호 레벨 설정값을 확인합니다.(2펄스 모드 지정축)

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

레벨 설정값을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

```
// 펄스모드(X,Y,Z 축을 CWCCW 모드로 설정
WIMC_Set_PulseMode(WIMC_XYZ,  WIMC_PULSEMODE_CWCCW);

// 펄스 출력 레벨
WIMC_Set_CwLevel(WIMC_XYZ, WIMC_TRUE );
WIMC_Set_CcwLevel(WIMC_XYZ, WIMC_TRUE );
```

21.1.11. WIMC_Is_CW

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_CW
(
              WIMC_UINT8              axis
);
```

➲ Description

모터가 시계방향(CW) 동작중인지 확인 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한리미트 신호를 반환합니다.

Return code/value	Description
WIMC_TRUE(1)	시계방향(CW) 회전 상태
WIMC_FALSE(0)	

➲ Example

➲ See Also

21.1.12. WIMC_Is_CCW

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_CCW
(
              WIMC_UINT8                axis
);
```

➲ Description

모터가 반 시계방향(CCW) 동작중인지 확인 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한리미트 신호를 반환합니다.

Return code/value	Description
WIMC_TRUE(1)	반 시계방향(CCW) 회전 상태
WIMC_FALSE(0)	

➲ Example

➲ See Also

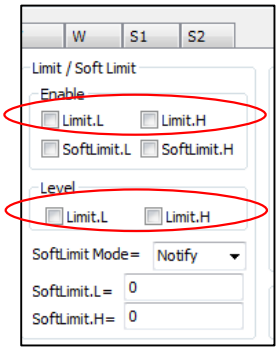
22. Limit

22.1. 리미트신호 기능설정

리미트신호는 각축의 상한 이동금지 및 하한 이동금지를 위한 신호입니다. 일반적으로 상한리미트신호와 하한리미트 신호로 구성되지만 WIMC 엔진은 개별 축마다 단 하나의 LIMIT신호만을 제공합니다. 하나의 리미트신호로써 하한리미트상태인 LIMIT.L와 상한리미트상태인 LIMIT.H신호 모두를 활성화/비활성화 하게 됩니다.
리미트 신호가 활성화 되면 알람이 발생하고 비상실행상태가 됩니다. 리미트 신호에 의한 알람상태가 발생하면 리미트신호의 발생원인을 제거하거나 리미트 신호를 비활성화 상태로 만들어야만 알람리셋명령에 의해 알람이 해제됩니다.

WIMC 엔진은 LIMIT신호기능을 비활성화(Disable)하면 일반적인 범용 입력신호로도 사용할 수 있습니다.

<모션마스터프로그램내의 설정>



<관련 라이브러리 함수>

```
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
```

<리셋시 설정값>

기능OFF

활성화 레벨 H

22.1.1. WIMC_Enable_LowerLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_LowerLimit
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 하한 LIMIT 신호 기능(리미트 알람)을 Enable/Disable 합니다.
Enable 설정 시 LIMIT 신호가 들어오면 모션보드는 알람상태가 됩니다.

소프트웨어적으로는 상한/하한 리미트로 나누어져 있지만, 보션보드 하드웨어는 1개의 입력 신호로 할당 되어 있습니다.
따라서, 보션보드 리미트 알람 시 상한/하한 구분 없이 리미트 알람이 발생됩니다.

관련 내용 및 설명은 (22.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable.
WIMC_FALSE : Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

22.1.2. WIMC_IsEnable_LowerLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_IsEnable_LowerLimit
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 하한리미트 알람 설정 상태를 읽습니다.

Enable 설정 시 LIMIT 신호가 들어오면 모션보드는 알람상태가 됩니다.
소프트웨어적으로는 상한/하한 리미트로 나누어져 있지만, 보션보드 하드웨어는 1개의 신호로 할당 되어 있습니다.
따라서, 보션보드 리미트 알람 시 상한/하한 구분 없이 리미트 알람이 발생됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한리미트 신호 알람 설정 상태를 반환합니다.

Return code/value	Description
WIMC_TRUE(1)	리미트 알람 설정 상태
WIMC_FALSE(0)	리미트 알람 해제 상태

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

22.1.3. WIMC_Enable_UpperLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_UpperLimit
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 상한 LIMIT 신호기능(리미트 알람)을 Enable/Disable 합니다.
Enable 설정 시 LIMIT 신호가 들어오면 모션보드는 알람상태가 됩니다.
소프트웨어적으로는 상한/하한 리미트로 나누어져 있지만, 보션보드 하드웨어는 1개의 입력 신호로 할당 되어 있습니다.
따라서, 보션보드 리미트 알람 시 상한/하한 구분 없이 리미트 알람이 발생됩니다.

관련 내용 및 설명은 (22.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable.
WIMC_FALSE : Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

22.1.4. WIMC_IsEnable_UpperLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_IsEnable_UpperLimit
(
              WIMC_UINT8                axis
);
```

➲ Description

특정축의 상한리미트 알람 설정 상태를 읽습니다.

Enable 설정 시 LIMIT 신호가 들어오면 모션보드는 알람상태가 됩니다.
소프트웨어적으로는 상한/하한 리미트로 나누어져 있지만, 보션보드 하드웨어는 1개의 신호로 할당 되어 있습니다.
따라서, 보션보드 리미트 알람 시 상한/하한 구분 없이 리미트 알람이 발생됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

상한리미트 신호 알람 설정 상태를 반환합니다.

Return code/value	Description
WIMC_TRUE(1)	리미트 알람 설정 상태
WIMC_FALSE(0)	리미트 알람 해제 상태

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

22.1.5. WIMC_Set_LowerLimitLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_LowerLimitLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 하한리미트 활성화 신호레벨을 설정합니다.

관련 내용 및 설명은 (22.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : 신호레벨 High로 설정합니다.

WIMC_LEVEL_L : 신호레벨 Low로 설정합니다.

➲ Return Value

하한리미트 신호레벨을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
```

WIMC_Get_LowerLimitLevel(WIMC_X);

22.1.6. WIMC_Get_LowerLimitLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Get_LowerLimitLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 하한리미트 활성화 신호레벨을 읽습니다.

관련 내용 및 설명은 (22.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한리미트 신호레벨을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

22.1.7. WIMC_Set_UpperLimitLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_UpperLimitLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 상한리미트 활성화 신호레벨을 설정합니다.

관련 내용 및 설명은 (22.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : 신호레벨 High로 설정합니다.

WIMC_LEVEL_L : 신호레벨 Low로 설정합니다.

➲ Return Value

상한리미트 신호레벨을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H	High Level
WIMC_LEVEL_L	Low Level

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
```

WIMC_Get_LowerLimitLevel(WIMC_X);

22.1.8. WIMC_Get_UpperLimitLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Get_UpperLimitLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 상한리미트 활성화 신호레벨을 읽습니다.

관련 내용 및 설명은 (22.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

상한리미트 신호레벨을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

22.1.9. WIMC_Is_LimitL

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_LimitL
(
              WIMC_UINT8                axis
);
```

➲ Description

하한리미트 신호를 읽습니다.

소프트웨어적으로는 상한/하한 리미트로 나누어져 있지만, 보션보드 하드웨어는 1개의 입력 신호로 할당 되어 있습니다.
따라서, 보션보드 리미트 신호는 상한/하한 구분 없이 동시 작동합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한리미트 신호를 반환합니다.

Return code/value	Description
WIMC_TRUE(1)	리미트 신호 입력 상태
WIMC_FALSE(0)	신호 없음

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

22.1.10. WIMC_Is_LimitH

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_LimitH
(
              WIMC_UINT8              axis
);
```

➲ Description

상한리미트 신호를 읽습니다.

소프트웨어적으로는 상한/하한 리미트로 나누어져 있지만, 보션보드 하드웨어는 1개의 입력 신호로 할당 되어 있습니다.
따라서, 보션보드 리미트 신호는 상한/하한 구분 없이 동시 작동합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한리미트 신호를 반환합니다.

Return code/value	Description
WIMC_TRUE(1)	리미트 신호 입력 상태
WIMC_FALSE(0)	신호 없음

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

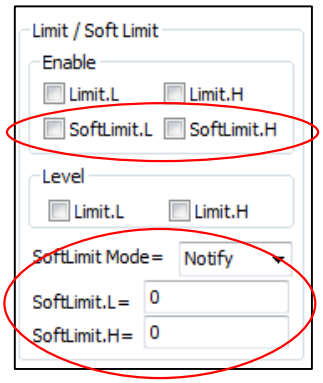
23. Soft Limit

23.1. 소프트 리미트 기능 설정

LIMIT신호에 의한 움직임제한 기능을 하드리미트(Hard Limit)기능이라고 부른다면, 소프트 리미트 기능은 모션엔진의 현재 위치카운터값의 하한, 상한 경계값을 두어 움직임을 제한하는 기능을 말합니다.
소프트리미트기능은 일반적으로 원점복귀에 의해서 모션시스템이 기계좌표계와 올바르게 연결된 이후에 사용해야 안전합니다.
(반면에 하드리미트기능은 모션시스템이 기동된 후 언제든지 동작됩니다).
소규모의 모션시스템은 배선구조를 간략화 하기 위해 원점복귀 기능을 위한 홈센서와 소프트리미트 기능만을 사용하기도 합니다.
소프트리미트 기능은 활성화(Enable) , 비활성화(Disable)를 선택할 수 있습니다.

WIMC 엔진은 현재 위치가 SoftLowerLimit(하한경계값)보다 작거나 SoftUpperLimit(상한경계값)보다 클 때 소프트리미트 상태가 발생됩니다. 소프트리미트 상태가 발생되면 아래의 3가지 중 하나의 동작을 취할 수 있습니다.
(1) Notify : 통지
동작을 정지시키지 않으며 알람신호만을 발생시킵니다.
(2) DecStop : 감속정지
알람신호와 함께 현재 속도모드와 가감속도계수로 부드럽게 감속정지시키며, 비상실행상태로 전환됩니다.
(3) EmgStop : 비상정지
알람신호와 함께 비상정지방식(급정지 혹은 급감속정지)으로 정지시키며, 비상실행상태로 전환됩니다.

<모션마스터프로그램내의 설정>



<관련 라이브러리 함수>

```
WIMC_Set_SoftLowerLimit (WIMC_X, 0);
WIMC_Get_SoftLowerLimit (WIMC_X);
WIMC_Set_SoftUpperLimit (WIMC_X, 1234);
WIMC_Get_SoftUpperLimit (WIMC_X);
WIMC_Set_SoftLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_NOTIFY);
WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_DECSTOP);
WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_EMGSTOP);
WIMC_Get_SoftLimitMode();
```

<리셋시 설정값>

기능 OFF

하한값 = 0

상한값 = 0

동작모드 = Notify

23.1.1. WIMC_Set_SoftLimitMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_SoftLimitMode
(
              WIMC_UINT8              axis,
              WIMC_INT32              mode
);
```

➲ Description

특정축의 소프트리미트 모드를 설정합니다.

관련 내용 및 설명은 (23.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

mode

WIMC_SOFTLIMITMODE_DISABLE :소프트 리미트 모드 비활성화 상태입니다.
WIMC_SOFTLIMITMODE_NOTIFY :통지 모드입니다.
WIMC_SOFTLIMITMODE_DECSTOP :감속정지 모드입니다.
WIMC_SOFTLIMITMODE_IMSTOP :비상정지 모드입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_SoftUpperLimit(WIMC_X, 1234);
WIMC_Get_SoftUpperLimit(WIMC_X);
WIMC_Set_SoftLowerLimit(WIMC_X, 1234);
WIMC_Get_SoftLowerLimit(WIMC_X);
WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_DISABLE);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_NOTIFY);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_DECSTOP);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_EMGSTOP);
WIMC_Get_SoftLimitMode(WIMC_X);
```

23.1.2. WIMC_Get_SoftLimitMode

➲ Syntax

```
WIMC_INT32      WIMC_API                WIMC_Get_SoftLimitMode
(
                WIMC_UINT8              axis
);
```

➲ Description

특정축의 소프트리미트 모드를 읽습니다.

관련 내용 및 설명은 (23.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

소프트 리미트 모드를 반환합니다. 아래 값들을 반환합니다.

Return code/value	Description
WIMC_SOFTLIMITMODE_DISABLE	소프트 리미트 모드 비활성화 상태입니다.
WIMC_SOFTLIMITMODE_NOTIFY	통지 모드입니다.
WIMC_SOFTLIMITMODE_DECSTOP	감속정지 모드입니다.
WIMC_SOFTLIMITMODE_IMSTOP	비상정지 모드입니다.

➲ Example

➲ See Also

```
WIMC_Set_SoftUpperLimit(WIMC_X, 1234);
WIMC_Get_SoftUpperLimit(WIMC_X);
WIMC_Set_SoftLowerLimit(WIMC_X, 1234);
WIMC_Get_SoftLowerLimit(WIMC_X);WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_DISABLE);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_NOTIFY);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_DECSTOP);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_EMGSTOP);
WIMC_Get_SoftLimitMode(WIMC_X);
```

23.1.3. WIMC_Set_SoftLowerLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_SoftLowerLimit
(
              WIMC_UINT8              axis,
              WIMC_INT32              pulse
);
```

➲ Description

특정축의 하한 소프트리미트 위치(펄스단위) 값을 설정합니다.

관련 내용 및 설명은 (23.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulse

하한 소프트 리미트 펄스 값입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_SoftUpperLimit(WIMC_X, 1234);
WIMC_Get_SoftUpperLimit(WIMC_X);
WIMC_Set_SoftLowerLimit(WIMC_X, 1234);
WIMC_Get_SoftLowerLimit(WIMC_X);
WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_DISABLE);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_NOTIFY);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_DECSTOP);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_EMGSTOP);
WIMC_Get_SoftLimitMode(WIMC_X);
```

23.1.4. WIMC_Get_SoftLowerLimit

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_SoftLowerLimit
(
              WIMC_UINT8              axis
);
```

➲ Description

해당축의 하한 소프트리미트 위치(펄스) 값을 읽습니다.

WIMC_Set_SoftLowerLimit 함수를 이용하여 리미트 위치(펄스단위) 값을 설정 할 수 있습니다.

관련 내용 및 설명은 (23.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한 소프트 리미트 펄스 값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_SoftUpperLimit(WIMC_X, 1234);
WIMC_Get_SoftUpperLimit(WIMC_X);
WIMC_Set_SoftLowerLimit(WIMC_X, 1234);
WIMC_Get_SoftLowerLimit(WIMC_X);
WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_DISABLE);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_NOTIFY);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_DECSTOP);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_EMGSTOP);
WIMC_Get_SoftLimitMode(WIMC_X);
```

23.1.5. WIMC_Set_SoftUpperLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_SoftUpperLimit
(
              WIMC_UINT8              axis,
              WIMC_INT32              pulse
);
```

➲ Description

특정축의 상한 소프트리미트 위치(펄스단위) 값을 설정합니다.

관련 내용 및 설명은 (23.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulse

상한 소프트 리미트 펄스 값입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_SoftUpperLimit(WIMC_X, 1234);
WIMC_Get_SoftUpperLimit(WIMC_X);
WIMC_Set_SoftLowerLimit(WIMC_X, 1234);
WIMC_Get_SoftLowerLimit(WIMC_X);
WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_DISABLE);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_NOTIFY);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_DECSTOP);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_EMGSTOP);
WIMC_Get_SoftLimitMode(WIMC_X);
```

23.1.6. WIMC_Get_SoftUpperLimit

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_SoftUpperLimit
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 상한 소프트리미트 위치(펄스) 값을 읽습니다.

WIMC_Set_SoftUpperLimit 함수를 이용하여 리미트 위치(펄스단위) 값을 설정 할 수 있습니다.

관련 내용 및 설명은 (23.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

상한 소프트 리미트 펄스 값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_SoftUpperLimit(WIMC_X, 1234);
WIMC_Get_SoftUpperLimit(WIMC_X);
WIMC_Set_SoftLowerLimit(WIMC_X, 1234);
WIMC_Get_SoftLowerLimit(WIMC_X);
WIMC_Set_SoftLimitMode(WIMC_SOFTLIMITMODE_DISABLE);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_NOTIFY);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_DECSTOP);
WIMC_Set_SoftLimitMode(WIMC_X, WIMC_SOFTLIMITMODE_EMGSTOP);
WIMC_Get_SoftLimitMode(WIMC_X);
```

23.1.7. WIMC_Enable_SoftLowerLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_SoftLowerLimit
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 하한 소프트 LIMIT 신호 기능을 Enable/Disable 합니다.
Enable 설정 시 소프트리미트 위치 범위를 벗어나게 되면 모션보드는 알람상태가 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 동작 활성화
WIMC_FALSE : 동작 비활성화

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

23.1.8. WIMC_Enable_SoftUpperLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_SoftUpperLimit
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 상한 소프트 LIMIT 신호 기능을 Enable/Disable 합니다.
Enable 설정 시 소프트리미트 위치 범위를 벗어나게 되면 모션보드는 알람상태가 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 동작 활성화
WIMC_FALSE : 동작 비활성화

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

23.1.9. WIMC_IsEnable_SoftLowerLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_IsEnable_SoftLowerLimit
(
              WIMC_UINT8                axis
);
```

➲ Description

특정축의 하한소프트리미트 활성화 설정 상태를 읽습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE(1)	하한 소프트 리미트 활성화
WIMC_FALSE(0)	하한 소프트 리미트 비활성화

➲ Example

➲ See Also

```
WIMC_Enable_SoftLowerLimit(WIMC_X, TRUE);
WIMC_Enable_SoftUpperLimit(WIMC_X, TRUE);
```

23.1.10. WIMC_IsEnable_SoftUpperLimit

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_IsEnable_SoftUpperLimit
(
              WIMC_UINT8                axis
);
```

➲ Description

특정축의 상한 소프트리미트 활성화 설정 상태를 읽습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE(1)	상한 소프트 리미트 활성화
WIMC_FALSE(0)	상한 소프트 리미트 비활성화

➲ Example

➲ See Also

```
WIMC_Enable_SoftLowerLimit(WIMC_X, TRUE);
WIMC_Enable_SoftUpperLimit(WIMC_X, TRUE);
```

23.1.11. WIMC_Is_SoftLimitL

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_SoftLimitL
(
              WIMC_UINT8              axis
);
```

➲ Description

소프트 하한리미트 신호를 읽습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한리미트 신호를 반환합니다.

Return code/value	Description
WIMC_TRUE(1)	소프트 리미트 신호 입력 상태
WIMC_FALSE(0)	신호 없음

➲ Example

➲ See Also

```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

23.1.12. WIMC_Is_SoftLimitH

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_SoftLimitH
(
              WIMC_UINT8                axis
);
```

➲ Description

소프트 상한리미트 신호를 읽습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

하한리미트 신호를 반환합니다.

Return code/value	Description
WIMC_TRUE(1)	소프트 리미트 신호 입력 상태
WIMC_FALSE(0)	신호 없음

➲ Example

➲ See Also

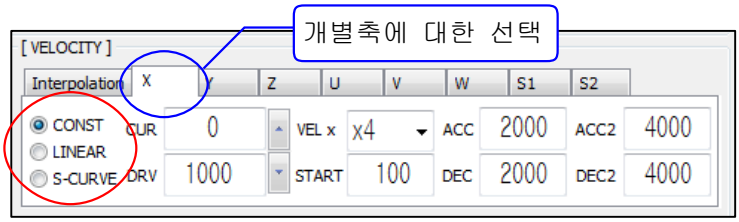
```
WIMC_Enable_UpperLimit(WIMC_X, TRUE);
WIMC_Enable_LowerLimit(WIMC_X, TRUE);
WIMC_Set_UpperLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Set_LowerLimitLevel(WIMC_X, WIMC_LEVEL_H);
WIMC_Get_UpperLimitLevel(WIMC_X);
WIMC_Get_LowerLimitLevel(WIMC_X);
```

24. 속도제어 설정

속도모드 설정

보간동작 속도모드와 동일하게 Contant Velocity, Linear Velocity, S-Curve Velocity 모드를 지원합니다.

<모션마스터프로그램내의 설정>



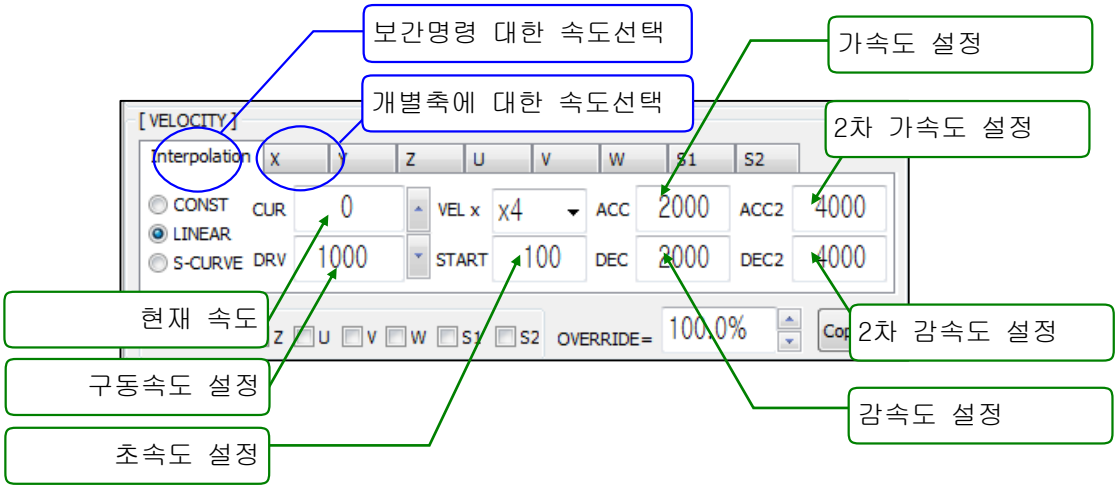
<관련 라이브러리 함수>

```
WIMC_Set_VelModeAxis(WIMC_X, WIMC_VELMODE_LINEAR);
```

개별축 현재속도, 구동속도, 초속도(시작속도), 가감속도 설정

모든 속도 및 가감속도는 속도배율값(Velocity Multiplier)을 곱해주어야 합니다.
S1, S2축과 같은 스핀들 기능을 포함하는 축의 스핀들 회전동작은 고정배율 x16이 사용됩니다.
현재 속도(Current Velocity)값은 특정값으로 설정할 수는 없으며 읽기만 가능합니다.

<모션마스터프로그램내의 설정>



<관련 라이브러리 함수>

```
WIMC_Set_DriveVel();
WIMC_Get_DriveVel();
WIMC_Get_CurVel();                  // 현재속도는 읽기만 가능합니다.

WIMC_Set_StartVel();
WIMC_Get_StartVel();
```

```
WIMC_Set_Acc();  WIMC_Get_Acc();
WIMC_Set_Dec();  WIMC_Get_Dec();
WIMC_Set_Acc2(); WIMC_Get_Acc2();
WIMC_Set_Dec2(); WIMC_Get_Dec2();
// 개별축의 경우 함수 이름끝에 Axis가 붙습니다.
// 예를 들면 WIMC_Set_DriveVel() -> WIMC_Set_DriveVelAxis() 로 표기함으로써 개별축에 대한
// 속도설정 명령이 됩니다.
```

<리셋시 설정값>

속도배율 = 4 , 구동속도 = 1000, 현재속도 = 0, 시작속도 = 100,

가속도 = 2000, 감속도 = 2000, 2차 가속도 = 4000, 2차 감속도 = 4000

24.1. 속도모드 설정

24.1.1. WIMC_Set_VelMode

➲ Syntax

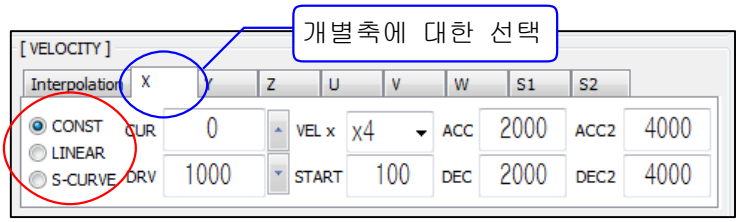
```
WIMC_BOOL     WIMC_API                WIMC_Set_VelMode
(
              WIMC_INT32              mode
);
```

➲ Description

보간동작용 속도모드를 설정합니다.

Constant Velocity, Linear Velocity, S-Curve Velocity 모드를 지원합니다.

<모션마스터프로그램내의 설정>



➲ Parameters

mode

WIMC_VELMODE_CONST : 일정속도 제어기능 (constant velocity mode)

WIMC_VELMODE_LINEAR : 1차 가감속에 의한 비대칭형 LINEAR속도제어 (Trapezoidal velocity mode)

WIMC_VELMODE_SCURVE : 2차 가감속에 의한 비대칭형 S-CURVE 속도제어 (S-Curve velocity mode)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.1.2. WIMC_Get_VelMode

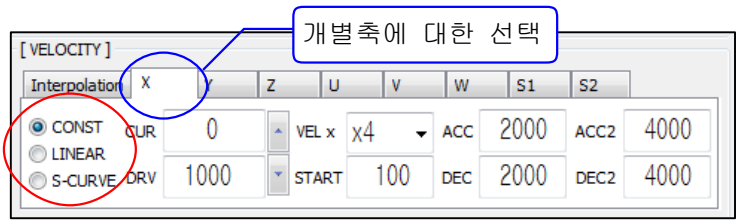
➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_VelMode( void);
```

➲ Description

보간동작용 속도모드를 확인합니다.

<모션마스터프로그램내의 설정>



➲ Parameters

➲ Return Value

속도모드를 반환합니다.

Return code/value	Description
WIMC_VELMODE_CONST	일정속도 제어기능 (constant velocity mode)
WIMC_VELMODE_LINEAR	1차 가감속에 의한 비대칭형 LINEAR속도제어 (Trapezoidal velocity mode)
WIMC_VELMODE_SCURVE	2차 가감속에 의한 비대칭형 S-CURVE 속도제어 (S-Curve velocity mode)

➲ Example

➲ See Also

24.1.3. WIMC_Set_VelModeAxis

➲ Syntax

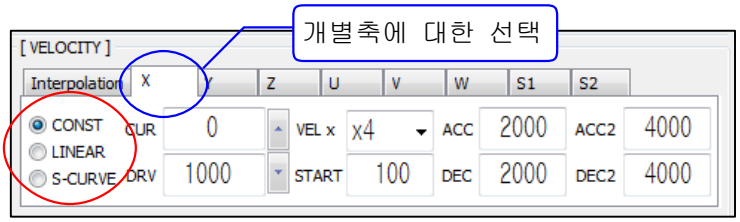
```
WIMC_BOOL     WIMC_API                WIMC_Set_VelModeAxis
(
              WIMC_UINT8              axis,
              WIMC_INT32              mode
);
```

➲ Description

특정축 동작용 속도모드를 설정합니다.

Constant Velocity, Linear Velocity, S-Curve Velocity 모드를 지원합니다.

<모션마스터프로그램내의 설정>



➲ Parameters

mode

WIMC_VELMODE_CONST : 일정속도 제어기능 (constant velocity mode)

WIMC_VELMODE_LINEAR : 1차 가감속에 의한 비대칭형 LINEAR속도제어 (Trapezoidal velocity mode)

WIMC_VELMODE_SCURVE : 2차 가감속에 의한 비대칭형 S-CURVE 속도제어 (S-Curve velocity mode)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.1.4. WIMC_Get_VelModeAxis

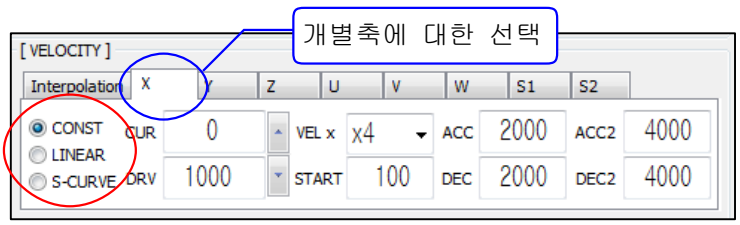
➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_VelModeAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축 동작용 속도모드를 확인합니다.

<모션마스터프로그램내의 설정>



➲ Parameters

➲ Return Value

특정축의 속도모드를 반환합니다.

Return code/value	Return code/value
WIMC_VELMODE_CONST	WIMC_VELMODE_CONST
WIMC_VELMODE_LINEAR	WIMC_VELMODE_LINEAR
WIMC_VELMODE_SCURVE	WIMC_VELMODE_SCURVE

➲ Example

➲ See Also

24.1.5. WIMC_Set_VelMode3StartMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_VelMode3StartMode
(
              WIMC_BOOL               mode
);
```

➲ Description

속도제어 연속이송방식 우선 옵션을 설정합니다.

➲ Parameters

mode

WIMC_TRUE : 연속이송방식 활성화

WIMC_FALSE : 연속이송방식 비활성화

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상 설정
WIMC_FALSE	

➲ Example

```
WIMC_Set_VelMode3StartMode(WIMC_TRUE);
```

➲ See Also

24.1.6. WIMC_Get_VelMode3StartMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Set_VelMode3StartMode()
```

➲ Description

속도제어 연속이송방식 우선 옵션을 설정 상태를 확인합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	연속이송방식 활성화 상태
WIMC_FALSE	연속이송방식 비활성화 상태

➲ Example

➲ See Also

```
WIMC_Set_VelMode3StartMode(WIMC_TRUE);
```

24.2. 현재속도

24.2.1. WIMC_Get_CurVel

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_CurVel( void);
```

➲ Description

보간동작명령용 현재속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

보간 동작명령용 현재속도 값을 반환합니다.
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.2.2. WIMC_Get_CurVelAxis

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_CurVelAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 현재속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

해당축의 현재속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.3. 구동속도

24.3.1. WIMC_Set_DriveVel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_DriveVel
(
              WIMC_INT32              pps
);
```

➲ Description

보간동작용 구동속도 설정명령입니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

pps
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.3.2. WIMC_Get_DriveVel

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_DriveVel( void);
```

➲ Description

보간동작명령용 구동속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

보간 동작명령용 구동속도값을 반환합니다.
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

WIMC_Set_DriveVel(); WIMC_Get_DriveVel();
WIMC_Get_CurVel(); // 현재속도는 읽기만 가능합니다.
WIMC_Set_StartVel(); WIMC_Get_StartVel();
WIMC_Set_Acc(); WIMC_Get_Acc();
WIMC_Set_Dec(); WIMC_Get_Dec();
WIMC_Set_Acc2(); WIMC_Get_Acc2();
WIMC_Set_Dec2(); WIMC_Get_Dec2();

24.3.3. WIMC_Set_DriveVelAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_DriveVelAxis
(
              WIMC_UINT8              axis,
              WIMC_INT32              pps
);
```

➲ Description

특정축의 구동속도를 설정합니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pps

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.3.4. WIMC_Get_DriveVelAxis

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_DriveVelAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 구동속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

해당축의 구동속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.4. 초속도

24.4.1. WIMC_Set_StartVel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_StartVel
(
              WIMC_INT32              pps
);
```

➲ Description

시작속도 설정명령입니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

pps
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.4.2. WIMC_Get_StartVel

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_StartVel( void);
```

➲ Description

시작속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

보간동작용 시작속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.4.3. WIMC_Set_StartVelAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_StartVelAxis
(
              WIMC_UINT8              axis,
              WIMC_INT32              pps
);
```

➲ Description

특정축의 시작속도를 설정합니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pps

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.4.4. WIMC_Get_StartVelAxis

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_StartVelAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 시작속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

해당축의 시작속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.5. 1 차 가속도

24.5.1. WIMC_Set_Acc

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_Acc
(
              WIMC_INT32              pps
);
```

➲ Description

보간동작용 1차 가속도 설정명령입니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

pps
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.5.2. WIMC_Get_Acc

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_Acc( void);
```

➲ Description

보간동작명령용 1차 가속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

보간 동작명령용 가속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.5.3. WIMC_Set_AccAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_AccAxis
(
              WIMC_UINT8              axis,
              WIMC_INT32              pps
);
```

➲ Description

특정축의 1차 가속도를 설정합니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pps

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.5.4. WIMC_Get_AccAxis

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_AccAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 1차 가속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

해당축의 가속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.6. 1 차 감속도

24.6.1. WIMC_Set_Dec

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_Dec
(
              WIMC_INT32              pps
);
```

➲ Description

보간동작용 1차 감속도 설정명령입니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

pps
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.6.2. WIMC_Get_Dec

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_Dec( void);
```

➲ Description

보간동작명령용 1차 감속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

보간 동작명령용 감속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.6.3. WIMC_Set_DecAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_DecAxis
(
              WIMC_UINT8              axis,
              WIMC_INT32              pps
);
```

➲ Description

특정축의 1차 감속도를 설정합니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pps

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.6.4. WIMC_Get_DecAxis

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_DecAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 감속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

해당축의 감속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.7. 2 차 가속도

24.7.1. WIMC_Set_Acc2

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_Acc2
(
              WIMC_INT32              pps
);
```

➲ Description

보간동작용 2차 가속도 설정명령입니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

pps
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.7.2. WIMC_Get_Acc2

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_Acc2( void);
```

➲ Description

보간동작명령용 2차 가속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

보간 동작명령용 2차 가속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.7.3. WIMC_Set_Acc2Axis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_Acc2Axis
(
              WIMC_UINT8              axis,
              WIMC_INT32              pps
);
```

➲ Description

특정축의 2차 가속도를 설정합니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pps

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.7.4. WIMC_Get_Acc2Axis

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_Acc2Axis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 2차 가속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

해당축의 2차 가속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.8. 2 차 감속도

24.8.1. WIMC_Set_Dec2

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_Dec2
(
              WIMC_INT32              pps
);
```

➲ Description

보간동작(합성축동작)용 2차 감속도 설정명령입니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

pps

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.8.2. WIMC_Get_Dec2

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_Dec2( void);
```

➲ Description

보간동작명령용 2차 감속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

보간 동작명령용 2차 감속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

24.8.3. WIMC_Set_Dec2Axis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_Dec2Axis
(
              WIMC_UINT8              axis,
              WIMC_INT32              pps
);
```

➲ Description

특정축의 2차 감속도를 설정합니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pps

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

24.8.4. WIMC_Get_Dec2Axis

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_Dec2Axis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 2차 감속도 값을 읽습니다.

관련 내용 및 설명은 (24절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

해당축의 2차 감속도 값을 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

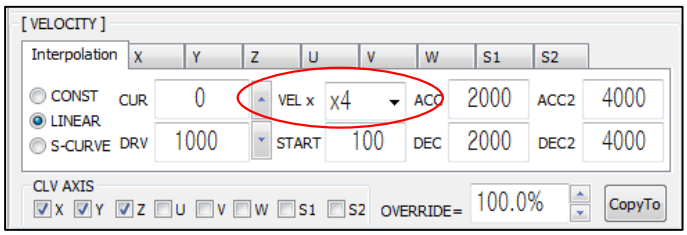
25. 속도 동작 옵션설정

25.1. 속도 배율수

모든 속도파라메터값은 속도배율기능에 의해 특정 배율값이 곱해진 값으로 동작됩니다. 속도배율값은 모션제어의 초기단계에서 전체 속도영역을 설정하는 값입니다. 속도파라메터의 데이터형이 부호없는 16비트로써 0~65535까지만 지정 가능합니다. 만약 속도배율이 없다면 최대 65.535kPPS까지 지정 가능하므로 수치상 부족함이 있습니다. 속도배율기능은 전체적인 속도설정파라메터값에 특정배율을 주어 설정영역을 크게 만드는 역할을 합니다. 또한 속도배율기능은 엔진의 내부처리속도를 향상시키기도 합니다.
속도배율값은 2의 누승값인 1, 2, 4, 8, 16, 32 로 지정됩니다. 이보다 큰 값을 지정할 수 있지만 속도의 분해해상도가 낮아지게 됩니다.
※속도 분해능 16비트는 전체 속도영역을 대략 65,000으로 분해합니다. 대부분 응용에서 65,000의 속도분해능으로 충분할 수 있습니다. 예를 들면 초속 10M로 이동하는 기기의 1/50,000 속도 분해능은 0.2mm/s 의 분해능 됩니다.

전체속도배율값의 선정할 때에는 대체로 구동최대속도값, 1차 가감속도값, 2차 가감속도값 들 중에서 가장 큰 값에 의해 결정하는 것이 바람직합니다. S자 가감속을 사용할 경우 일반적으로 2차가속도값이 가장 큽니다. 예를 들어 2차 가속도값이 400,000 이다면 속도배율을 8배율로 결정하여야 충분합니다 (60,000*8=480,000). 그러므로 전체속도배율값은 8이 되고 2차 가속도값은 50,000으로 지령하면 400,000 이 됩니다.

<모션마스터프로그램내의 속도배율 설정>



<관련 라이브러리 함수>

```
WIMC_Set_VelMultiplier(4); // x4 배율값
```

<리셋시 설정값>

배율값 = x4

25.1.1. WIMC_Set_VelMultiplier

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_VelMultiplier
(
              WIMC_INT32              multiplier
);
```

➲ Description

속도배율을 설정합니다.

관련 내용 및 설명은 (25.1절)을 참고 하실수 있습니다.

➲ Parameters

multiplier

속도배율값입니다.

속도배율값은 1, 2, 4, 8, 16, 32 입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_VelMultiplier(4);
WIMC_Get_VelMultiplier();
```

25.1.2. WIMC_Get_VelMultiplier

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_VelMultiplier( void);
```

➲ Description

속도배율을 읽습니다.

관련 내용 및 설명은 (25.1절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

속도배율값을 반환합니다.

속도배율값은 1, 2, 4, 8, 16, 32 입니다.

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_VelMultiplier(4);
WIMC_Get_VelMultiplier();
```

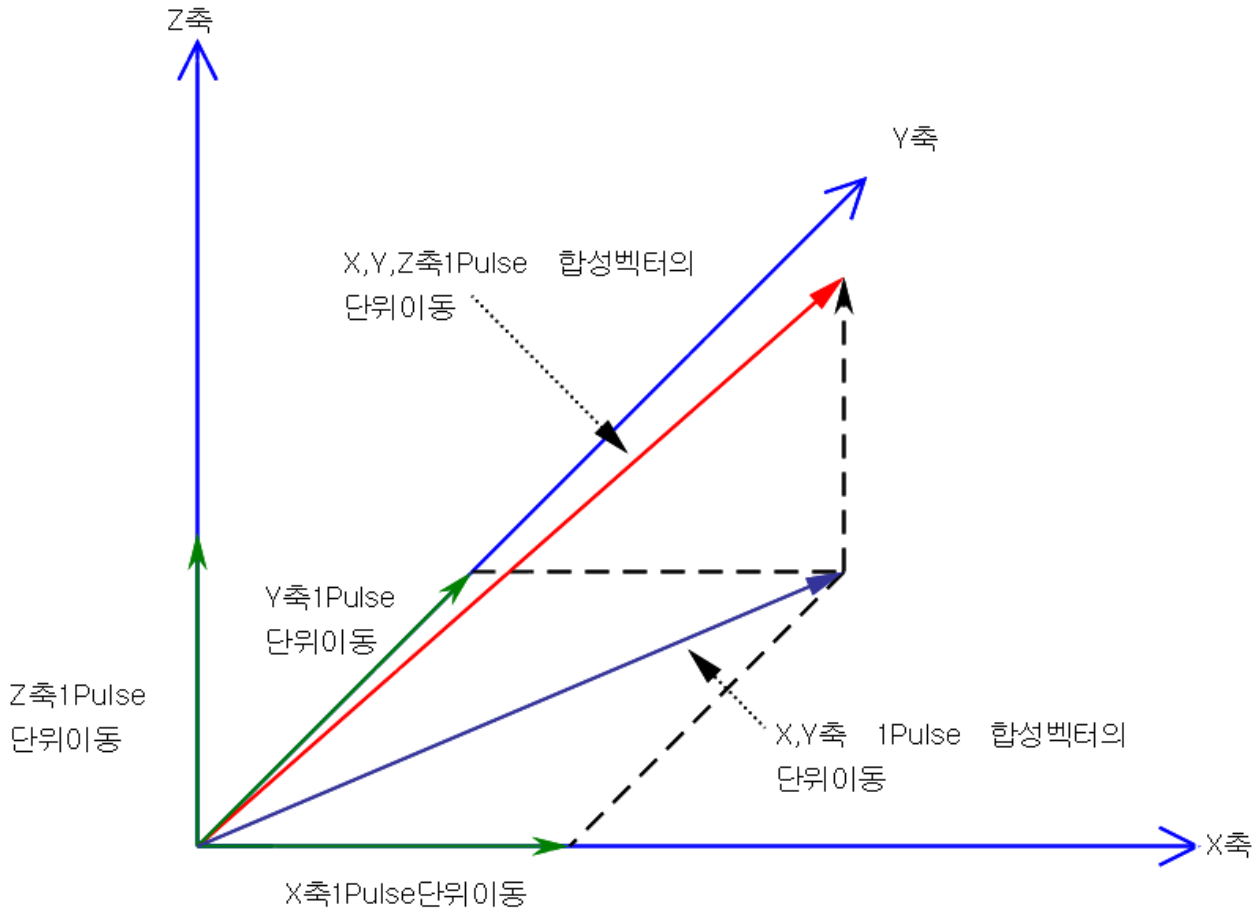
25.2. 선속도 일정기능

선속도(The Linear velocity of the curve)란 각 직교축의 속도항이 합성된 이동경로속도입니다..
선속도 일정기능(CLV : Constant Linear Velocity)은 직교축에 대해서 아래와 같은 동작을 실행함으로 써 구현됩니다.

▶ 1축만 1Pulse 이동할 때 선속도와 축속도는 동일합니다.
▶ 2축이 동시에 1Pulse 이동하면 선속도는 축속도의 1/√2 배입니다.
▶ 3축이 동시에 1Pulse 이동하면 선속도는 축속도의 1/√3 배입니다.

WIMC 엔진은 3차원까지만 선속도 일정제어를 수행합니다. 또한 선속도 일정제어를 포함하는 임의 축을 선택할 수 있습니다. 이것은 직선 X,Y,Z 축과 U축을 회전인덱스로 조합해서 사용할 때 선속도일정제어를 X,Y,Z축에만 적용하는 기능을 말합니다. 선속도일정축을 3개 이상 설정하더라도 X,Y,Z,U,V,W,S1,S2 순서로 먼저 설정된 3개의 축까지만 적용합니다.

아래 그림은 X,Y,Z축에 벡터 합성을 나타냅니다.

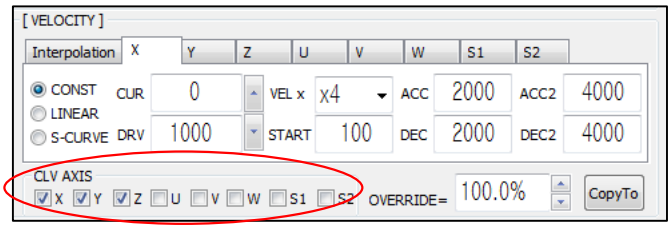


`

> 우선 모든 축이 1펄스에 의해 이동하는 거리인 [축단위이동거리]를 1로 가정합니다. 이때 X,Y 축이 동시에 1펄스씩 발생하면 실제 이동거리는 √2 가 됩니다. 또한 X,Y,Z 3축이 동시에 1펄스씩 발생하면 실제 이동거리는 √3 이 됩니다. 그러므로 구동속도를 10000[PPS]로 설정하면 실제 이동속도는 최대 √2 X 10000 ≒ 14142[PPS] 혹은 √3 X 10000 ≒17320[PPS] 와 같은 효과가 발생합니다. 그러므로 실제 공간을 이동해 가는 속도(경로속도 혹은 선속도)를 일정하게 유지하기 위해서는 펄스가 발생되는 차원에 의해 자동으로 1/√2, 1/√3 배의 펄스속도[PPS]를 발생시켜야 합니다. WIMC 엔진의 선속도일정기능을 활성화 하면 펄스속도가 자동으로 조절됩니다.

※ ※ 펄스단위의 선속도 일정기능은 연속적인 벡터의 수학적 선속도일정방식에 비해 다소 큰 오차를 발생할 수도 있지만 임의적인 형상경로에 대해 간단하게 일반화 할 수 있으므로 X6V3엔진에서 사용되고 있습니다.
(전자기어비가 적용되지 않을 때 2차원 경로는 대략 최대 8%, 3차원 경로는 대략 최대 13% 오차발생할 수 있습니다.)

<모션마스터프로그램내의 설정>



<관련 라이브러리 함수>

```
WIMC_Enable_CLV(TRUE);          // CLV기능 설정
WIMC_Enable_CLV(FALSE);         // CLV기능 해제
WIMC_Set_CLVAxis(WIMC_XYZ);     // CLV 동작축 설정
WIMC_Get_CLVAxis();             // CLV 동작축 읽기
```

<리셋시 설정값>

선속도 일정축 = X,Y,Z
CLV = Off

25.2.1. WIMC_Enable_CLV

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_CLV
(
              WIMC_BOOL               bEnable
);
```

➲ Description

선속일정 기능을 설정(Enable/Disable) 합니다.

관련 내용 및 설명은 (25.2절)을 참고 하실수 있습니다.

➲ Parameters

bEnable

WIMC_TRUE : CLV 기능 설정

WIMC_FALSE : CLV 기능 해제

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

25.2.2. WIMC_Is_CLV

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_CLV( void);
```

➲ Description

선속도일정모드를 확인합니다.

관련 내용 및 설명은 (25.2절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	선속도 일정모드 입니다.
WIMC_FALSE	선속도 일정모드가 아닙니다.

➲ Example

➲ See Also

25.2.3. WIMC_Set_CLVAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_CLVAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

선속일정 기능 동작축을 설정합니다.

관련 내용 및 설명은 (25.2절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bOn

WIMC_TRUE : 선속일정기능을 설정합니다.

WIMC_FALSE : 선속일정기능을 해제합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

25.2.4. WIMC_Get_CLVAxis

➲ Syntax

```
WIMC_UINT8    WIMC_API                  WIMC_Get_CLVAxis( void);
```

➲ Description

선속일정 기능 동작축을 읽습니다.

관련 내용 및 설명은 (25.2절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

선속일정기능 동작 축을 반환합니다.

Return code/value	Description
WIMC_TRUE	선속도 일정모드 입니다.
WIMC_FALSE	선속도 일정모드가 아닙니다.

➲ Example

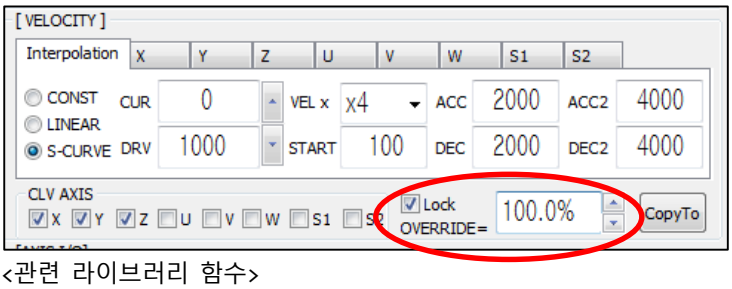
➲ See Also

25.3. 속도 오버라이딩

속도배율 오버라이디이 기능은 구동속도를 0.1%단위로 배율조절할 수 있는 기능을 제공합니다. 속도배율 오버라이딩은 구동속도의 설정값 변경없이 전체 동작속도를 변경할 수 있습니다.
이 기능은 보간동작명령과 스핀들회전명령 각각에 존재합니다. 그러나 IMOVE, CMOVE, XMOVE의 개별축동작에는 영향을 주지 않습니다.

모션마스터프로그램에서는 아래의 그림과 같이 하나의 속도배율 오버라이딩 설정으로 보간동작명령의 속도배율오버라이딩과 스핀들회전명령의 속도배율오버라이딩 파라메터 모두를 설정합니다.
두가지 속도배율오버라이딩 모두 조절단위는 0.1% ~ 1000.0%까지입니다.

보간속도 오버라이딩기능은 현재 오버라이딩값에 관계없이 Lock기능에 의해 항상 100%로 동작시킬 수 있으며, Unlock기능에 의해 오버라이딩값을 적용할 수 있게 됩니다. Lock기능은 보간속도 오버라딩기능의 Disable역할을 하게 됩니다. 오버라이딩 Lock기능은 스핀들회전속도에 대한 오버라이딩기능에는 영향을 주지 않습니다.



<관련 라이브러리 함수>

```
WIMC_Set_VelOverride(123.4);          // 보간명령에 대한 구동속도를 123.4%로 실행시킵니다.
WIMC_Get_VelOverride();               // 현재 보간명령의 속도오버라이딩값을 읽습니다.
WIMC_Set_S1VelOverride(123.4);        // 스핀들 회전명령에 대한 구동속도를 123.4%로 실행시킵니다.
WIMC_Get_S1VelOverride();             // 현재 스핀들 회전명령에 대한 속도오버라이딩값을 읽습니다.
WIMC_Lock_VelOverride();              // 보간명령의 VelOverride효과를 없애고 강제로 100%로 동작시킵니다.
WIMC_Unlock_VelOverride();            // 보간명령의 VelOverride효과를 적용합니다.
WIMC_Is_Lock_VelOverride();           // 현재 보간명령의 VelOverride가 lock상태인지를 확인.
```

<리셋시 설정값>
보간동작명령 속도배율 오버라이딩 = 100.0%
오버라이딩 UNLOCK
스핀들회전명령에 대한 속도배율 오버라이딩 = 100.0%

25.3.1. WIMC_Set_VelOverride

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_VelOverride
(
              WIMC_INT32              percentagex10
);
```

➲ Description

속도 오버라이딩을 10% 단위로 설정합니다.

(값 범위 1~100 (10%~1000%))

➲ Parameters

percentagex10

10% 단위(ex, 1(10%), 10(100%), 100(1000%)

10~100 이내 값(10~1000%)

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상 설정 완료
WIMC_FALSE	err(파라메터)

➲ Example

```
WIMC_Set_VelOverride(5);          // 보간명령에 대한 구동속도를 50%로 실행시킵니다.
```

➲ See Also

25.3.2. WIMC_Get_VelOverride

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_VelOverride( void);
```

➲ Description

보간동작명령의 속도배율오버라이딩 값을 읽습니다.

속도 오버라이딩은 10% 단위로 설정됩니다.
(값 범위 1~100 (10%~1000%))

➲ Parameters

➲ Return Value

Return code/value	Description
1~100	속도오버라이딩 값을 반환합니다(값 범위 1~100 (10%~1000%))

➲ Example

➲ See Also

25.3.3. WIMC_Set_S1VelOverride

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_S1VelOverride
(
              WIMC_INT32              percentagex10
);
```

➲ Description

S1 스핀들 회전명령에 대한 속도오버라이딩값을 설정합니다.
(값 범위 1~100 (10%~1000%))

관련 내용 및 설명은 (25.3절)을 참고 하실수 있습니다.

➲ Parameters

percentage
값 범위 1~100 (10%~1000%)

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상 설정 완료
WIMC_FALSE	err(파라메터)

➲ Example

```
WIMC_Set_S1VelOverride(20);              // 스핀들 회전명령에 대한 구동속도를 200%로 실행시킵니다.
```

➲ See Also

25.3.4. WIMC_Get_S1VelOverride

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_S1VelOverride( void);
```

➲ Description

S1 스핀들 회전명령에 대한 속도오버라이딩값을 읽습니다.
(값 범위 1~100 (10%~1000%))

관련 내용 및 설명은 (25.3절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

속도오버라이딩 값을 반환합니다.

Return code/value	Description
1~100	10%~1000%

➲ Example

➲ See Also

25.3.5. WIMC_Set_S2VelOverride

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_S2VelOverride
(
              WIMC_INT32              percentagex10
);
```

➲ Description

S2 스핀들 회전명령에 대한 속도오버라이딩값을 설정합니다.
(값 범위 1~100 (10%~1000%))

S2축을 지원하는 모션보드에서만 사용가능합니다.

➲ Parameters

percentage
값 범위 1~100 (10%~1000%)

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상 설정 완료
WIMC_FALSE	err(파라메터)

➲ Example

```
WIMC_Set_S1VelOverride(20);              // 스핀들 회전명령에 대한 구동속도를 200%로 실행시킵니다.
```

➲ See Also

25.3.6. WIMC_Get_S2VelOverride

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_S2VelOverride( void);
```

➲ Description

S2 스핀들 회전명령에 대한 속도오버라이딩값을 읽습니다.
(값 범위 1~100 (10%~1000%))

S2축을 지원하는 모션보드에서만 사용가능합니다.

➲ Parameters

➲ Return Value

속도오버라이딩 값을 반환합니다.

Return code/value	Description
1~100	10%~1000%

➲ Example

➲ See Also

25.3.7. WIMC_Lock_VelOverride

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Lock_VelOverride( void);
```

➲ Description

보간명령의 VelOverride효과를 없애고 강제로 100%로 동작시킵니다.

관련 내용 및 설명은 (25.3절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

25.3.8. WIMC_Unlock_VelOverride

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Unlock_VelOverride( void);
```

➲ Description

보간명령의 VelOverride 효과를 적용합니다.

관련 내용 및 설명은 (25.3절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

25.3.9. WIMC_Is_Lock_VelOverride

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_Lock_VelOverride( void);
```

➲ Description

현재 보간명령의 속도배율 오버라이딩이 lock 상태인지를 확인합니다.

관련 내용 및 설명은 (25.3절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	속도배율 오버라이딩 Lock 상태입니다.
WIMC_FALSE	속도배율 오버라이딩 Unlock 상태입니다.

➲ Example

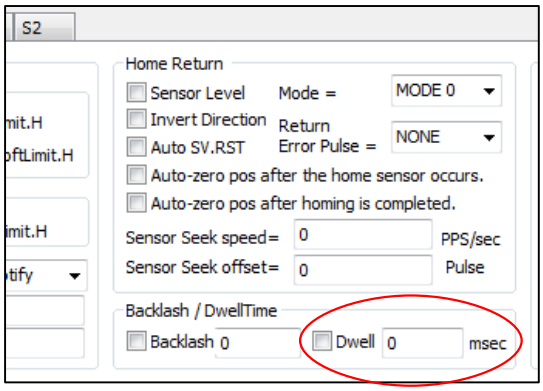
➲ See Also

25.4. 드웰타임

개별축 드웰타입 기능설정

드웰기능은 상태변화가 발생하는 순간에 휴지기능을 주는 것을 말합니다.
흔히 회전방향이 바뀌는 순간에 일정시간 동작을 정지하거나, 가속도의 방향이 바뀌는 순간에 일정시간 정속을 유지하는 등의 동작에 적용됩니다.
개별 축 드웰타임 기능은 원점복귀 동작에서 회전방향이 바뀔 때 잠시 정지하는 시간으로만 사용됩니다.
드웰타임은 msec단위로 부호없는 16비트로 설정됩니다. 그러므로 최대 65535msec입니다.
개별 축 드웰타임 기능은 활성화/비활성화 될 수 있습니다.

<모션마스터프로그램내의 설정>



<관련 라이브러리 함수>

```
WIMC_Enable_DwellTimeAxis(WIMC_X, TRUE);
WIMC_Enable_DwellTimeAxis(WIMC_X, FALSE);
WIMC_Set_DwellTimeAxis(WIMC_X, 0);
WIMC_Get_DwellTimeAxis(WIMC_X);
```

<리셋시 설정값>

기능 OFF

드웰타임값 = 0msec

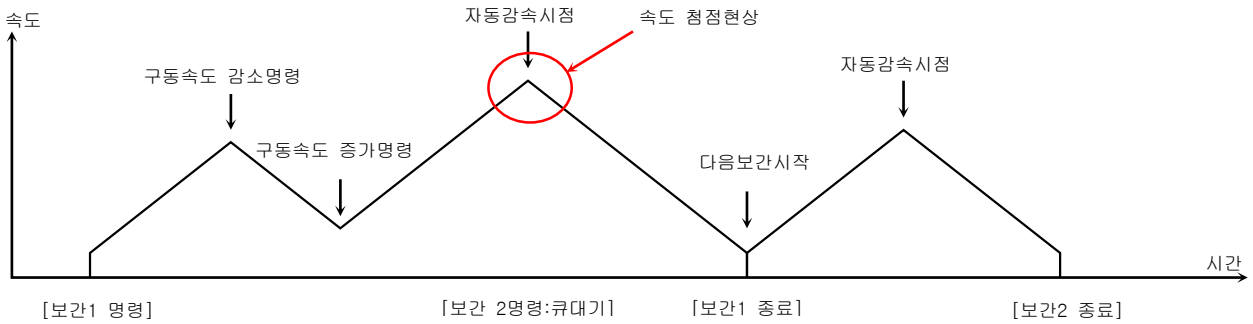
직선가감속에서 정속유지 기능 (DWELL 기능)

직선가감속에서 가속 후 정속구간 없이 바로 감속되는 속도의 첨점현상을 억제하기 위한 기능입니다. 보간동작의 DWELL기능은 가감속의 방향이 변하려 할 때 현재 속도상태를 일정시간 유지하는 기능을 합니다. 속도의 첨점현상은 보간이동거리가 너무 짧아 가감속이 충분하지 못할 때나 새로운 구동속도명령으로 인해 반대방향으로 가감속할 때 발생합니다. 아래의 그림을 참조하세요.

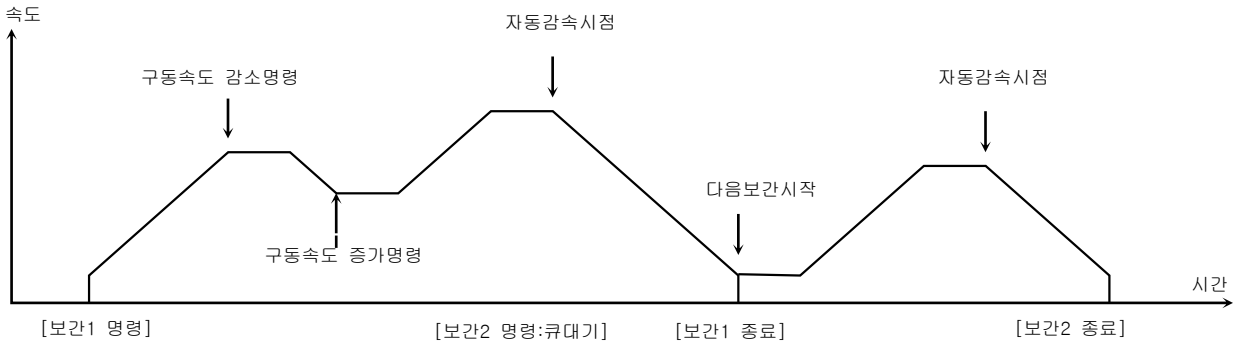
※ 주의 1> 보간동작의 DwellTime 은 직선가감속에서는 정속유지시간으로 동작하며, 개별축 동작설정의 DwellTime 은 원점복귀동작에서의 휴지시간으로 동작합니다. 이 두가지 DwellTime 은 서로 다른 파라메터입니다.

※ 주의 2> 정속유지동작은 보간동작이 처음 시작될 때에도 발생할 수 있습니다. 보간동작이 처음 동작할 때 정속유지동작이 실행되면 마치 보간실행이 늦추어 실행되는 현상처럼 보입니다. 그러므로 안정된 동작만이 고려되고 고속의 연속동작이 필요하지 않을 경우에만 사용하는 것이 바람직합니다.

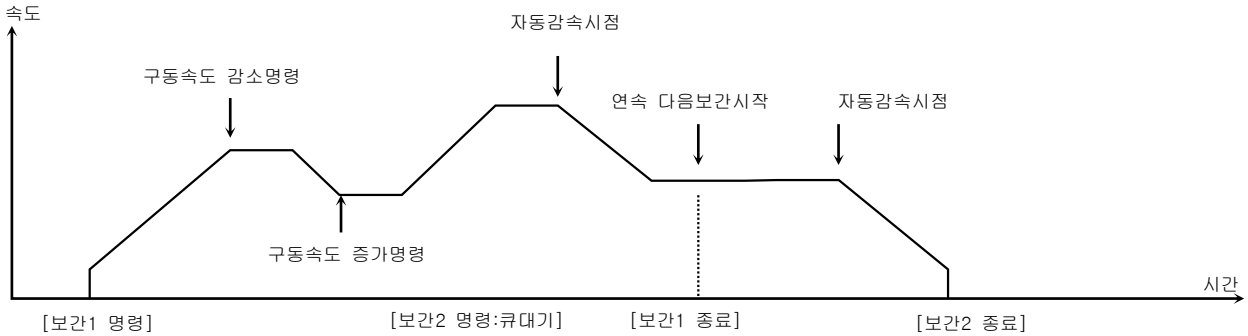
<비연속 보간 : 정속유지시간 = 0 일 때 속도그래프>



<비연속보간 : 드웰타임이 적용되어 정속유지시간이 0 이 아닐 때 속도그래프>



<연속보간 : 정속유지시간이 0 이 아닐 때 속도그래프>

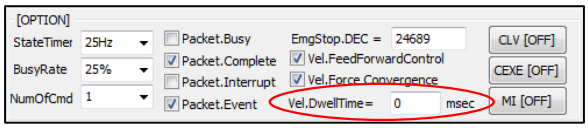


※ **주의> 짧은 이동명령들을 빠르게 연속동작 시키기 위해서는 정속유지시간을 적용하지 않는 것이 유리합니다. 정속유지시간이 적용되면 연속동작사이에 정속유지시간이 적용될 수 있어 비 연속적인 이동동작으로 바뀔 수도 있습니다.**

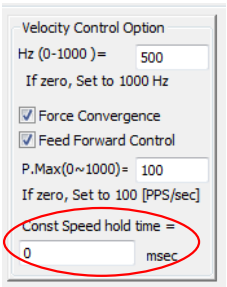
아래에는 모션마스터프로그램에서의 정속유지시간 설정부분을 보여줍니다. 2부분에서 설정가능하며 두가지 모두 같은

파라메터입니다.

<모션엔진 모니터링화면>



<모션엔진 설정화면>



<관련 라이브러리 함수>

```
WIMC_Set_DwellTime(0);
```

<리셋시 설정값>

정속유지시간 = 0

원점복귀 동작전환 휴지타임 (Dwell Time)

드웰타임은 원점복귀 동작에서 원점센서의 SEARCH동작과 회귀동작사이의 정지동작시간을 말합니다. msec단위로 지정하며 원점탐색동작이 보다 안정되게 수행될 수 있도록 도와줍니다.

<관련 라이브러리함수>

```
WIMC_Set_DwellTimeAxis(axis, msecTime);          // 개별축의 드웰타임값으로 원점복귀동작의 드웰기능을 수행합니다
```

25.4.1. WIMC_Set_DwellTime

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_DwellTime
(
              WIMC_INT32              msec
);
```

➲ Description

드웰타임을 설정합니다.

관련 내용 및 설명은 (25.4절)을 참고 하실수 있습니다.

➲ Parameters

msec

milliseconds 단위입니다.(0~65,535 mSec)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

25.4.2. WIMC_Get_DwellTime

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_DwellTime( void);
```

➲ Description

드웰타임을 읽습니다.

관련 내용 및 설명은 (25.4절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

Dwell Time 을 반환합니다.(msec 단위로 최대 65535 msec 입니다.)

Return code/value	Description

➲ Example

➲ See Also

WIMC_Lock_BindingVel();

25.4.3. WIMC_Enable_DwellTimeAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_DwellTimeAxis
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축 드웰타임 기능을 enable/disable 합니다.

관련 내용 및 설명은 (25.4절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 드웰타임 기능 Enable.

WIMC_FALSE : 드웰타임 기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

25.4.4. WIMC_Set_DwellTimeAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_DwellTimeAxis
(
              WIMC_UINT8              axis,
              WIMC_INT32              msec
);
```

➲ Description

특정축 드웰타임을 설정합니다. (msec 단위로 최대 65535 msec 입니다.)

관련 내용 및 설명은 (25.4절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

msec

milliseconds 단위입니다.(0~65,535 mSec)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_DwellTimeAxis(WIMC_X, TRUE);
WIMC_Enable_DwellTimeAxis(WIMC_X, FALSE);
WIMC_Set_DwellTimeAxis(WIMC_X, 0);
WIMC_Get_DwellTimeAxis(WIMC_X);
```

25.4.5. WIMC_Get_DwellTimeAxis

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_DwellTimeAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축 드웰타임을 읽습니다. (msec 단위로 최대 65535 msec 입니다.)

관련 내용 및 설명은 (25.4절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

특정축 드웰타임을 milliseconds 단위로 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

25.5. 속도제어 주파수

25.5.1. WIMC_Set_VelCtrlHz

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_VelCtrlHz
(
              WIMC_INT32              hz
);
```

➲ Description

속도제어 엔진의 실행속도를 설정합니다.

WIMC 엔진의 내부 모션엔진코어는 가장 중요한 부분은 속도제어엔진과 펄스발생엔진입니다. 이 두가지 부분엔진을 합쳐서 모션엔진으로 부릅니다. WIMC 엔진은 보간동작명령을 담당하는 주모션엔진 1개와 개별축동작명령을 담당하는 보조모션엔진 4개 그리고 스핀들 전용의 고속회전을 담당하는 스핀들모션엔진 1개를 포함하고 있습니다.

속도제어엔진은 펄스발생엔진의 펄스주기등을 조절하여 모터가 적절한 가감속도로 움직일 수 있도록 도와줍니다. 펄스발생엔진의 실행주기는 속도제어엔진에서 만들어진 주기값으로 동작되며 초당 최대 수백KHz로 실행될 수 있습니다. 반면에 속도제어엔진은 수백Hz이하로 실행하게 됩니다.

실제 500Hz정도의 속도제어엔진 실행주기는 초당 1M로 이송하는 기계(Feed값 60,000해당) 에서 2mm의 움직임마다 속도제어동작이 이루어지는 것을 의미합니다. WIMC 엔진은 최대 1KHz까지 속도제어엔진의 속도를 조절할 수 있으며 모션시스템에 따라 속도의 변화가 매우 중요할 경우 속도제어엔진의 실행속도를 높여서 사용할 수 있습니다. 최소 설정값은 100Hz입니다.

➲ Parameters

hz

hz단위입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

WIMC_Get_VelCtrlHz();

25.5.2. WIMC_Get_VelCtrlHz

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_VelCtrlHz( void);
```

➲ Description

속도제어 엔진의 실행속도를 읽습니다.

WIMC 엔진의 내부 모션엔진코어는 가장 중요한 부분은 속도제어엔진과 펄스발생엔진입니다. 이 두가지 부분엔진을 합쳐서 모션엔진으로 부릅니다. WIMC 엔진은 보간동작명령을 담당하는 주모션엔진 1개와 개별축동작명령을 담당하는 보조모션엔진 4개 그리고 스핀들 전용의 고속회전을 담당하는 스핀들모션엔진 1개를 포함하고 있습니다.

속도제어엔진은 펄스발생엔진의 펄스주기등을 조절하여 모터가 적절한 가감속도로 움직일 수 있도록 도와줍니다. 펄스발생엔진의 실행주기는 속도제어엔진에서 만들어진 주기값으로 동작되며 초당 최대 수백KHz로 실행될 수 있습니다. 반면에 속도제어엔진은 수백Hz이하로 실행하게 됩니다.

실제 500Hz정도의 속도제어엔진 실행주기는 초당 1M로 이송하는 기계(Feed값 60,000해당) 에서 2mm의 움직임마다 속도제어동작이 이루어지는 것을 의미합니다. WIMC 엔진은 최대 1KHz까지 속도제어엔진의 속도를 조절할 수 있으며 모션시스템에 따라 속도의 변화가 매우 중요할 경우 속도제어엔진의 실행속도를 높여서 사용할 수 있습니다. 최소 설정값은 100Hz입니다.

➲ Parameters

➲ Return Value

속도제어 엔진의 실행속도를 Hz단위로 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

WIMC_Set_VelCtrlHz();

25.6. 속도강제수렴

25.6.1. WIMC_Enable_VelForceConvergence

➲ Syntax

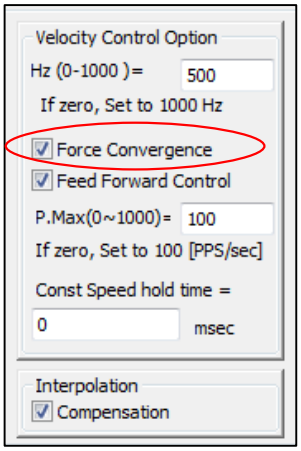
```
WIMC_BOOL     WIMC_API                  WIMC_Enable_VelForceConvergence
(
              WIMC_BOOL                 bEnable
);
```

➲ Description

강제 속도수렴 기능을 설정(Enable/Disable)합니다.

직선가감속은 설정된 구동속도로 정확하게 수렴하지만 S자 가감속은 목표된 속도에 수렴하지 않을 수도 있습니다. 이때 강제수렴기능을 설정하면 현재속도가 구동속도에 접근할 때 강제로 속도를 수렴시키게 됩니다.

오른쪽 그림은 강제 속도수렴 기능을 위한 모션마스터프로그램의 설정부분을 보여줍니다.



➲ Parameters

bEnable

WIMC_TRUE : Enable.
WIMC_FALSE : Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_VelForceConvergence(TRUE);     // 강제속도수렴기능 설정
WIMC_Enable_VelForceConvergence(FALSE);    // 강제속도수렴기능 해제
WIMC_Is_VelForceConvergence();             // 강제속도수렴기능 여부확인
```

25.6.2. WIMC_Is_VelForceConvergence

➲ Syntax

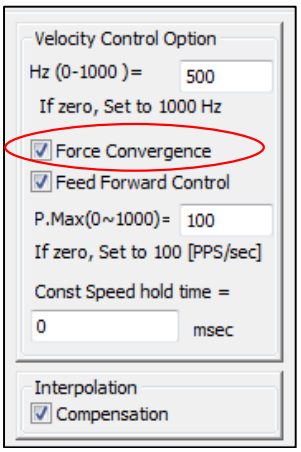
```
WIMC_BOOL     WIMC_API                WIMC_Is_VelForceConvergence( void);
```

➲ Description

강제 속도수렴 기능여부를 확인합니다.

직선가감속은 설정된 구동속도로 정확하게 수렴하지만 S자 가감속은 목표된 속도에 수렴하지 않을 수도 있습니다. 이때 강제수렴기능을 설정하면 현재속도가 구동속도에 접근할 때 강제로 속도를 수렴시키게 됩니다.

오른쪽 그림은 강제 속도수렴 기능을 위한 모션마스터프로그램의 설정부분을 보여줍니다.



➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	강제속도 수렴기능 설정 상태입니다.
WIMC_FALSE	강제속도 수렴기능 해제 상태입니다.

➲ Example

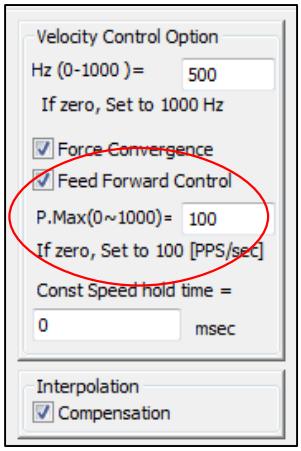
➲ See Also

```
WIMC_Enable_VelForceConvergence(TRUE);     // 강제속도수렴기능 설정
WIMC_Enable_VelForceConvergence(FALSE);    // 강제속도수렴기능 해제
WIMC_Is_VelForceConvergence();             // 강제속도수렴기능 여부확인
```

25.7. 피드포워드제어기능 FFC

모션엔진의 속도제어기는 일반적으로 수백~수KHz 내에서 동작합니다. 이 속도는 펄스발생기의 처리속도에 비해 비교적 작습니다. 이러한 속도제어기의 보다 낮은 처리속도로 인해 펄스발생기와 속도제어기 사이에 동작 불균형이 발생할 수 있습니다. 이 불균형은 속도제어기의 처리 속도가 펄스발생기의 처리속도를 따라가지 못하기 때문에 발생하는 것으로써 결국 동작 불균형을 초래합니다. WIMC 엔진은 이러한 불균형을 조절하기 위해 속도 피드포워드제어를 실행합니다. 이 설정은 옵션화 되어 Enable/Disable 할수 있습니다. 대부분 응용에서 이 설정은 Enable하는 것이 바람직합니다. 속도의 피드포워드제어가 적용되면 속도제어기는 펄스발생기의 동작과 정지에 균형을 맞추려고 노력합니다. 피드포워드제어는 결국 가감속 파라메터값을 내부적으로 조금씩 조절하게 됩니다. 조절되는 방식은 P제어(비례제어)방식입니다. 이때 조절되는 속도변화의 최대크기를 설정할 수 있습니다. 이 값을 P값으로 하여 간주될 수 있습니다. 이 값이 0 이면 내부적으로 정해진 값(100)으로 결정됩니다.

아래에 모션마스터프로그램에서의 속도 피드포워드 제어 설정부분을 보여줍니다.



< 관련 라이브러리 함수 >

```
WIMC_Enable_FFC(TRUE);          // FFC(Feed Forward Control) 기능 설정
WIMC_Enable_FFC(FALSE);         // FFC(Feed Forward Control) 기능 해제
WIMC_Is_FFC();                  // FFC설정여부 읽기
WIMC_Set_FFC_PMax(100);
```

< 리셋시 설정값 >

기능 ON

FFC.PMax = 100

25.7.1. WIMC_Enable_FFC

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_FFC
(
              WIMC_BOOL               bEnable
);
```

➲ Description

속도 피드 포워드 제어기능(Feed Forward Control) 기능을 설정(Enable/Disable)합니다.

관련 내용 및 설명은 (25.7절)을 참고 하실수 있습니다.

➲ Parameters

bEnable

WIMC_TRUE : 속도 피드 포워드 제어기능 Enable.

WIMC_FALSE : 속도 피드 포워드 제어기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_FFC(TRUE);          // FFC(Feed Forward Control) 기능 설정
WIMC_Enable_FFC(FALSE);         // FFC(Feed Forward Control) 기능 해제
WIMC_Is_FFC();                  // FFC설정여부 읽기
WIMC_Set_FFC_PMax(100);
```

25.7.2. WIMC_Is_FFC

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_FFC( void);
```

➲ Description

속도 피드 포워드 제어기능(Feed Forward Control) 설정 여부를 확인합니다.

관련 내용 및 설명은 (25.7절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

설정상태를 반환합니다.

Return code/value	Description
WIMC_TRUE	속도 피드 포워드 제어기능이 설정된 상태입니다.
WIMC_FALSE	속도 피드 포워드 제어기능이 설정된 상태가 아닙니다.

➲ Example

➲ See Also

```
WIMC_Enable_FFC(TRUE);          // FFC(Feed Forward Control) 기능 설정
WIMC_Enable_FFC(FALSE);         // FFC(Feed Forward Control) 기능 해제
WIMC_Is_FFC();                  // FFC설정여부 읽기
WIMC_Set_FFC_PMax(100);
```

25.7.3. WIMC_Set_FFC_PMax

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_FFC_PMax
(
              WIMC_INT32              p
);
```

➲ Description

속도의 피드포워드제어가 적용되면 속도제어기는 펄스발생기의 동작과 정지에 균형을 맞추려고 노력합니다.
피드포워드제어는 결국 가감속 파라메터값을 내부적으로 조금씩 조절하게 됩니다. 조절되는 방식은
P제어(비례제어)방식입니다. 이때 조절되는 속도변화의 최대크기를 설정합니다.

관련 내용 및 설명은 (25.7절)을 참고 하실수 있습니다.

< 리셋시 설정값 >
FFC.PMax = 100

➲ Parameters

p

0~1000

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Enable_FFC(TRUE);          // FFC(Feed Forward Control) 기능 설정
WIMC_Enable_FFC(FALSE);         // FFC(Feed Forward Control) 기능 해제
WIMC_Is_FFC();                  // FFC설정여부 읽기
WIMC_Set_FFC_PMax(100);
```

25.8. Automatic transistion 속도/위치제어 자동절환

Vel/Pos Auto Transition (회전제어 & 위치제어간 자동전환기능)

스핀들 전용명령은 모션엔진내부에 개별축 회전명령 (ROTATE) 으로 일반화 되어 있습니다.
모션엔진은 일반적인 위치제어용 동작명령과 회전전용명령을 구분하여 동작시킵니다. 위치제어용 동작명령은 현재위치카운터값이 점진적으로 변화되면서 명령을 수행하지만, 회전전용명령은 현재위치카운터값이 변하지 않고 연속으로 회전합니다.

스핀들 전용명령이 적용된 축은 상기의 위치제어용 동작명령과 회전전용명령을 혼합하여 사용될 수 있습니다. 이런 경우는 흔히 스핀들 축이 서보모터와 같은 장치일 때 일반적인 회전의 절삭동작과 암나사산을 절삭하기 위해 Z축과 선형(직선)보간동작을 수행하게 되는 탭핑(Tapping)작업의 교환에서 발생됩니다.

스핀들의 전용회전명령은 매우 빠르게 회전하는 것이 일반적이며 탭핑작업은 비교적 느리게 동작하는 것이 일반적입니다.

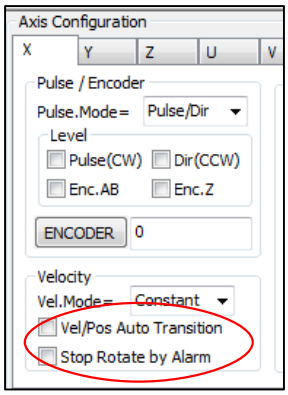
만약 현재 스핀들 축이 전용회전명령에 의해 회전동작이 실행되고 있을 때 스핀들축과 Z축의 직선보간명령을 전송하게 되면 모션엔진은 COMMAND 알람을 발생하게 됩니다. 모션엔진은 현재 회전명령에 의해 점유된 축을 위치제어를 위한 동작으로 바꿀 수 없음을 알려줍니다.

그러나 Vel/Pos Auto Transition을 설정하게 되면 자동으로 회전동작이 감속정지한 후 보간동작이 이어서 실행할 수 있게 됩니다.

Stop rotate by alarm (알람발생시 회전정지기능)

WIMC 엔진은 스핀들 전용명령에 의해 스핀들축이 회전하고 있을 때 알람이 발생해도 스핀들의 회전은 정지하지 않게 됩니다. 만약 알람에 의한 비상실행상태에서는 스핀들이 자동으로 정지하고 싶다면 이 옵션을 활성화 해야 합니다.

<모션마스터프로그램내의 설정>



<관련 라이브러리 함수>

```
WIMC_Enable_VelPosAutoTransition(TRUE);
WIMC_Enable_VelPosAutoTransition(FALSE);
WIMC_Is_VelPosAutoTransition();
WIMC_Enable_StopRotateByAlarm(TRUE);
WIMC_Enable_StopRotateByAlarm(FALSE);
WIMC_Is_StopRotateByAlarm();
```

25.8.1. WIMC_Is_VelPosAutoTransition

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_VelPosAutoTransition
(
              WIMC_UINT8                axis
);
```

➲ Description

회전제어 및 위치제어간 자동 전환기능 설정상태를 읽습니다.

관련 내용 및 설명은 (25.8절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	자동전환 설정상태입니다.
WIMC_FALSE	자동전환 설정상태가 아닙니다.

➲ Example

➲ See Also

25.8.2. WIMC_Enable_VelPosAutoTransition

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Enable_VelPosAutoTransition
(
              WIMC_UINT8                axis,
              WIMC_BOOL                 bEnable
);
```

➲ Description

회전제어 및 위치제어간 자동 전환기능을 설정(Enable/Disable)합니다.

관련 내용 및 설명은 (25.8절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 자동 전환기능을 enable 합니다.

WIMC_FALSE : 자동 전환기능을 disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

26. 위치 제어

26.1. Current Position

26.1.1. WIMC_Set_CurPos

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_CurPos
(
              WIMC_UINT8              axis,
              WIMC_INT32              pulse
);
```

➲ Description

현재위치를 설정합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulse

펄수 수입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

26.1.2. WIMC_Get_CurPos

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_CurPos
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 현재위치를 읽습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

특정축의 현재위치를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

WIMC_Set_CurPos();

26.2. Encoder

26.2.1. WIMC_Set_EncPos

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_EncPos
(
              WIMC_UINT8              axis,
              WIMC_INT32              pos
);
```

➲ Description

특정축의 엔코더 위치값을 설정합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulse

엔코더 위치(펄스)값을 지정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

26.2.2. WIMC_Get_EncPos

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_EncPos
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 엔코더 위치(펄스)값을 읽습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

특정축의 엔코더 위치(펄스)를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

WIMC_Set_CurPos();

26.2.3. WIMC_Get_EncZCount

➲ Syntax

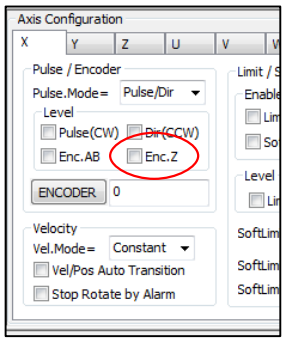
```
WIMC_INT32    WIMC_API                WIMC_Get_EncZCount
(
              WIMC_UINT8              axis
);
```

➲ Description

회전엔코더 Z 상의 카운터(횟수)를 읽습니다.

WIMC 엔진은 회전엔코더 Z상의 입력만을 받습니다. 엔코더 Z상은 원점복귀 및 회전 RPM측정을 위한 카운터 등에 필요로 합니다. 회전엔코도 신호설정기능은 엔코더의 Z상 활성화 레벨 설정기능만을 갖고 있습니다.
엔코더Z상의 카운터값은 16비트 부호없는 값으로 0~65535 값을 반복합니다.
(0,1,2,3....,65534,65535, → 0,1,2,3.... 반복)
엔코더Z상의 카운터값은 읽기만 가능하며 특정값의 설정동작은 허용되지 않습니다.

<모션마스터프로그램내의 설정>



➲ Parameters

axis
적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

엔코더Z상의 카운터값은 16비트 부호없는 값으로 0~65535 값을 반복합니다.
(0,1,2,3....,65534,65535, → 0,1,2,3.... 반복)

Return code/value	Description

➲ Example

➲ See Also

```
WIMC_Set_EncZLevel(WIMC_LEVEL_H);
WIMC_Set_EncZLevel(WIMC_LEVEL_L);
WIMC_Get_EncZCount(WIMC_X);
```

26.3. Gear

WIMC 엔진의 전자기어비 기능은 스텝모터드라이브와 같이 내부적으로 전자기어비기능이 없는 단순형 모터 드라이브를 위해 제공되는 기능입니다. 서보드라이브와 같은 인텔리젼트형의 드라이브들은 내부적으로 전자기어비기능을 갖추고 있어 별도의 전자기어비 기능을 필요로 하지 않습니다.

※ WIMC 엔진의 전자기어비 기능을 사용하면 모션엔진코어의 연산부하를 가중시켜 최대 펄스 출력주파수의 70% 이내로 사용하는 것이 안전합니다.

WIMC 엔진의 전자기어비는 동작주축을 기준으로 다른 축의 펄스 발생비율을 조절하는 방식으로 동작됩니다. 또한 전자기어비의 기능을 적용되는 축과 적용하지 않는 축을 선택할 수 있습니다.

※ 전자기어비의 동작주축이란 동일 이동거리를 움직일 때 가장 많은 펄스를 발생해야하는 축을 말합니다.

➲ 전자기어비의 적용여부에 대한 가장 중요한 기준은 원호보간입니다.

- 원호보간이 사용되는 축은 전자기어비를 적용하는 것이 바람직합니다.
  전자기어비를 사용하지 않으면 1펄스 당 움직이는 거리비율이 달라 원이 아닌 타원의 형태로 동작하기 때문입니다.(이 기능을 역으로 타원형상의 보간동작에 이용할 수 도 있습니다.)

- 원호보간이 사용되지 않고 오직 직선보간이나 단순한 이동명령만이 사용된다면 전자기어비 기능을 적용하지 않더라도 위치는 정확하게 이동하게 됩니다.
  그러나 전자기어비를 적용하지 않은 축의 단위 펄스당 이동거리가 달라지면 선속도 일정동작은 정확하지 않게 됩니다.

➲ 전자기어비는 스핀들 회전명령을 제외한 모든 동작명령 (보간동작명령, 독립이송명령, 연속이송명령, 조건이송명령)에 적용됩니다.

➲ 극단적인 전자기어비의 설정은 WIMC 엔진의 연산효율이 떨어져 펄스발생속도를 현저하게 낮추게 됩니다.

전자기어비는 동작 주축을 기준으로 다른 축들의 펄스발생비율을 조절하는 방식입니다. 동작 주축이란 1 펄스 당 이동값(Travel Value Per Pulse)이 가장 작은 축을 말합니다. 이 축은 다른 축들에 비해 가장 많은 펄스를 출력해야하는 축이 되며, 이 축을 기준으로 다른 축들의 펄스발생비율이 조절됩니다.

WIMC 엔진의 동작연산의 부하량은 동작 주축의 펄스연산부하가 됩니다.
그러므로 아래와 같은 극단적인 사용은 엔진의 보간동작속도를 크게 떨어뜨릴 수 있습니다.
예> X,Y축이 전자기어비에 적용받으며 아래의 값으로 설정되었다고 가정합니다.

X 의 전자기어비 값 = 1000
Y 의 전자기어비 값 = 10

이 의미는 X축이 1000개의 펄스를 출력할 동안 Y는 10개의 펄스만을 출력하는 비율로 모든 동작명령을 수행하라는 뜻입니다.
즉 펄스의 발생비율이 100 대 1이 됩니다.
만약 Y축 단독으로 초당 100펄스속도로 직선이동하는 명령을 내리더라도 모션엔진은 100PPS X 100(전자기어비동작주축의 연산수)만큼 곱해져서 내부연산량은 10,000PPS의 속도로 펄스를 출력하는 연산량과 동일하게 됩니다.

그러므로 전자기어비를 설정할 경우 각 축의 펄스발생비율이 최대한 비슷한 것이 유리합니다.

➲ 전자기어비의 설정값은 32 비트 부호있는 정수입니다. 내부연산의 안전을 위해 26 비트 크기(0~67108864) 이내로 설정하는 것이 바람직합니다.

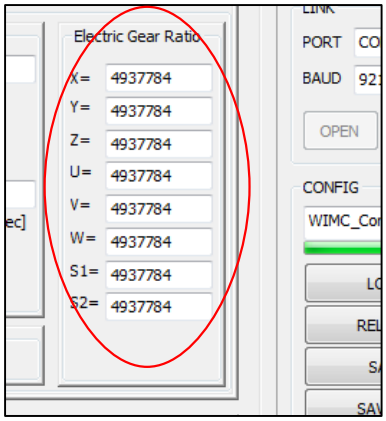
동작 주축을 1.0으로 하여 다른 축들의 펄스발생비율을 실수영역으로 변환하다면 소수점 8자리의 정밀도(≒0.000000015)를 얻을 수 있습니다.

➲ WIMC 엔진은 전자기어비의 적용축과 비적용축에 대한 구분을 동작주축의 전자기어비값과 동일한 값을 갖는 축인지 아닌지를 비교하여 구분합니다.

X 전자기어비값 = 1000
Y 전자기어비값 = 680
Z 전자기어비값 = 450
U 전자기어비값 = 1000
V 전자기어비값 = 1000
W 전자기어비값 = 1000
S1 전자기어비값 = 1000
S2 전자기어비값 = 1000

전자기어비가 적용되는 축은 X를 기준으로 Y, Z축이며 다른 축들은 전자기어비 동작주축인 X축 값과 동일하므로 전자기어비가 적용되지 않는 축이 됩니다.
※ 실제 전자기어비가 적용되지 않는다는 말보다는 X축의 펄스발생비율과 동일하다는 것이 보다 정확한설명입니다.

<모션마스터프로그램내의 설정>



<관련 라이브러리함수>

```
WIMC_Set_GearRatio( WIMC_XYZ, 4937784 );        // X, Y, Z 축의 전자기어비값을 4937784로 설정합니다.
WIMC_Get_GearRatio( WIMC_X);                    // X축에 대한 전자기어비값을 읽습니다.
```

26.3.1. WIMC_Set_GearRatio

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_GearRatio
(
              WIMC_UINT8              axis,
              WIMC_INT32              gearRatio
);
```

➲ Description

특정축의 전자기어비를 설정합니다.

WIMC_Set_GearRatio(WIMC_XYZ, 4937784); // X, Y, Z 축의 전자기어비값을 4937784로 설정합니다.

관련 내용 및 설명은 (26.3절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

gearRatio

전자기어비 값입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

26.3.2. WIMC_Get_GearRatio

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_GearRatio
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 전자기어비 설정값을 읽습니다.

관련 내용 및 설명은 (26.3절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

특정축의 현재위치를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

WIMC_Set_CurPos();

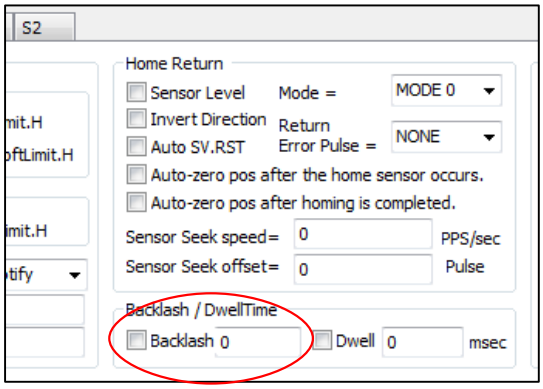
26.4. Backlash

백래시는 회전방향이 바뀔 때 특정 이동량 만큼 따라가지 못하는 현상을 말합니다. (이동 불감대영역)
특히 리드스크류를 이용한 이송동작이나 기어장치를 이용한 동력전달로 움직일 때 나사산이나 기어치차간의 결합여유로 인해 역동작시 결합여유량만큼 움직이지 않게 됩니다.

백래시기능은 회전방향이 바뀔 때 실제로 움직이지 않는 양만큼을 보상하는 펄스를 출력하는 기능을 말합니다. 백래시 기능은 개별축마다 활성화/ 비활성화 할 수 있습니다.

- WIMC 엔진의 백래시기능은 스핀들 회전명령을 제외한 모든 이동동작명령에 백래시기능이 적용됩니다.
- 백래시 펄스속도는 현재펄스 발생속도가 됩니다. 대부분 역회전시 초속도로 움직이므로 연속실행모드가 아니다면 초속도와 동일하게 됩니다.

<모션마스터프로그램내의 설정>



<관련 라이브러리 함수>

```
WIMC_Enable_Backlash(WIMC_X, FALSE);      // 특정축의 백래시동작을 활성화/비활성화 설정합니다.
WIMC_Set_Backlash(WIMC_X, 0);             // 특정축의 백래시 펄스값을 설정합니다.
WIMC_Get_Backlash(WIMC_X);                // 특정축의 백래시 펄스값을 읽습니다.
```

<리셋시 설정값>
모든축 백래시 기능=OFF
모든축 백래시 카운터값 = 0

26.4.1. WIMC_Enable_Backlash

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_Backlash
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축 백래시 기능을 Enable/Disable 합니다.

관련 내용 및 설명은 (26.4절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 백래시 기능 Enable.

WIMC_FALSE : 백래시 기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

26.4.2. WIMC_Set_Backlash

➲ Syntax

```
WIMC_BOOL      WIMC_API                WIMC_Set_Backlash
(
               WIMC_UINT8              axis,
               WIMC_INT32              pulse
);
```

➲ Description

특정축의 백래시 펄스값을 설정합니다.

관련 내용 및 설명은 (26.4절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulse

백래시 카운터값(펄스)을 지정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

26.4.3. WIMC_Get_Backlash

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_Backlash
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 백래시 펄스값을 읽습니다.

관련 내용 및 설명은 (26.4절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

특정축의 백래시 펄스값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

27. 보간명령

27.1. 직선보간

27.1.1. WIMC_Line

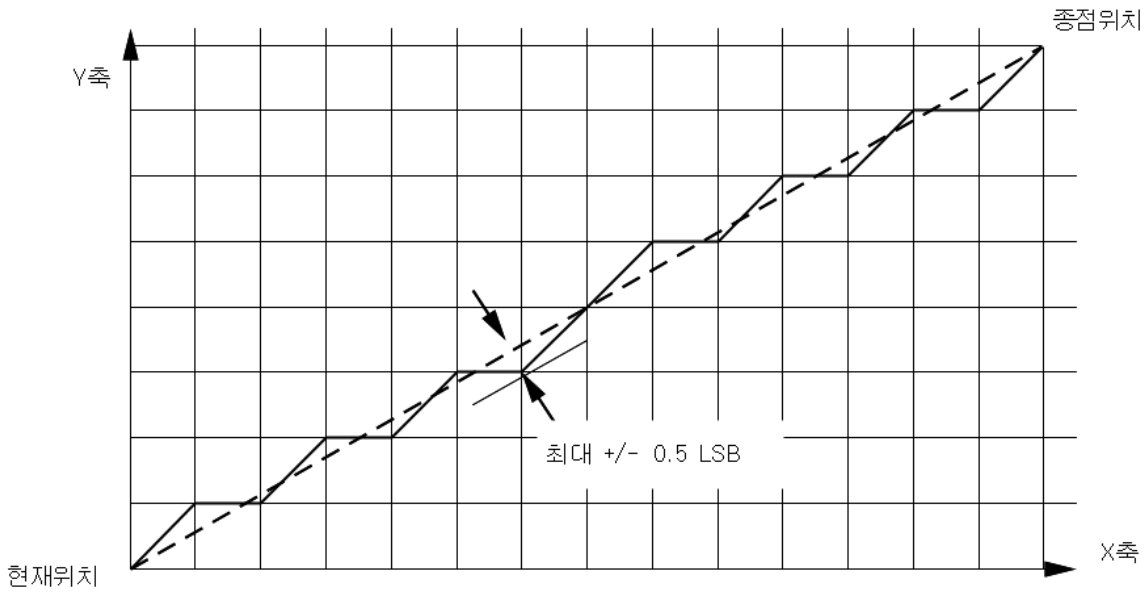
➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Line
(
              WIMC_UINT8              axis,
              WIMC_INT32              pulse[]
);
```

➲ Description

특정축과 종료위치의 펄스카운터값을 지정하여 직선보간명령을 실행합니다.

직선보간명령은 스핀들 축을 포함해서 제어가능한 모든 축에 적용됩니다. 최종 이동할 종점좌표를 지정해야 합니다. 아래에 직선보간 동작을 그림으로 나타냅니다.



직선보간은 최대 +/- 0.5LSB의 정밀도로 각 축에 해당하는 비율로 펄스를 출력합니다. 그러므로 1펄스당 0.01mm을 움직인다면 펄스의 출력에 의한 오차는 0.005mm로 가정할 수 있습니다.

<관련 라이브러리 함수>
WIMC_Line(axis, pos[]);
// axis는 직선보간에 적용되는 축을 지정하고, pos[]는 종료위치의 펄스카운터값을 지정합니다.
// 배열내의 요소수는 기본적으로 8개입니다(X,Y,Z,U,V,W,S1,S2축).

※pos 의 배열크기는 8 입니다. 이 배열크기는 위칸 모션컨트롤러(WIMC_)의 지원 축수입니다. WIMC 엔진도 내부적으로는 최대 8 축으로 정의되었으며 단지, 동작 구현만 6 축으로 제한되었습니다. 그러므로 내부의 모든 설정

파라메터들은 기본적으로 8 축 전체를 사용하고 있습니다. 위칸 모션컨트롤러의 제품들은 사용축수에 관계없이 기본적으로 배열크기 8 을 사용하며 이 크기의 배열을 전달하여 명령을 실행하는 것이 바람직합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pos[]

종료위치의 펄스카운터값입니다.

배열내의 요소수는 기본적으로 8개입니다(X,Y,Z,U,V,W,S1,S2축).

➲ Return Value

Return code/value	Description

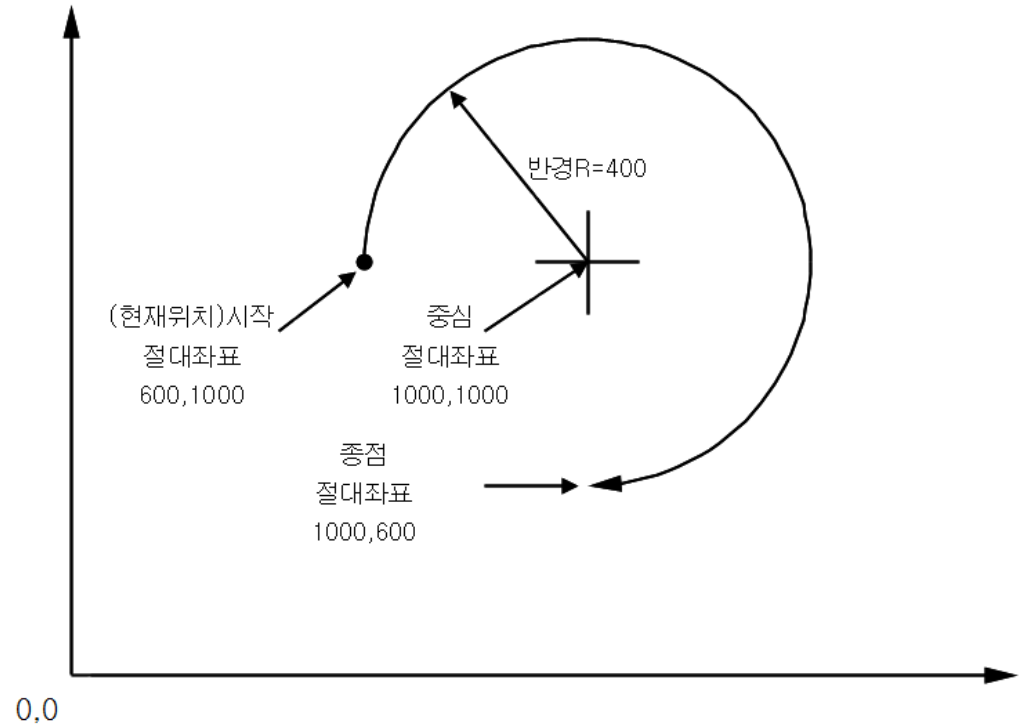
➲ Example

```
WIMC_Enter_QueueCmd();

// 속도설정
WIMC_Set_VelMode( WIMC_VELMODE_LINEAR);

WIMC_Set_DriveVel( 10000);
WIMC_Set_StartVel( 500);
WIMC_Set_Acc( 500);
WIMC_Set_Dec( 500);

// 이동량설정
WIMC_INT32        pulse[8] = { 0, };
pulse[WIMC_AXISNO_X] = 1000;

// 라인 명령
WIMC_Line( WIMC_X, pulse);

WIMC_Leave_QueueCmd();
```

➲ See Also

27.2. 원호보간

CW(ClockWise)는 시계방향을 뜻하며 CCW(Count ClockWise)는 시계반대방향을 의미합니다.
CW원호보간에 내용은 CCW원호보간에서 동일하게 적용됨으로 생략합니다.
원호보간을 실행하기 위해서 2개의 위치정보를 필요로 합니다. 하나는 원호보간하고자 하는 종점위치이며 다른 하나는 원호의 중심위치입니다. 다음 그림을 통해서 살펴봅니다.



상기 그림에 대한 원호보간의 상대위치정보들을 계산해보면 아래와 같습니다.

<상대좌표 입력방식의 경우>
▶ 종점좌표 X: [R07].X = 400
▶ 종점좌표 Y: [R07].Y = -400
▶ 중심좌표 X: [R08].X = 400
▶ 중심좌표 Y: [R08].Y = 0

원호보간은 임의의 2축에 대해서만 가능함으로 축설정바이트는 항상 2축을 지정해야 합니다. 그렇지 않으면 명령오류(Command Error)로 인해 알람이 발생합니다. 명령오류에 의한 알람발생은 명령을 중지시키고 비상실행상태로 전환됩니다.

<관련 라이브러리 함수>
```
WIMC_ArcCW (axis, centerPos[], endPos[]);
WIMC_ArcCCW (axis, centerPos[], endPos[]);
```

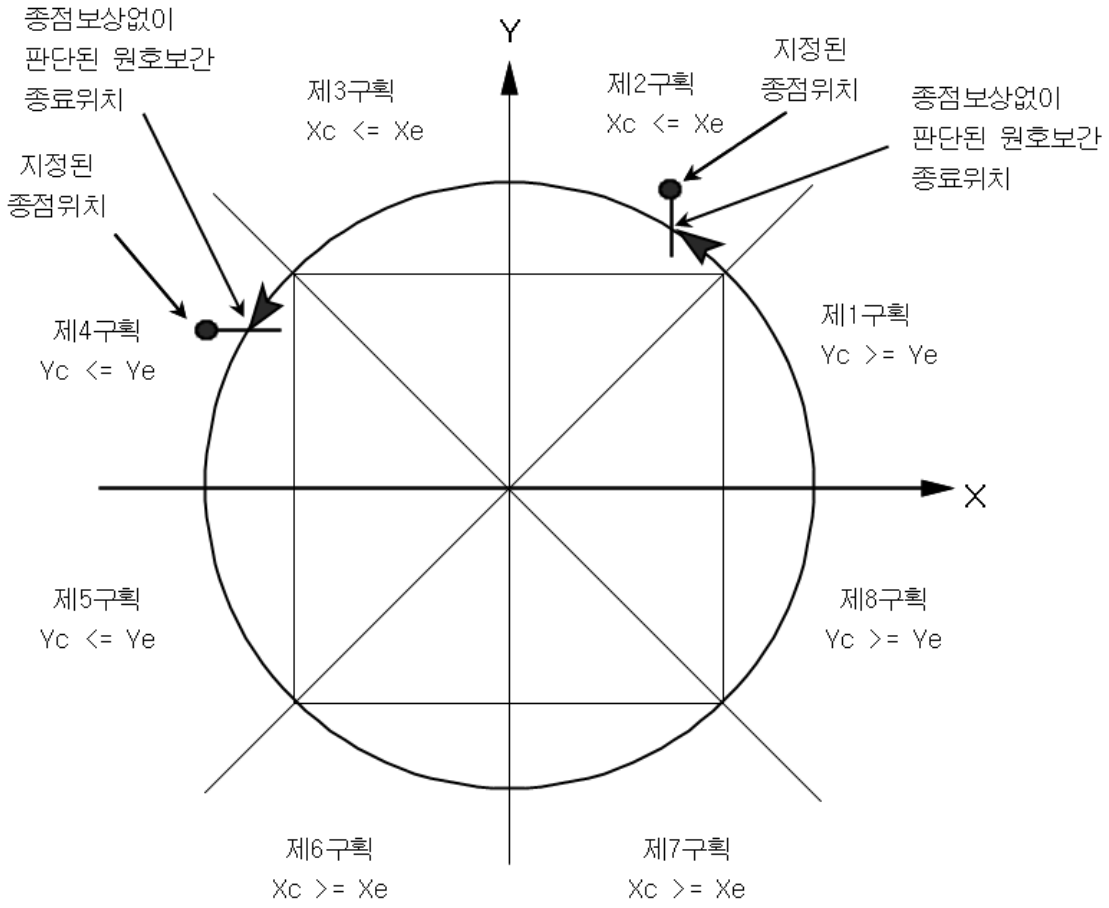
> 완전한 원 보간
시작위치와 종점위치가 같다면 360도를 회전하는 완전한 원을 생성합니다.

> 원호보간의 종점보상
직선보간은 지정된 종점위치로 정확하게 이동을 완료합니다. 그러나 원호보간의 경우 상위제어기가 정확한 종점위치를 계산하지 않으면 실제 원호보간의 동작이 완료된 위치와 다를 수 있습니다. 특히 상위제어기가 실수영역에서 계산한 후

정수화하여 명령을 전송하게 될 때 +/-1LSB(Pulse)의 오차가 발생할 수 있습니다. 이러한 종점오차는 지속적으로 누적되어 큰 오차값을 갖게 될 수도 있습니다. WIMC 엔진은 종점보상기능을 통해서 이러한 원치 않는 누적오차가 발생하지 않도록 지정된 종점위치로 강제로 이동할 수 있는 기능을 제공합니다.

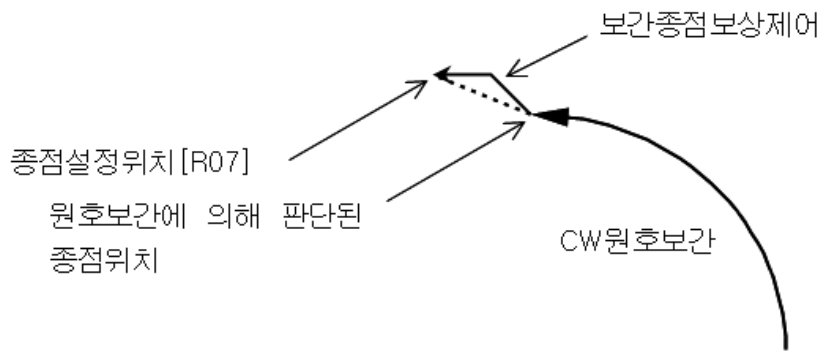
아래의 그림은 원호보간의 종점판단방식을 나타내고 있습니다.

<CCW 원호보간 종점판단방식>



WIMC 엔진은 상기 그림과 같이 8개의 구획으로 나누어서 원보호간을 수행합니다. 각 구획마다 현재 원호보간의 종점판단 조건식을 표시하고 있습니다. 여기서 Xc, Yc는 현재 보간의 위치값이며 Xe, Ye는 최종 원호보간의 종점위치값입니다.
CW원호보간에서는 부등호의 방향만 바뀌게 됩니다.
보간의 종점보상은 모션엔진의 원호보간에서 판단한 종료위치가 원호명령의 종점위치와 다를 때 그 차이만큼의 펄스를 강제로 발생하는 동작을 말합니다. 이때 펄스는 직선보간 형태가 아닌 1:1의 비율로 발생합니다.

<보간종점보상제어 동작>



<관련 라이브러리 함수>

```
WIMC_Enable_IEndComp(TRUE);              // 종점보상기능 활성화
WIMC_Enable_IEndComp(FALSE);             // 종점보상기능 비활성화
```

> 원호보간의 종점보상 알람 발생

종점보상기능이 활성화 되었을때, 종점보상펄스가 1Pulse를 초과하게 되면 종점보상 알람을 발생하도록 설정할 수 있습니다. 종점보상 알람은 현재동작을 중지시키고 비상실행상태로 전환하게 됩니다. 종점보상펄스가 1pulse를 초과한 것은 상위제어기가 원호보간에 대한 종점위치를 정확하게 계산하지 못했음을 의미합니다.

<관련 라이브러리 함수>

```
WIMC_Set_AlarmOption(WIMC_ALARMOPTION_IENDCOMP);   // 보간종점보상알람 설정
WIMC_Clr_AlarmOption(WIMC_ALARMOPTION_IENDCOMP);   // 보간종점보상알람 해제
```

27.2.1. WIMC_ArcCCW

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_ArcCCW
(
              WIMC_UINT8              axis,
              WIMC_INT32              centerPos[],
              WIMC_INT32              endPos[]
);
```

➲ Description

원호보간하고자 하는 종점위치와 원호의 중심 위치로 시계반대방향(count clockwise)의 원호보간을 실행합니다.

원호보간은 임의의 2축에 대해서만 가능함으로 축설정바이트는 항상 2축을 지정해야 합니다. 그렇지 않으면 명령오류(Command Error)로 인해 알람이 발생합니다. 명령오류에 의한 알람발생은 명령을 중지시키고 비상실행상태로 전환됩니다.

관련 내용 및 설명은 (27.2절)을 참고 하실수 있습니다.

➲ Parameters

axis
보간에 적용되는 축을 지정합니다.(WIMC_X, WIMC_Y, WIMC_Z,WIMC_U, WIMC_V, WIMC_W, WIMC_S1, WIMC_S2)

centerPos
원호의 중심위치(펄스)입니다.

endPos
종점 위치(펄스)입니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

27.2.2. WIMC_ArcCW

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_ArcCW
(
              WIMC_UINT8              axis,
              WIMC_INT32              centerPos[],
              WIMC_INT32              endPos[]
);
```

➲ Description

원호보간하고자 하는 종점위치와 원호의 중심 위치로 시계방향(clockwise)의 원호보간을 실행합니다.

원호보간은 임의의 2축에 대해서만 가능함으로 축설정바이트는 항상 2축을 지정해야 합니다. 그렇지 않으면 명령오류(Command Error)로 인해 알람이 발생합니다. 명령오류에 의한 알람발생은 명령을 중지시키고 비상실행상태로 전환됩니다.

관련 내용 및 설명은 (27.2절)을 참고 하실수 있습니다.

➲ Parameters

axis
보간에 적용되는 축을 지정합니다.(WIMC_X, WIMC_Y, WIMC_Z,WIMC_U, WIMC_V, WIMC_W, WIMC_S1, WIMC_S2)

centerPos[]
원호의 중심위치입니다.

endPos
종점 위치입니다.

➲ Return Value

Return code/value	Description
WIMC_TRUE	정상처리 되었습니다.
WIMC_FALSE	실패 했습니다. 자세한 정보는 WIMC_Get_LastErr()를 통해 확인 할 수 있습니다.

➲ Example

➲ See Also

27.3. 비트패턴

비트패턴보간은 2bit값을 하나의 펄스 동작값으로 정의하여 각 축마다 64개의 펄스패턴을 하나의 보간동작으로 실행할 수 있는 명령입니다. WIMC 엔진은 2bit를 하나의 펄스가 아닌 다수의 펄스를 하나의 비트패턴으로 동작시킬 수 있습니다. 이 펄스묶음의 동작단위를 BPU(Bit Pattern Unit)라 합니다. BPU값은 16비트 부호없는 정수값으로 0~65535까지 가능합니다.

2bit에 대한 펄스동작은 아래와 같습니다.

비트값	동작
00	펄스발생안함
01	BPU 설정 만큼의 CW펄스 발생
10	BPU 설정 만큼의 CCW펄스 발생
11	펄스발생종료(보간종료)

<관련 라이브러리 함수>

```
WIMC_Set_BPU( 100 ); // 비트패턴당 100펄스 출력하도록 설정합니다.
WIMC_Get_BPU(); // BPU 설정값을 읽습니다.
WIMC_BitPattern( axis, bitPatternData[][8], bitPatternNo);
```

bitPatternData[][8]은 2차원 부호없는 32비트 데이터 배열입니다. 최대 64패턴이므로 2비트x64패턴=128비트, 32비트 단위로 나누면 bitPatternData는 최대 [4][8]이 됩니다. 마지막 bitPatternNo는 최대 패턴수를 지정합니다.

비트패턴의 종료시점은 보간동작축의 2bit 펄스패턴중 단 하나라도 11의 패턴이 발생하였거나 bitPatternNo로 지정된 값의 패턴만큼 펄스를 출력하게 되면 종료합니다. 비트패턴 11위치와 bitPatternNo의 위치가 서로 다르면 비트패턴 11의 값이 우선적으로 종료시점으로 결정함으로써 속도의 절단현상이 발생할 수 있습니다. 그러므로 BPN의 위치와 비트패턴 11의 위치는 정확해야 합니다.

※ bitPatternData[][8]의 8 은 위칸 모션컨트롤러(WIMC_)의 지원 축수입니다. WIMC_라이브러리는 모션컨트롤 제품의 실제 동작축수에 관계없이 항상 배열크기 8 을 사용합니다.

27.3.1. WIMC_Set_GIOPattern

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Set_GIOPattern
(
             WIMC_INT32              bitSize,
             WIMC_INT8               patternBuf[]
);
```

➲ Description

사용자 비트 패턴을 모션동작(가장많이 펄스를 발생하는 축)과 동기하여 출력 포트(GOUT0)로 출력합니다.
예를들어 x축 50펄스, Y축 100펄스이면 Y축이 보간 주축으로 동기됩니다.
1펄스 출력당 패턴 1비트 출력됩니다.

➲ Parameters

bitsSize
패턴 비트 수(최대 100개)

patternBuf[]
On/Off 패턴 설정 버퍼

➲ Return Value

Return code/value	Description

➲ Example

```
//-----------------------------------------------------------------------
//      패턴 출력 예시
//      GOUT 포트 0 에 출력 됩니다.
//-----------------------------------------------------------------------
WIMC_Enter_QueueCmd();

// 속도설정
WIMC_Set_VelMode( WIMC_VELMODE_LINEAR);
WIMC_Set_DriveVel( 10);
WIMC_Set_StartVel( 1);
WIMC_Set_Acc( 5000);
WIMC_Set_Dec( 5000);

// 이동량설정
WIMC_INT32                      pulse[8] = { 0, };
pulse[WIMC_AXISNO_X] = 80;
```

```
//------------------------------------------------------------------------
//      패턴 설정
//------------------------------------------------------------------------
#define MAX_BUF          100

WIMC_UINT8       pattern_buf[MAX_BUF] = {0,};
UINT16           on_pattern_bits = 0;
UINT16           off_pattern_bits = 0;
UINT16           total_pattern_bits = 0;

if (1)
{
        on_pattern_bits = 2;
        off_pattern_bits = 14;
        total_pattern_bits = on_pattern_bits + off_pattern_bits;

        // ON 패턴 설정
        int              i = 0;
        int              byte_cnt = 0;
        for ( i= 0; i< on_pattern_bits; i++)
        {
                byte_cnt = i / 8;
                pattern_buf[byte_cnt] |= 1 << (i % 8);
                printf("byte_idx(%d), bits(%d), val(%02x)\n", byte_cnt, (i%8),
pattern_buf[byte_cnt]);
        }

        // OFF 패턴 설정
        for ( i = on_pattern_bits; i< on_pattern_bits + off_pattern_bits; i++)
        {
                byte_cnt = i / 8;
                pattern_buf[byte_cnt] &= ~( 1 << (i % 8));
                printf("byte_idx(%d), bits(%d), val(%02x)\n", byte_cnt, (i%8),
pattern_buf[byte_cnt]);
        }

        WIMC_Set_GIOPattern( total_pattern_bits, pattern_buf);
}
else
{
        pattern_buf[0] = 0x01;
        pattern_buf[1] = 0x00;

        WIMC_Set_GIOPattern( 16, pattern_buf);
}
```

```
//------------------------------------------------------------------------
//      라인명령
//------------------------------------------------------------------------
WIMC_Line( WIMC_X, pulse);
WIMC_Line( WIMC_X, pulse);
WIMC_Line( WIMC_X, pulse);
WIMC_Line( WIMC_X, pulse);

//------------------------------------------------------------------------
//      최종 출력 상태 설정
//------------------------------------------------------------------------
WIMC_Clr_GIOPattern(WIMC_LEVEL_L);

//------------------------------------------------------------------------
//
//------------------------------------------------------------------------
WIMC_Leave_QueueCmd();
```

➲ See Also

27.3.2. WIMC_Clr_GIOPattern

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Clr_GIOPattern
(
             WIMC_BOOL               end_level
);
```

➲ Description

비트패턴 출력 종료 시 최종 출력 LEVEL을 설정합니다.

➲ Parameters

end_level

최종 출력 Level 설정

WIMC_LEVEL_H(1) : high level로 설정합니다.

WIMC_LEVEL_L(0) : Low level로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

```
//-----------------------------------------------------------------------
//      패턴 출력 예시
//      GOUT 포트 0 에 출력 됩니다.
//-----------------------------------------------------------------------
WIMC_Enter_QueueCmd();

// 속도설정
WIMC_Set_VelMode( WIMC_VELMODE_LINEAR);
WIMC_Set_DriveVel( 10);
WIMC_Set_StartVel( 1);
WIMC_Set_Acc( 5000);
WIMC_Set_Dec( 5000);

// 이동량설정
WIMC_INT32                      pulse[8] = { 0, };
pulse[WIMC_AXISNO_X] = 80;

//-----------------------------------------------------------------------
```

```
//      패턴 설정
//-------------------------------------------------------------------
#define MAX_BUF          100

WIMC_UINT8       pattern_buf[MAX_BUF] = {0,};
UINT16           on_pattern_bits = 0;
UINT16           off_pattern_bits = 0;
UINT16           total_pattern_bits = 0;

if (1)
{
        on_pattern_bits = 2;
        off_pattern_bits = 14;
        total_pattern_bits = on_pattern_bits + off_pattern_bits;

        // ON 패턴 설정
        int              i = 0;
        int              byte_cnt = 0;
        for ( i= 0; i< on_pattern_bits; i++)
        {
                byte_cnt = i / 8;
                pattern_buf[byte_cnt] |= 1 << (i % 8);
                printf("byte_idx(%d), bits(%d), val(%02x)\n", byte_cnt, (i%8),
pattern_buf[byte_cnt]);
        }

        // OFF 패턴 설정
        for ( i = on_pattern_bits; i< on_pattern_bits + off_pattern_bits; i++)
        {
                byte_cnt = i / 8;
                pattern_buf[byte_cnt] &= ~( 1 << (i % 8));
                printf("byte_idx(%d), bits(%d), val(%02x)\n", byte_cnt, (i%8),
pattern_buf[byte_cnt]);
        }

        WIMC_Set_GIOPattern( total_pattern_bits, pattern_buf);
}
else
{
        pattern_buf[0] = 0x01;
        pattern_buf[1] = 0x00;

        WIMC_Set_GIOPattern( 16, pattern_buf);
}


//-------------------------------------------------------------------
//      라인명령
```

```
//-------------------------------------------------------------------
WIMC_Line( WIMC_X, pulse);
WIMC_Line( WIMC_X, pulse);
WIMC_Line( WIMC_X, pulse);
WIMC_Line( WIMC_X, pulse);

//-------------------------------------------------------------------
//      최종 출력 상태 설정
//-------------------------------------------------------------------
WIMC_Clr_GIOPattern(WIMC_LEVEL_L);

//-------------------------------------------------------------------
//
//-------------------------------------------------------------------
WIMC_Leave_QueueCmd();
```

➲ See Also

27.4. 원점복귀

27.4.1. WIMC_Home

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Home
(
              WIMC_UINT8              axis,
              WIMC_INT32              pulseMax
);
```

➲ Description

주모션엔진을 사용하는 보간동작방식의 원점복귀명령입니다.
원점복귀동작에 필요한 최대한의 펄스수 pulseMax 값을 입력합니다.
원점복귀 명령은 원점센서를 발견할 때까지 계속적으로 동작할 수 있습니다.
이때 최대펄스값을 지정하며 원점복귀 명령으로 이동하는 축의 원점복귀동작 한계거리를 설정합니다. 이 설정거리 이내에 원점복귀 동작을 완료하지 않는다면 원점복귀동작은 강제로 완료되며 원점복귀알람을 발생합니다.

➲ Parameters

axis
적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulseMax
원점복귀동작에 필요한 최대한의 펄스수입니다.

➲ Return Value

Return code/value	Description

➲ Example

```
WIMC_Enter_QueueCmd();

// 원점복귀모드설정
WIMC_Set_HomeMode(WIMC_X, WIMC_HOMEMODE_0); // EncZ 사용안함
WIMC_Set_HomeMode(WIMC_Y, WIMC_HOMEMODE_0); // EncZ 사용안함
WIMC_Set_HomeMode(WIMC_Z, WIMC_HOMEMODE_0); // EncZ 사용안함

// 속도모드설정
WIMC_Set_VelMode(WIMC_VELMODE_LINEAR);

// 전체속도배율설정
```

```
WIMC_Set_VelMultiplier(4);

// 원점복귀 속도
WIMC_Set_DriveVelAxis(WIMC_X, 1000);
WIMC_Set_DriveVelAxis(WIMC_Y, 1000);
WIMC_Set_DriveVelAxis(WIMC_Z, 1000);

// 원점복귀 가속도/감속도
WIMC_Set_Acc( 5000);
WIMC_Set_Dec( 5000);

// 원점복귀 탐색속도
WIMC_Set_HomeSeekVel(WIMC_X, 100);
WIMC_Set_HomeSeekVel(WIMC_Y, 100);
WIMC_Set_HomeSeekVel(WIMC_Z, 100);


// 보간동작 원점복귀 (단축동작만 가능)
WIMC_Home(WIMC_X, 100000);
WIMC_Sync();                                // 이전명령이 완전히 종료된 후 다음명령을 처리하도록 지시

WIMC_Home(WIMC_Y, 100000);
WIMC_Sync();                                // 이전명령이 완전히 종료된 후 다음명령을 처리하도록 지시

WIMC_Home(WIMC_Z, 100000); WIMC_Sync();

WIMC_Leave_QueueCmd();
```

➲ See Also

28. 다중보간

28.1. 다중보간(Multiple Interpolation)

하나의 보간동작만이 실행되는 것을 단일보간동작이라 합니다.
다중보간은 서로 다른 2개의 보간명령이 서로 다른 축공간상에서 동시에 수행되는 것을 말합니다.
이때 두 보간은 동일한 시점에서 시작해서 동일한 시점에 종료됩니다.
다중보간과 단일보간의 전환시에는 연속실행이 이루어지지 않습니다. 그 반대의 전환에서도 연속실행은 이루어지지 않습니다.
다중보간은 단일보간동작명령들이 완전히 종료된 후 다중보간 실행모드로 전환한 후 보간명령을 전송해야 합니다. 다중보간 실행이 완료되면 다시 단일보간 실행모드로 바꾸어야 합니다. 물론 다중보간 명령들 간의 연속실행은 허용됩니다. 다중보간 실행모드에서도 단일보간 동작이 가능합니다. 단 단일보간실행모드일 때와 다르게 보간명령후에 다중보간실행명령인 MI_EXE 명령을 전송해야만 합니다. 그러므로 단일보간명령을 다중보간명령으로 실행할 때는 약간의 비용이 발생하게 됩니다.

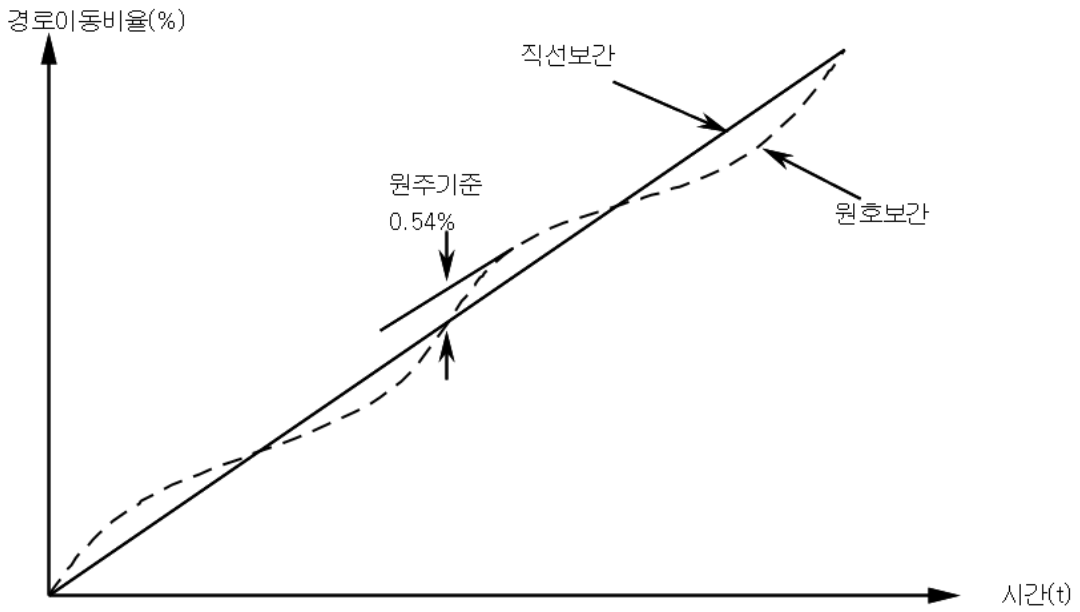
다중보간 실행모드에서 다중보간실행 명령 순서
(1) 다중보간 시작명령
- WIMC_Enable_MI(TRUE) 명령을 다중보간 실행모드로 전환합니다.
(2) 제 1 보간동작명령
- Line, ArcCW/CCW, BitPattern 등의 첫 번째 보간명령을 실행합니다.
- 이 명령은 MI_EXE(다중보간실행명령)이 오기까지 실행보류 됩니다.
(3) 제 2 보간동작명령
- Line, ArcCW/CCW, BitPattern 등의 두 번째 보간명령을 실행합니다.
- 이 명령은 MI_EXE(다중보간실행명령)이 오기까지 실행보류 됩니다.
(4) 다중보간 실행명령
- WIMC_MIExe() 명령을 실행하면 첫 번째 보간명령과 두 번째 보간명령이 동시에 시작됩니다.
(5) 다음 다중보간이 있으면 (2)~(4)까지 반복
(6) 다중보간 OFF 명령
- WIMC_Enable_MI(FALSE) 명령으로 다중보간 동작이 모두 완료하면 단일보간 실행모드로 전환합니다.

다중보간 실행모드에서 단일보간실행 명령
다중보간 실행 명령순서에서 (3) 제 2 보간동작명령 없이 바로 다중보간실행명령(MI EXE)을 전송하면 단일보간실행과 동일하게 명령을 실행합니다.

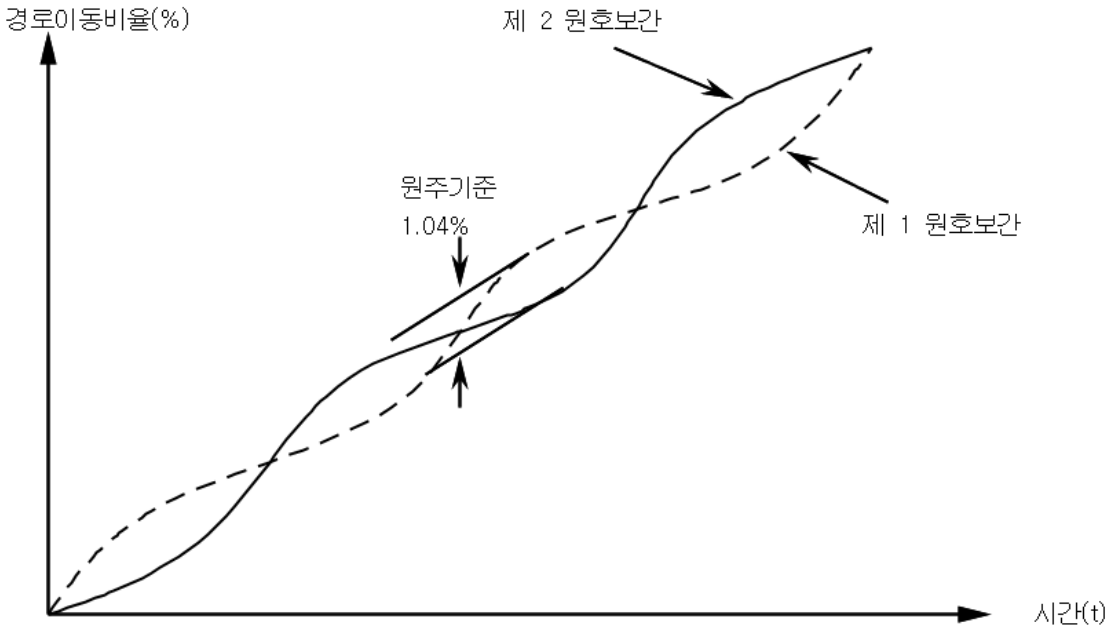
다중보간 유의점
다중보간은 서로 다른 2개의 보간 동작이 서로 다른 축공간에서 이루어지는 것이 바람직합니다. 같은 축을 공유하게 되면 펄스는 OR된 출력결과를 갖게 됩니다. 서로 다른 직선보간을 다중보간으로 동작시킬 필요는 없습니다. 이것은 오히려 WIMC 엔진이 출력할 수 있는 동작 최대 주파수를 낮추게 됩니다. 다중보간은 단일보간에 비해 80%정도내에서 사용되어야 안전합니다.

다중보간에서 상호 선형성(Mutual Linearity) 검토
직선보간과 원호보간이 다중보간으로 동작할 때 상호 선형성은 π/8(45도)마다 최대 4.3%의 오차를 갖습니다. 그러므로 완전한 원일 경우 원주에 대해서 0.54%(4.3%/8)의 오차가 발생합니다. 상호 선형성과 관계없이 동기 다중보간에서 시작과 종료시각은 일치합니다.
상호 선형성이란 임의의 두 보간이 얼마나 동일한 비율로 이동하는가를 나타냅니다. 아래의 그림을 통해서 이해해 봅니다.

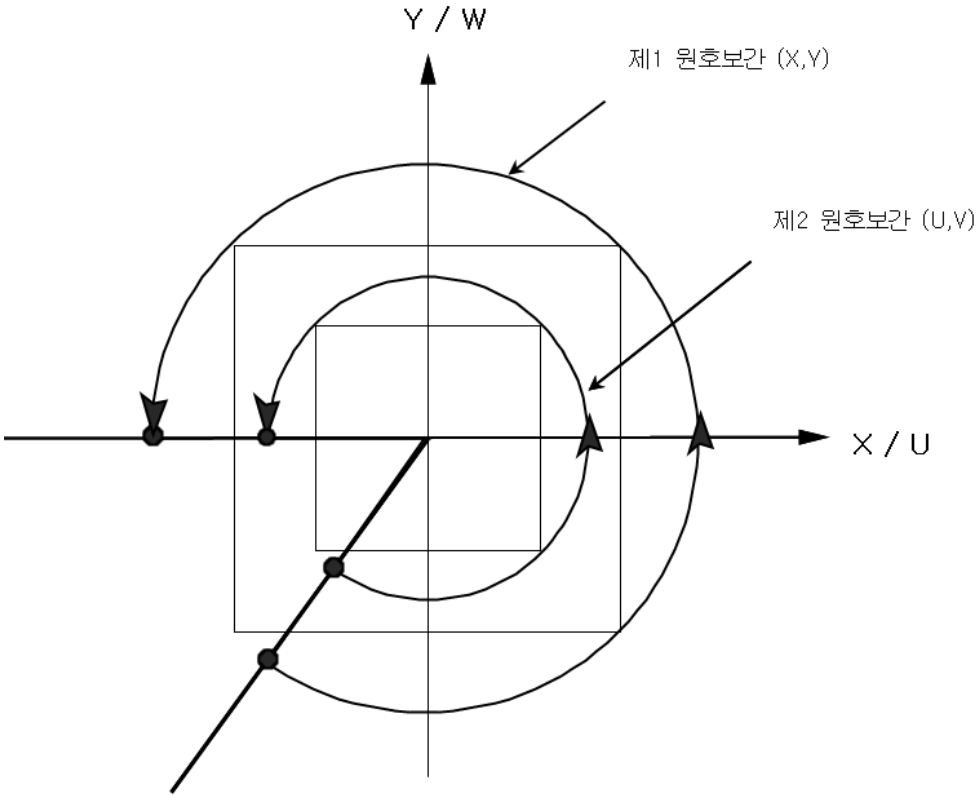
<직선보간과 원호보간의 상호선형성 오차>



<서로 다른 원호보간에서 상호선형성 최대 오차>



서로 다른 두 원의 시작각도가 45도씩 어긋났다면 최대 1.04%의 상호선형오차가 발생됩니다. 그러나 다음과 같이 동일한 위상의 같는 원호보간 명령에서는 상호간에 완전한 선형성을 유지할 수 있습니다.


Y / W
제1 원호보간 (X,Y)
제2 원호보간 (U,V)
X / U

28.1.1. WIMC_Enable_MI

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Enable_MI
(
              WIMC_BOOL                 bEnable
);
```

➲ Description

한번에 2개의 보간동작이 동시에 시작하고 동시에 종료하는 다중보간명령실행을 설정(Enable/Disable)합니다.

관련 내용 및 설명은 (28.1절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

28.1.2. WIMC_Is_MI

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_MI( void);
```

➲ Description

모션엔진의 다중보간동작 설정(Enable/Disable)을 읽습니다.

관련 내용 및 설명은 (28.1절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	다중보간동작 상태입니다.
WIMC_FALSE	다중보간동작 상태가 아닙니다.

➲ Example

➲ See Also

28.1.3. WIMC_MIExe

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_MIExe( void);
```

➲ Description

다중보간 실행명령입니다.

관련 내용 및 설명은 (28.1절)을 참고 하실수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

29. 개별축동작명령

29.1. 독립이송

29.1.1. WIMC_IMove

➲ Syntax

```
WIMC_BOOL     WIMC_API              WIMC_IMove
(
              WIMC_UINT8            axis,
              WIMC_INT32            pulse[]
);
```

➲ Description

특정축의 이송 위치(pulse[])로 독립이송을 실행합니다.

독립이송 명령은 개별축제어 혹은 독립축제어를 위한 보조모션엔진에 의해 실행됩니다. WIMC 엔진은 보조모션엔진을 4개를 보유하고 있으므로 동시에 4축에 대한 독립이송명령을 실행할 수 있습니다. 독립이송은 보간동작이 실행 중일 때에도 바로 실행되므로 보간동작과 축을 공유하지 않는 것이 바람직합니다. 만약 이전 동작명령이 실행중인 축과 공유된 축동작을 실행시키려 한다면 해당 축의 동작은 대기상태에 놓이며 이전 동작명령이 완료된 후에 실행하게 됩니다.

X6V3 엔진의 경우는 총 6개의 축을 지원하지만 보조모션엔진은 4개 밖에 없기 때문에 만약 6개의 축을 동시에 독립이송명령을 실행하게 되면 X,Y,Z,U의 4개의 축이 동시에 실행되고 나머지 V축과 S1축을 대기상태에 놓입니다. 이전에 실행된 4개의 보조모션엔진 중 먼저 실행이 완료된 엔진이 실행대기 중인 다른 축을 찾아서 자동으로 실행을 이어갑니다.
그러므로 6개 축을 동시에 독립이송 명령을 사용하면 4축만 먼저 실행되고 나머지 2축은 지연되어 실행되게 됩니다.

IMOVE 명령은 단일 축에 대한 직선보간명령과 동일하지만 주모션엔진으로 동작하지 않기 때문에 별도의 속도 및 가속도 파라메터를 사용합니다. 보조모션엔진을 사용하는 명령들은 연속실행이 허용되지 않습니다.

독립이송명령은 현재 주동작인 보간동작이 실행중일 때, 동작하지 않는 다른 축을 이용하여 보조적인 장치들의 동작을 실행할 때 유용하게 사용할 수 있습니다. 이들 보조적인 장치는 재료자동공급장치(Feeder), 자동공구교환장치(ATC:Auto Tool Changer), 로딩/언로딩장치(Loader/Unloader)등에 활용될 수 있습니다.

※ IMOVE 와 같은 개별축명령이 보간동작과 동시에 수행될 때 동작최대주파수가 감소됩니다. 모니터링프로그램을 통해서 안전한 주파수 영역에서 사용해야합니다. WIMC 엔진은 전체 동작주파수 합이 100KPPS 를 넘지 않는 것이 바람직합니다.

➲ Parameters

axis
적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulse[]

독립 이송 위치(펄스)입니다.
WIMC_라이브러리는 모션컨트롤 제품의 실제 동작축수에 관계없이 항상 배열크기 8을 사용합니다.배열내의 요소는 X,Y,Z,U,V,W,S1,S2축 입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

29.2. 연속이송

29.2.1. WIMC_CMove

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_CMove
(
              WIMC_UINT8              axis,
              WIMC_INT32              pulseMax[]
);
```

➲ Description

특정축의 위치값(pos[])으로 연속 이송 명령을 실행합니다.

연속이송명령도 독립이송명령과 같이 보조모션엔진에 의해 동작합니다. 연속이송(CMOVE)명령은 지정된 축을 연속해서 회전시키는 명령입니다. 이 명령은 스핀들의 회전명령처럼 단순한 회전과는 달리 현재위치값을 변화시키며 스캔타임에 영향을 줍니다. 그러므로 높은 주파수로 회전시킬 경우 다른 모션동작의 허용동작 주파수가 낮아질 수 있습니다.
연속이송동작을 정지시키려면 개별축 구동속도값을 0으로 설정해야 합니다. 반대로 개별축 구동속도값이 0인 축에 연속이송명령을 전송하면 아무런 동작이 발생하지 않습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulseMax[]

해당 축의 목표(최대) 이송 위치값으로 1보다 크면 CW 회전방향의 연속이송명령이며, -1보다 작으면 CCW 회전방향의 연속이송명령이 됩니다.
WIMC_라이브러리는 모션컨트롤 제품의 실제 동작축수에 관계없이 항상 배열크기 8을 사용합니다.배열내의 요소는 X,Y,Z,U,V,W,S1,S2축 입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

29.3. 조건이송

29.3.1. WIMC_XMove

➲ Syntax

```
WIMC_BOOL    WIMC_API              WIMC_XMove
(
             WIMC_UINT8            axis,
             WIMC_INT32            pulse[],
             WIMC_UINT32           decCond,
             WIMC_UINT32           stopCond,
             WIMC_INT32            decVel,
             WIMC_INT32            decPos
);
```

➲ Description

조건이송기능을 실행합니다.

❶ 조건이송명령

조건이동명령 역시 보조모션엔진을 사용하는 명령입니다. 조건이동 명령은 특정조건에서 감속, 혹은 정지할 수 있는 명령입니다. 특정조건에는 (1)감속조건과 (2)정지조건이 있습니다.
이들 조건은 원점센서변화, 엔코더Z상변화, 특정 GIO입력, 축위치값 조건 등이 있습니다.

감속조건의 경우 감속할 속도와 감속할 위치를 지정할 수 있습니다. 만약 감속조건속도를 0으로 설정되었다면 감속조건이 만족하여 감속동작이 이루어질 때 현재속도가 0에 도달하면 명령을 종료하게 됩니다.

이들 조건들은 32비트 부호없는 정수값의 조건플래그에 설정됩니다.
이 조건플래그의 내용들을 아래에 정리해 봅니다.

감속 및 정지 조건플래그 32비트

비트 번호	31	30	29	28	27	26	25	24
조건	홈센서 감지	엔코더Z상 변화감지	감속위치 도달	-	Servo UserIn 신호감지	Servo Alarm 신호감지	Servo InPos 신호감지	Servo Ready 신호감지
비트 번호	23 ~ 16							
조건	사용안함							
비트 번호	15 ~ 0							
조건	GIO의 입력포트 0~15 중 조건플래그의 설정값이 1인 비트에 해당하는 신호값이 1 이면 감속							

각 비트별 해당 조건을 활성화하려면 1로 설정합니다.
여러 개의 조건이 동시에 활성화되면 논리 OR된 조건으로 동작됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulse[]

조건이송명령 목표(최대) 이동 위치입니다.

decCond/ stopCond

감속/정지조건 32비트 플래그입니다.

아래는 감속 및 정지 조건 매크로명입니다.

```
WIMC_XMOVE_COND_HOMESENSOR      // 홈센서감지시 감속/정지
WIMC_XMOVE_COND_ENCZ            // 엔코더Z상 변화감지시 감속/정지
WIMC_XMOVE_COND_POS             // decPos로 설정된 위치에서 감속/정지
WIMC_XMOVE_COND_SV_READY        // Sv.Ready신호 감지시 감속/정지
WIMC_XMOVE_COND_SV_INPOS        // Sv.Inpos신호 감지시 감속/정지
WIMC_XMOVE_COND_SV_ALARM        // Sv.Alarm신호 감지시 감속/정지
WIMC_XMOVE_COND_SV_USERIN       // Sv.UserIn신호 감지시 감속/정지
WIMC_XMOVE_COND_GIN(io)         //16비트io로 지정된 Gin포트값이 1이 감지되면 감속/정지
WIMC_XMOVE_COND_GIN_0           //Gin 0번 포트값이 1이 감지되면 감속/정지
…
WIMC_XMOVE_DEC_COND_GIN_15      //Gin 15번 포트값이 1이 감지되면 감속/정지
```

decVel

감속조건속도 16비트 부호없는 정수입니다.

decPos

강제 감속위치입니다. 32비트 부호있는 정수입니다.

➲ Return Value

Return code/value	Description

➲ Example

```
// 즉시실행명령으로 실행
WIMC_Enter_ImCmd();

// 속도모드설정
WIMC_Set_VelModeAxis(WIMC_X, WIMC_VELMODE_LINEAR);

// 전체속도배율설정
WIMC_Set_VelMultiplier(4);

// 원점복귀 가속도/감속도
```

```
WIMC_Set_AccAxis( WIMC_X, 1000);
WIMC_Set_DecAxis( WIMC_X, 1000);

WIMC_Set_DriveVelAxis( WIMC_X, 5000);

// 개별축명령 원점복귀 (최대 4축 동시가능)
WIMC_INT32 pulse[WIMC_AXIS_MAX]={300000,300000,300000,};

WIMC_UINT32 decCond = 0;
WIMC_UINT32 stopCond = 0;
WIMC_INT32 decPos = 0;
WIMC_INT32 decVel = 0;

// 특정위치(현재위치레지스터기준)에서 감속시작
decCond = WIMC_XMOVE_DECCOND_POS;
decPos = 100000;

// GIO 입력 4의 반전신호에 의해 정지
stopCond = WIMC_XMOVE_STOPCOND_SYNC | WIMC_XMOVE_STOPCOND_INV | WIMC_XMOVE_STOPCOND_DECSTOP;
decVel =  1000;
// 조건이동명령
WIMC_XMove(WIMC_X, pulse, decCond, stopCond, decVel, decPos);

// 즉시실행명령종료
WIMC_Leave_ImCmd();
```

➲ See Also

29.4. 스핀들회전

29.4.1. WIMC_Spindle

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Spindle
(
              WIMC_UINT8              axis,
              WIMC_INT32              pps
);
```

➲ Description

핀들 회전 명령을 실행합니다.
axis는 WIMC_S1으로 고정됩니다.

❶ 스핀들 회전명령

스핀들 회전명령은 S1축 혹은 S2축에만 지정될 수 있으며 최대 1MHz까지 출력될 수 있습니다.
스핀들 회전명령은 주모션엔진과 보조모션엔진을 사용하지 않고 스핀들전용모션엔진을 사용합니다.
스핀들전용모션엔진에 의한 펄스는 하드웨어로직에 의해 발생되므로 실행부하를 주지 않아 스캔타임에 영향이 없습니다. 스핀들 회전명령은 일반적인 설정명령이나 읽기/쓰기명령과 같이 바로 처리되기 때문 동작완료상태정보를 전송하지 않습니다.

스핀들 회전명령으로 작동되는 축은 위치제어모드에서 속도제어모드로 바뀌게 되며 현재위치도 변화하지 않게 됩니다.
스핀들 회전명령에 의한 회전속도 역시 개별 축 속도파라메터에 의해 설정됩니다.
스핀들 회전명령은 고정된 x16속도 배율에 적용받습니다.
스핀들 회전명령에 의한 회전동작의 속도모드는 일정속도모드와 직선(선형)속도모드중 하나를 선택할 수 있습니다.
스핀들 회전명령에 의한 회전축은 S-Curve속도모드를 지원하지 않습니다.

해당 축이 SPINDLE 회전명령에 의해 회전하고 있을 때 위치이동명령을 전송하게 되면 오류를 발생하게 됩니다. 그러나 자동절환기능을 활성화하게 되면 자동으로 감속정지 시킨 후 위치이동명령이 실행하게 됩니다. 대부분의 경우, 스핀들회전명령에 의해 특정속도로 회전시켰다가 다시 위치제어로 전환될 때에는 축을 정지시키고 원점복귀명령에 의해 원점위치로 이동한 후에 위치제어를 수행하는 것이 바람직합니다.

➲ Parameters

axis

WIMC_S1(으로 고정됩니다.)

pps

회전 속도값입니다.
값이 1000이면 CW 회전방향으로 16*1000 PPS로 회전합니다.
값이 -1000이면 CCW 회전방향으로 16*1000 PPS로 회전합니다.
값이 0이면 해당축은 정지합니다.
단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

29.5. 원점복귀

29.5.1. WIMC_IHome

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_IHome
(
              WIMC_UINT8              axis,
              WIMC_INT32              pulseMax[]
);
```

➲ Description

보조모션엔진을 사용하는 개별축동작방식의 원점복귀명령입니다.
원점복귀동작에 필요한 최대한의 펄스수 pulseMax 값을 입력합니다.

원점복귀 명령은 원점센서를 발견할 때까지 계속적으로 동작할 수 있습니다.
이때 최대펄스값을 지정하며 원점복귀 명령으로 이동하는 축의 원점복귀동작 한계거리를 설정합니다. 이 설정거리 이내에 원점복귀 동작을 완료하지 않는다면 원점복귀동작은 강제로 완료되며 원점복귀알람을 발생합니다.

➲ Parameters

axis
적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulseMax
원점복귀동작에 필요한 최대한의 펄스수입니다.

➲ Return Value

Return code/value	Description

➲ Example

```
WIMC_Enter_QueueCmd();

// 원점복귀모드설정
WIMC_Set_HomeMode(WIMC_X, WIMC_HOMEMODE_0); // EncZ 사용안함
WIMC_Set_HomeMode(WIMC_Y, WIMC_HOMEMODE_0); // EncZ 사용안함
WIMC_Set_HomeMode(WIMC_Z, WIMC_HOMEMODE_0); // EncZ 사용안함

// 속도모드설정
WIMC_Set_VelModeAxis(WIMC_X, WIMC_VELMODE_LINEAR);
```

```
WIMC_Set_VelModeAxis(WIMC_Y, WIMC_VELMODE_LINEAR);
WIMC_Set_VelModeAxis(WIMC_Z, WIMC_VELMODE_LINEAR);

// 전체속도배율설정
WIMC_Set_VelMultiplier(4);

// 원점복귀 속도
WIMC_Set_DriveVelAxis(WIMC_X, 1000);
WIMC_Set_DriveVelAxis(WIMC_Y, 1000);
WIMC_Set_DriveVelAxis(WIMC_Z, 1000);

// 원점복귀 가속도/감속도
WIMC_Set_AccAxis( WIMC_X, 5000);
WIMC_Set_AccAxis( WIMC_Y, 5000);
WIMC_Set_AccAxis( WIMC_Z, 5000);
WIMC_Set_DecAxis( WIMC_X, 5000);
WIMC_Set_DecAxis( WIMC_Y, 5000);
WIMC_Set_DecAxis( WIMC_Z, 5000);

// 원점복귀 탐색속도
WIMC_Set_HomeSeekVel(WIMC_X, 100);
WIMC_Set_HomeSeekVel(WIMC_Y, 100);
WIMC_Set_HomeSeekVel(WIMC_Z, 100);

// 개별축명령 원점복귀 (최대 4축 동시가능)
WIMC_INT32 maxPulse[WIMC_AXIS_MAX]={100000,100000,100000,};
WIMC_IHome(WIMC_X | WIMC_Y | WIMC_Z, maxPulse);

WIMC_Leave_QueueCmd();
```

➲ See Also

30. 원점복귀 옵션

30.1. 원점복귀

원점복귀 동작모드

WIMC 엔진은 기본적으로 아래의 3가지 방식의 원점복귀동작을 지원합니다.
(1) 모드 0 = 원점센서 회귀정지형
- 원점복귀 이동시 원점센서신호(혹은 홈센서신호)를 발견하면 감속정지하고 다시 역방향으로 회귀동작하면서 원점센서검출 후에 정지합니다.
(2) 모드 1 = 엔코더 Z상신호를 이용한 원점센서 회귀정지형
- 원점센서 회귀정지형과 동일한 방식이지만 마지막 정지시 원점센서검출 후 엔코더Z상의 변화가 발견될 때 정지합니다.
(3) 모드 2 = 원점센서 즉시 정지형
- 원점복귀방향으로 천천히 이동하다 원점센서신호를 발견하면 바로 정지합니다. 주로 스핀들의 원점복귀와 같이 역방향 동작을 실행하지 않아야만 할 경우에 사용됩니다.

<관련 라이브러리함수>
WIMC_Set_HomeMode(WIMC_HOMEMODE_0); // 원점복귀동작을 모드 0 으로 설정합니다.

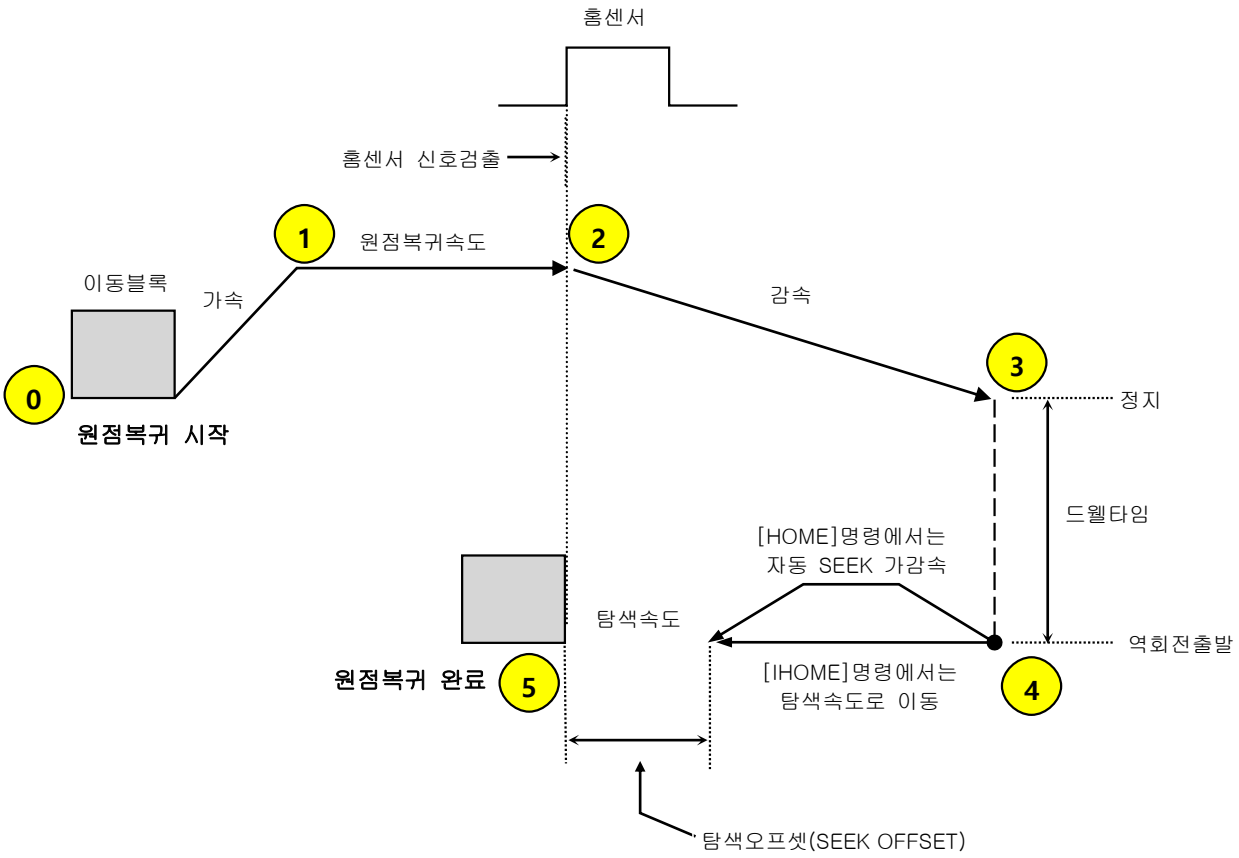
원점복귀 동작 옵션

전체적인 동작방식 이외에도 원점복귀를 위해서 다양한 동작옵션설정을 갖고 있습니다. 아래에 원점복귀 동작을 위한 옵션들을 살펴봅니다.
(1) 원점복귀 속도설정
(2) 원점복귀 방향설정
(3) 홈센서 감지레벨 설정
(4) 홈센서 탐색 속도설정
(5) 회귀동작 드웰타임 설정
(6) 자동 영점설정 기능
(7) 원점복귀 동작 오류감지 기능
(8) 원점복귀 완료 후 SERVO RESET신호 발생기능
(9) 홈센서 탐색 오프셋 펄스수 설정 (주모션엔진의 [HOME]명령에만 설정)

※ 원점복귀동작의 속도모드는 일반적으로 1 차 선형가감속 속도모드를 사용합니다. 일정속도모드는 정지시 충격이 크며, S 자 가감속모드는 홈센서발견시 긴 정지거리를 필요로 하기 때문에 선형가감속모드로써 원점복귀동작을 실행하는 것이 바람직합니다.

원점복귀 동작 방식

▶ 동작모드 0 = 원점센서 회귀정지형



- 동작시퀀스-

① 원점복귀속도로 원점복귀 방향을 향해 이동하며 홈센서(원점센서)를 찾습니다.

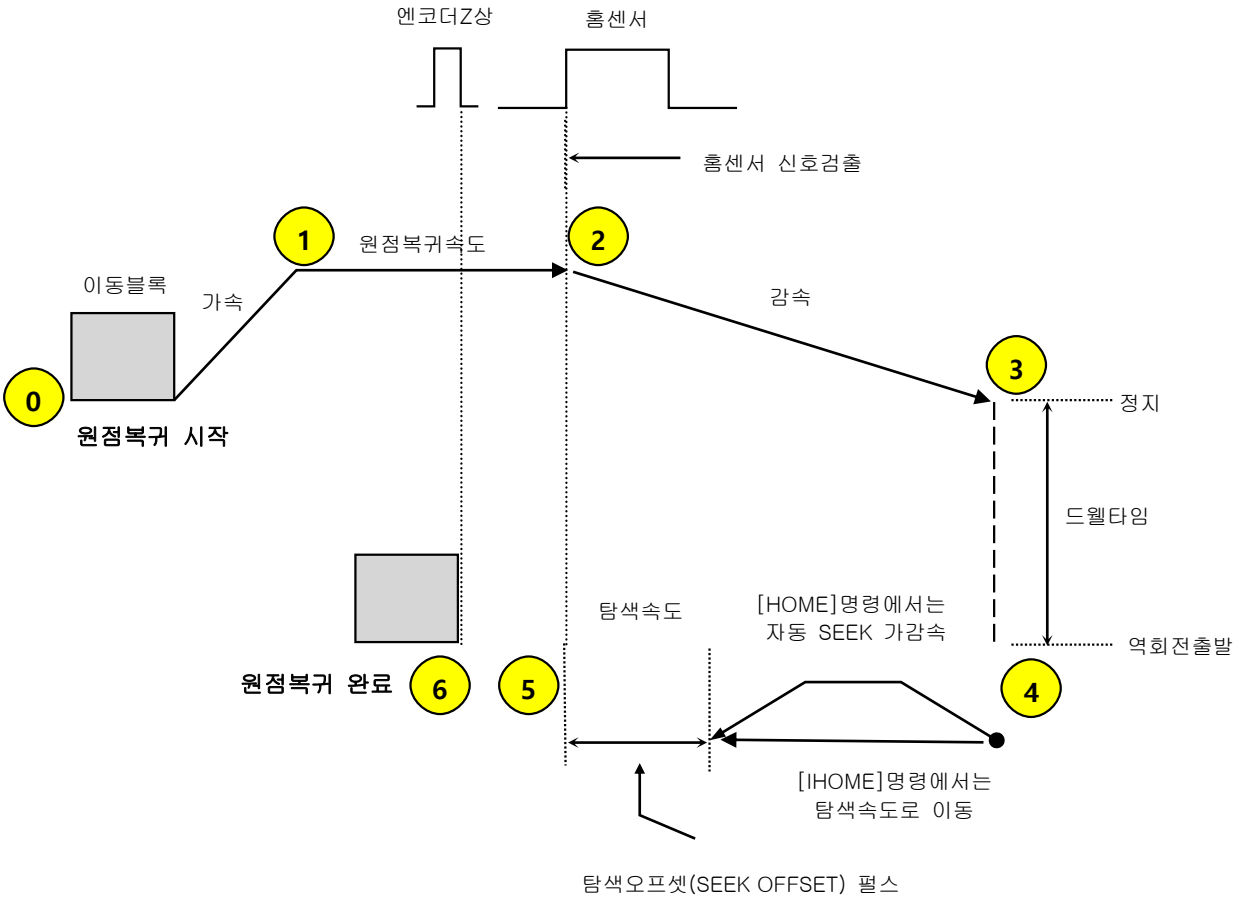
② 홈센서신호를 발견하면 감속하여 정지합니다.

③ 설정된 드웰타임만큼 정지상태를 유지합니다.

④ 홈센서 재탐색을 위해 회귀동작을 실행합니다.

⑤ 홈센서신호를 발견하고 홈센서신호가 다시 사라질 때 정지하면서 원점복귀동작을 완료합니다.

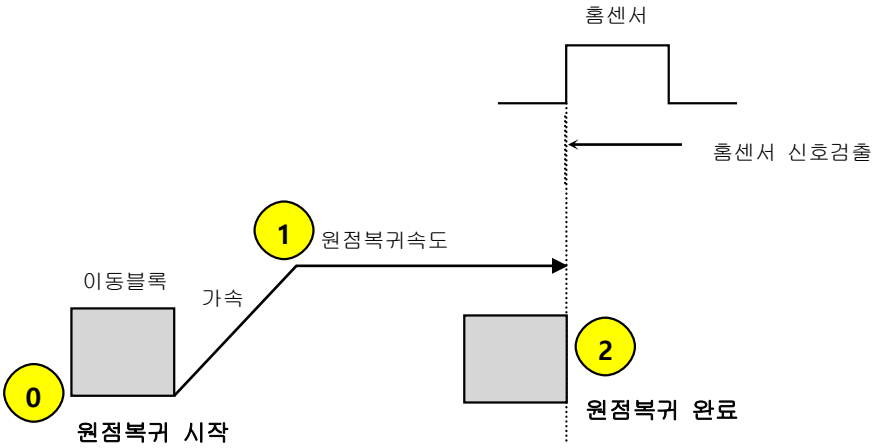
▶ 동작 모드 1 = 엔코더 Z상을 이용한 원점센서 회귀정지형



- 동작시퀀스-

① 설정된 원점복귀 방향으로 구동속도를 원점복귀속도로 하여 홈센서(원점센서)를 찾아갑니다.
② 홈센서신호를 발견하면 설정된 속도모드로 감속하여 정지합니다.
③ 설정된 드웰타임만큼 정지상태를 유지합니다.
④ 홈센서 재탐색을 위해 회귀동작을 실행합니다.
⑤ 홈센서신호를 발견하고 홈센서신호가 다시 사라질 때부터 엔코더z상신호의 변화를 찾습니다.
⑥ 엔코더z상의 신호변화를 감지하면 즉시 정지하고 원점복귀동작을 완료합니다.

▶ 동작모드 2 = 원점센서 즉시 정지형



- 동작시퀀스-

① 설정된 원점복귀 방향으로 구동속도를 원점복귀속도로 하여 홈센서(원점센서)를 찾아갑니다.

② 홈센서신호를 발견하면 즉시 정지하면서 원점복귀동작을 완료합니다.

엔코더Z신호를 사용할 경우 엔코더Z신호와 홈센서신호를 연결하여 사용할 수 있습니다.

< 관련 라이브러리 함수 >

```
WIMC_Set_HomeMode(axis, WIMC_HOMEMODE_0); // 원점복귀방식설정
WIMC_Home(axis, pulseMax); // 주모션엔진을 사용하는 보간동작방식의 원점복귀명령
WIMC_IHome(axis, pulseMax); // 보조모션엔진을 사용하는 개별축동작방식의 원점복귀명령
// pulseMax값은 원점복귀동작에 필요한 최대한의 펄스수를 입력합니다.
// 원점복귀 명령은 원점센서를 발견할 때까지 계속적으로 동작할 수 있습니다.
// 이때 최대펄스값을 지정하며 원점복귀 명령으로 이동하는 축의 원점복귀동작 한계거리를 설정합니다.
// 이 설정거리이내에 원점복귀 동작을 완료하지 않는다면 원점복귀동작을 강제로 완료되며
// 원점복귀알람을 발생합니다.
```

30.1.1. WIMC_Set_HomeMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_HomeMode
(
              WIMC_UINT8              axis,
              WIMC_INT32              mode
);
```

➲ Description

특정축의 원점복귀 방식을 설정합니다.

❶ 원점복귀 동작모드

WIMC 엔진은 기본적으로 아래의 3가지 방식의 원점복귀동작을 지원합니다.

(1) 모드 0 = 원점센서 회귀정지형

- 원점복귀 이동시 원점센서신호(혹은 홈센서신호)를 발견하면 감속정지하고 다시 역방향으로 회귀동작하면서 원점센서검출 후에 정지합니다.

(2) 모드 1 = 엔코더 Z상신호를 이용한 원점센서 회귀정지형

- 원점센서 회귀정지형과 동일한 방식이지만 마지막 정지시 원점센서검출 후 엔코더Z상의 변화가 발견될 때 정지합니다.

(3) 모드 2 = 원점센서 즉시 정지형

- 원점복귀방향으로 천천히 이동하다 원점센서신호를 발견하면 바로 정지합니다. 주로 스핀들의 원점복귀와 같이 역방향 동작을 실행하지 않아야만 할 경우에 사용됩니다.

관련 내용 및 설명은 (30.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

mode

WIMC_HOMEMODE_0 :원점센서 회귀정지형입니다.

WIMC_HOMEMODE_1 :엔코더 Z상을 이용한 원점센서 회귀정지형입니다.

WIMC_HOMEMODE_2 : 원점센서 즉시 정지형입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.2. WIMC_Get_HomeMode

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_HomeMode
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 원점복귀 설정 방식을 읽습니다.

❶ 원점복귀 동작모드

WIMC 엔진은 기본적으로 아래의 3가지 방식의 원점복귀동작을 지원합니다.

(1) 모드 0 = 원점센서 회귀정지형

- 원점복귀 이동시 원점센서신호(혹은 홈센서신호)를 발견하면 감속정지하고 다시 역방향으로 회귀동작하면서 원점센서검출 후에 정지합니다.

(2) 모드 1 = 엔코더 Z상신호를 이용한 원점센서 회귀정지형

- 원점센서 회귀정지형과 동일한 방식이지만 마지막 정지시 원점센서검출 후 엔코더Z상의 변화가 발견될 때 정지합니다.

(3) 모드 2 = 원점센서 즉시 정지형

- 원점복귀방향으로 천천히 이동하다 원점센서신호를 발견하면 바로 정지합니다. 주로 스핀들의 원점복귀와 같이 역방향 동작을 실행하지 않아야만 할 경우에 사용됩니다.

관련 내용 및 설명은 (30.1절)을 참고 하실수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

원점복귀 설정 방식을 반환합니다.

Return code/value	Description
WIMC_HOMEMODE_0	원점센서 회귀정지형입니다.
WIMC_HOMEMODE_1	엔코더 Z상을 이용한 원점센서 회귀정지형입니다.
WIMC_HOMEMODE_2	원점센서 즉시 정지형입니다.

➲ Example

➲ See Also

30.1.3. WIMC_Set_HomeDev

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_HomeDev
(
              WIMC_UINT8              axis,
              WIMC_INT32              dev
);
```

➲ Description

원점복귀 편차량(Deviation)을 설정합니다.

❶ 원점복귀 편차 설정 (Home Return Alarm Deviation)

일반적으로 원점복귀 완료 후 현재좌표값은 0 이 되도록 설정하는 경우가 대부분입니다. WIMC 엔진은 모션시스템이 원점복귀동작 완료 후 현재좌표가 0 이 되도록 구성되었을 경우, 원점복귀완료 후 실제 결과가 0에서 얼마큼의 오차가 발생했는지를 조사하여 알람을 발생시킬 수 있습니다.
이때 편차량(Deviation)을 아래의 라이브러리 함수로써 설정할 수 있습니다.
오차크기는 임의적인 값을 사용할 수 없으며 정해진 값들 중 하나를 선택할 수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

dev

편차량입니다.
-1, 1, 2, 4, 8, 16, 32, 64, 128, 10, 20, 40, 80, 160, 320, 640, 1280중 하나를 선택할 수 있습니다.
-1을 선택할 경우 WIMC_Enable_HomeDevAlarm(FALSE)와 동일한 역할을 하며,다른 값을 선택할 경우 WIMC_Enable_HomeDevAlarm(TRUE)기능을 포함하게 됩니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.4. WIMC_Get_HomeDec

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_HomeDev
(
              WIMC_UINT8              axis
);
```

➲ Description

원점복귀 편차량(Deviation)을 읽습니다.

❶ 원점복귀 편차 설정 (Home Return Alarm Deviation)

일반적으로 원점복귀 완료 후 현재좌표값은 0 이 되도록 설정하는 경우가 대부분입니다. WIMC 엔진은 모션시스템이 원점복귀동작 완료 후 현재좌표가 0 이 되도록 구성되었을 경우, 원점복귀완료 후 실제 결과가 0에서 얼마큼의 오차가 발생했는지를 조사하여 알람을 발생시킬 수 있습니다.
이때 편차량(Deviation)을 아래의 라이브러리 함수로써 설정할 수 있습니다.
오차크기는 임의적인 값을 사용할 수 없으며 정해진 값들 중 하나를 선택할 수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

편차량을 반환합니다.

(-1, 1, 2, 4, 8, 16, 32, 64, 128, 10, 20, 40, 80, 160, 320, 640, 1280중 하나입니다.)

Return code/value	Description

➲ Example

➲ See Also

30.1.5. WIMC_Set_HomeDir

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_HomeDir
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bDir
);
```

➲ Description

원점복귀 방향을 설정합니다.

ʊ 원점복귀 방향 (Home Return Direction)

원점복귀 방향은 초기 원점센서를 찾아갈 때 좌표값이 작아지는 방향을 (-)방향으로 정의하고, 좌표값이 커지는 방향을 (+)방향으로 정의합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bDir

WIMC_HOMEDIR_MINUS : 현재좌표값이 작아지는 방향을 (-)방향입니다.

WIMC_HOMEDIR_PLUSE : 현재좌표값이 커지는 방향을 (+)방향입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.6. WIMC_Is_HomeDir

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeDir
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 원점복귀 방향을 확인합니다.

❶ 원점복귀 방향 (Home Return Direction)

원점복귀 방향은 초기 원점센서를 찾아갈 때 좌표값이 작아지는 방향을 (-)방향으로 정의하고, 좌표값이 커지는 방향을 (+)방향으로 정의합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_HOMEDIR_MINUS	현재좌표값이 작아지는 방향을 (-)방향입니다.
WIMC_HOMEDIR_PLUSE	현재좌표값이 커지는 방향을 (+)방향입니다.

➲ Example

➲ See Also

30.1.7. WIMC_Set_HomeSeekOffsetPulse

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_HomeSeekOffsetPulse
(
              WIMC_UINT8              axis,
              WIMC_INT32              pulse
);
```

➲ Description

특정축의 홈센서 탐색 오프셋 펄스값을 설정합니다.

홈센서 탐색 오프셋 펄스값 (SeekOffset)

원점복귀의 회귀동작에서 홈센서 off(비활성화상태)를 기준으로 감속 정지할 상대거리를 설정합니다.
이 파라메터는 홈센서탐색을 위한 회귀동작에서 가감속동작을 실행하는 보간동작형 원점복귀명령인 [HOME]에만 적용되며 회귀동작시 일정속도(원점센서탐색속도)로 움직이는 개별축동작형 원점복귀명령인 [IHOME]명령에는 적용되지 않습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pulse

감속정지할 상대거리(펄스)값입니다.(0~65535)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.8. WIMC_Get_HomeSeekOffsetPulse

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_HomeSeekOffsetPulse
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 홈센서 탐색 오프셋 펄스값을 읽습니다.

❶ 홈센서 탐색 오프셋 펄스값 (SeekOffset)

원점복귀의 회귀동작에서 홈센서 off(비활성화상태)를 기준으로 감속 정지할 상대거리를 설정합니다.
이 파라메터는 홈센서탐색을 위한 회귀동작에서 가감속동작을 실행하는 보간동작형 원점복귀명령인 [HOME]에만 적용되며 회귀동작시 일정속도(원점센서탐색속도)로 움직이는 개별축동작형 원점복귀명령인 [IHOME]명령에는 적용되지 않습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

감속정지할 상대거리(펄스)값을 반환합니다.(0~65535)

Return code/value	Description

➲ Example

➲ See Also

30.1.9. WIMC_Set_HomeSeekVel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_HomeSeekVel
(
              WIMC_UINT8              axis,
              WIMC_INT32              pps
);
```

➲ Description

특정축의 원점복귀 탐색속도 (Home Seek Velocity)를 설정합니다.

❶ 원점복귀 탐색속도 (Home Seek Velocity)

원점복귀의 홈센서 회귀동작시 홈센서를 탐색하는 속도를 설정합니다. 이 설정속도가 너무 빠르면 홈센서의 탐색위치가 정확하지 않을 수 있으며, 너무 느리면 원점복귀완료시간이 길어지게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

pps

원점복귀 탐색 속도입니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.10. WIMC_Get_HomeSeekVel

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_HomeSeekVel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 원점복귀 탐색속도 (Home Seek Velocity)를 읽습니다.

❶ 원점복귀 탐색속도 (Home Seek Velocity)

원점복귀의 홈센서 회귀동작시 홈센서를 탐색하는 속도입니다. 이 설정속도가 너무 빠르면 홈센서의 탐색위치가 정확하지 않을 수 있으며, 너무 느리면 원점복귀완료시간이 길어지게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

원점복귀 탐색 속도를 반환합니다.

단위는 기본적으로 PPS(Pulse Per Second)입니다. 단위는PPS^2이지만 간단히 모두 PPS 로 표현합니다.

Return code/value	Description

➲ Example

➲ See Also

30.1.11. WIMC_Set_HomeSensorLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_HomeSensorLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축 홈센서 레벨을 설정(high/low)합니다.

홈센서 레벨 (Home Sensor Level)

홈센서의 전기적 신호레벨이 L일 때와 H일 때를 선택하여 홈센서감지상태로 설정할 수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : high level로 설정합니다.

WIMC_LEVEL_L : Low level로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.12. WIMC_Is_HomeSensorLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeSensorLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 홈센서 레벨 설정(high/low)을 확인합니다.

홈센서 레벨 (Home Sensor Level)

홈센서의 전기적 신호레벨이 L일 때와 H일 때를 선택하여 홈센서감지상태로 설정할 수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

홈센서 레벨을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

30.1.13. WIMC_Is_HomeSensor

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeSensor
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 원점복귀 센서 동작을 확인합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	홈센서가 검출되었습니다.
WIMC_FALSE	홈센서가 검출되지 않았습니다.

➲ Example

➲ See Also

30.1.14. WIMC_Enable_HomeReturnAutoZero

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_HomeReturnAutoZero
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축 원점복귀 완료 후 자동영점설정기능을 Enable/Disable 합니다.

❶ 자동 영점 설정(Home Auto Zero Mode)

자동영점설정기능이란 특정상황에서 현재위치값을 전부 0으로 설정하는 기능을 말합니다. WIMC 엔진은 두가지 방식의 자동영점설정기능을 제공합니다.
하나는 원점복귀 완료후 자동영점설정기능이며, 다른 하나는 홈센서감지에 의한 자동영점설정기능입니다.
원점복귀 완료 후 자동영점설정기능을 활성화 하게 되면 현재위치가 항상 0이 되어 원점복귀편차알람을 발생할 없게 됩니다.

※ 홈센서감지에의 의한 자동영점설정기능은 홈센서가 비신호상태(비활성화상태)에서 신호상태(활성화상태)로 바뀌는 순간에 현재위치값이 0 으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 기능을 enable 합니다.

WIMC_FALSE : 기능을 disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.15. WIMC_Is_HomeReturnAutoZero

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeReturnAutoZero
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 원점복귀 완료 후 자동 영점설정기능의 설정(Enable/Disable)을 확인합니다.

❶ 자동 영점 설정(Home Auto Zero Mode)

자동영점설정기능이란 특정상황에서 현재위치값을 전부 0으로 설정하는 기능을 말합니다. WIMC 엔진은 두가지 방식의 자동영점설정기능을 제공합니다.
하나는 원점복귀 완료후 자동영점설정기능이며, 다른 하나는 홈센서감지에 의한 자동영점설정기능입니다.
원점복귀 완료 후 자동영점설정기능을 활성화 하게 되면 현재위치가 항상 0이 되어 원점복귀편차알람을 발생할 없게 됩니다.

※ 홈센서감지에의 의한 자동영점설정기능은 홈센서가 비신호상태(비활성화상태)에서 신호상태(활성화상태)로 바뀌는 순간에 현재위치값이 0 으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

자동영점 설정을 반환합니다.

Return code/value	Description
WIMC_TRUE	자동영점 기능이 설정되어 있습니다.
WIMC_FALSE	자동영점 기능이 해제되어 있습니다.

➲ Example

➲ See Also

30.1.16. WIMC_Enable_HomeSensorAutoZero

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_HomeSensorAutoZero
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

홈센서 감지에 의한 자동 영점 설정 기능을 Enable/Disable 합니다.

❶ 자동 영점 설정(Home Auto Zero Mode)

자동영점설정기능이란 특정상황에서 현재위치값을 전부 0으로 설정하는 기능을 말합니다. WIMC 엔진은 두가지 방식의 자동영점설정기능을 제공합니다.

하나는 원점복귀 완료후 자동영점설정기능이며, 다른 하나는 홈센서감지에 의한 자동영점설정기능입니다.

원점복귀 완료 후 자동영점설정기능을 활성화 하게 되면 현재위치가 항상 0이 되어 원점복귀편차알람을 발생할 없게 됩니다.

※ 홈센서감지에의 의한 자동영점설정기능은 홈센서가 비신호상태(비활성화상태)에서 신호상태(활성화상태)로 바뀌는 순간에 현재위치값이 0으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 기능을 enable 합니다.

WIMC_FALSE : 기능을 disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.17. WIMC_Is_HomeSensorAutoZero

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeSensorAutoZero
(
              WIMC_UINT8              axis
);
```

➲ Description

홈센서 감지에 의한 자동 영점 설정 기능을 확인합니다.

❶ 자동 영점 설정(Home Auto Zero Mode)

자동영점설정기능이란 특정상황에서 현재위치값을 전부 0으로 설정하는 기능을 말합니다. WIMC 엔진은 두가지 방식의 자동영점설정기능을 제공합니다.

하나는 원점복귀 완료후 자동영점설정기능이며, 다른 하나는 홈센서감지에 의한 자동영점설정기능입니다.

원점복귀 완료 후 자동영점설정기능을 활성화 하게 되면 현재위치가 항상 0이 되어 원점복귀편차알람을 발생할 없게 됩니다.

※ 홈센서감지에의 의한 자동영점설정기능은 홈센서가 비신호상태(비활성화상태)에서 신호상태(활성화상태)로 바뀌는 순간에 현재위치값이 0으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

자동영점 설정을 반환합니다.

Return code/value	Return code/value
WIMC_TRUE	자동영점 기능이 설정되어 있습니다.
WIMC_FALSE	자동영점 기능이 해제되어 있습니다.

➲ Example

➲ See Also

30.1.18. WIMC_Enable_HomeServoReset

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_HomeServoReset
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

원점복귀 후 Servo Reset 신호 출력을 설정(Enable/ Disable) 합니다.

❶ 원점복귀 후 Servo Reset 신호 출력 (Home Return Servo Complete)

WIMC 엔진은 원점복귀 동작 후 서보드라이브의 편차클리어신호와 대응되는 Servo Reset신호에 펄스를 발생할 수 있습니다. 이 신호에 의해 서보드라이브는 내부적인 서보동작의 편차를 클리어할 수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 기능을 enable 합니다.

WIMC_FALSE : 기능을 disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.19. WIMC_Is_HomeServoReset

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeServoReset
(
              WIMC_UINT8              axis
);
```

➲ Description

원점복귀 후 Servo Reset 신호 출력 설정(Enable/ Disable)을 확인합니다.

❶ 원점복귀 후 Servo Reset 신호 출력 (Home Return Servo Complete)

WIMC 엔진은 원점복귀 동작 후 서보드라이브의 편차클리어신호와 대응되는 Servo Reset신호에 펄스를 발생할 수 있습니다. 이 신호에 의해 서보드라이브는 내부적인 서보동작의 편차를 클리어할 수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Enable 상태입니다.
WIMC_FALSE	Disable 상태입니다.

➲ Example

➲ See Also

30.1.20. WIMC_Enable_HomeDevAlarm

➲ Syntax

```
void        WIMC_API                WIMC_Enable_HomeDevAlarm
(
            WIMC_UINT8              axis,
            WIMC_BOOL               bEnable
);
```

➲ Description

원점복귀 완료 오차에 따른 알람발생 기능을 Enable/Disable 합니다.

❶ 원점복귀 알람

원점복귀동작 중 2가지 상황에서 알람이 발생할 수 있습니다.

① 원점복귀동작에 소요된 펄스수가 지정된 최대펄스수를 초과했을 때

② 원점복귀완료 후 현재위치가 0을 중심으로 지정된 범위를 벗어났을 때

첫 번째 알람을 발생시키는 인자는 원점복귀 명령에서 직접적으로 전달합니다.

(WIMC_Home(axis, pulseMax);// pulseMax로 원점복귀동작에 허용하는 최대펄스수를 지정합니다.)

두 번째 알람은 선택적으로 활성화 할 수 있으며 편차펄스의 범위지정도 가능합니다.

위치오차의 허용치는 WIMC_Set_HomeDev(axis, dev)함수로 설정할 수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 원점 복귀 완료 오차에 따른 알람발생 기능 Enable.

WIMC_FALSE : 원점 복귀 완료 오차에 따른 알람발생 기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.21. WIMC_Is_HomeDevAlarm

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeDevAlarm
(
              WIMC_UINT8              axis
);
```

➲ Description

원점복귀 완료 오차에 따른 알람발생 기능을 확인합니다.

❶ 원점복귀 알람

원점복귀동작 중 2가지 상황에서 알람이 발생할 수 있습니다.

① 원점복귀동작에 소요된 펄스수가 지정된 최대펄스수를 초과했을 때

② 원점복귀완료 후 현재위치가 0을 중심으로 지정된 범위를 벗어났을 때

첫 번째 알람을 발생시키는 인자는 원점복귀 명령에서 직접적으로 전달합니다.

(WIMC_Home(axis, pulseMax);// pulseMax로 원점복귀동작에 허용하는 최대펄스수를 지정합니다.)

두 번째 알람은 선택적으로 활성화 할 수 있으며 편차펄스의 범위지정도 가능합니다.

위치오차의 허용치는 WIMC_Set_HomeDev(axis, dev)함수로 설정할 수 있습니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	알람발생 기능 Enable 상태입니다.
WIMC_FALSE	알람발생 기능 Disable 상태입니다.

➲ Example

➲ See Also

30.1.22. WIMC_Is_HomeReturnExe

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeReturnExe
(
              WIMC_UINT8              axis
);
```

➲ Description

원점복귀 동작 상태를 확인 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	원점복귀 동작 상태 입니다.
WIMC_FALSE	원점복귀 동작상태가 아닙니다.

➲ Example

➲ See Also

30.1.23. WIMC_Is_HomeReturnAlarm

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeReturnAlarm
(
              WIMC_UINT8              axis
);
```

➲ Description

원점복귀 알람 상태를 확인 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	원점복귀 알람 상태 입니다.
WIMC_FALSE	원점복귀 알람상태가 아닙니다.

➲ Example

➲ See Also

30.1.24. WIMC_ HomeLimitAxis

➲ Syntax

```
void        WIMC_API                WIMC_HomeLimitAxis
(
            WIMC_UINT8              axis,
            WIMC_BOOL               on
);
```

➲ Description

원점복귀 센서를 리미트 센서로 사용하도록 설정합니다.
원점복귀 시 원점 센서로 이용되며, 그외에는 리미트 센서로 인식하도록 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 원점복귀 센서를 리미트 센서로 사용합니다.
WIMC_FALSE : 기능 해제

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.25. WIMC_Is_HomeLimitAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_HomeLimitAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

원점복귀 센서를 리미트 센서로 사용하는 옵션 설정 상태를 확인합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

원점복귀 센서를 리미트 센서로 사용하는 옵션 설정 상태를 반환합니다.

Return code/value	Description
WIMC_TRUE	설정 상태
WIMC_FALSE	해제 상태

➲ Example

➲ See Also

30.1.26. WIMC_ HomeBackSearchAxis

➲ Syntax

```
void        WIMC_API                WIMC_HomeBackSearchAxis
(
            WIMC_UINT8              axis,
            WIMC_BOOL               on
);
```

➲ Description

리미트 역탐색 기능을 설정합니다.
리미트 역탐색 기능은 원점 복귀 시 리미트 센서를 만나면 역탐색 이동하는 기능입니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : 기능 설정
WIMC_FALSE : 기능 해제

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

30.1.27. WIMC_Is_HomeBackSearchAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_HomeBackSearchAxis
(
              WIMC_UINT8                axis
);
```

➲ Description

리미트 역탐색 기능을 설정을 확인 합니다.
리미트 역탐색 기능은 원점 복귀 시 리미트 센서를 만나면 역탐색 이동하는 기능입니다.

➲ Parameters

axis
적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

리미트 역탐색 기능 설정 상태를 반환합니다.

Return code/value	Description
WIMC_TRUE	설정 상태
WIMC_FALSE	해제 상태

➲ Example

➲ See Also

31. SERVO

31.1. Sv.On

31.1.1. WIMC_Enable_ServoOnAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_ServoOnAuto
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 Servo On 신호의 자동화 기능을 Enable/Disable 합니다.

ʊ Servo On 출력신호

Servo On 신호의 자동화 기능을 설정하면 아래의 4가지 방식중 하나를 선택할 수 있습니다.

(1) 모드0 : 신호발생조건 = Reset후 서보알람이 아닐 때

(2) 모드1 : 신호발생조건 = Reset후 서보알람과 그 외 알람에 의한 비상실행상태가 아닐 때

(3) 모드2 : 신호발생조건 = 모드 0 조건 & 모션동작이 실행될 때

(4) 모드3 : 신호발생조건 = 모드 1 조건 & 모션동작이 실행될 때

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Servo On 출력신호 자동화 기능을 Enable 합니다.

WIMC_FALSE : Servo On 출력신호 자동화 기능을 Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.1.2. WIMC_Is_ServoOnAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoOnAuto
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo On 신호의 자동화 기능을 확인합니다.

❶ Servo On 출력신호

Servo On 신호의 자동화 기능을 설정하면 아래의 4가지 방식중 하나를 선택할 수 있습니다.

(1) 모드0 : 신호발생조건 = Reset후 서보알람이 아닐 때

(2) 모드1 : 신호발생조건 = Reset후 서보알람과 그 외 알람에 의한 비상실행상태가 아닐 때

(3) 모드2 : 신호발생조건 = 모드 0 조건 & 모션동작이 실행될 때

(4) 모드3 : 신호발생조건 = 모드 1 조건 & 모션동작이 실행될 때

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Servo On 신호의 자동화 기능 설정 상태를 반환합니다.

Return code/value	Description
WIMC_TRUE	Servo On 출력신호 자동화 기능이 Enable 상태입니다.
WIMC_FALSE	Servo On 출력신호 자동화 기능이 Disable 상태입니다.

➲ Example

➲ See Also

31.1.3. WIMC_Set_ServoOnLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoOnLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 Servo On 활성화 신호레벨을 설정합니다.

Servo On 출력신호

Servo On 신호의 자동화 기능을 설정하면 아래의 4가지 방식중 하나를 선택할 수 있습니다.

(1) 모드0 : 신호발생조건 = Reset후 서보알람이 아닐 때

(2) 모드1 : 신호발생조건 = Reset후 서보알람과 그 외 알람에 의한 비상실행상태가 아닐 때

(3) 모드2 : 신호발생조건 = 모드 0 조건 & 모션동작이 실행될 때

(4) 모드3 : 신호발생조건 = 모드 1 조건 & 모션동작이 실행될 때

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : 신호레벨 High 설정입니다.

WIMC_LEVEL_L : 신호레벨 Low 설정입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.1.4. WIMC_Is_ServoOnLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoOnLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo On 활성화 신호레벨을 확인합니다.

❶ Servo On 출력신호

Servo On 신호의 자동화 기능을 설정하면 아래의 4가지 방식중 하나를 선택할 수 있습니다.

(1) 모드0 : 신호발생조건 = Reset후 서보알람이 아닐 때

(2) 모드1 : 신호발생조건 = Reset후 서보알람과 그 외 알람에 의한 비상실행상태가 아닐 때

(3) 모드2 : 신호발생조건 = 모드 0 조건 & 모션동작이 실행될 때

(4) 모드3 : 신호발생조건 = 모드 1 조건 & 모션동작이 실행될 때

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

servo On 활성화 신호레벨을 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

31.1.5. WIMC_Set_ServoOnMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoOnMode
(
              WIMC_UINT8              axis,
              WIMC_INT32              mode
);
```

➲ Description

특정축의 Servo On 자동화 기능 동작 모드를 설정합니다.

Servo On 출력신호

Servo On 신호의 자동화 기능을 설정하면 아래의 4가지 방식중 하나를 선택할 수 있습니다.

(1) 모드0 : 신호발생조건 = Reset후 서보알람이 아닐 때

(2) 모드1 : 신호발생조건 = Reset후 서보알람과 그 외 알람에 의한 비상실행상태가 아닐 때

(3) 모드2 : 신호발생조건 = 모드 0 조건 & 모션동작이 실행될 때

(4) 모드3 : 신호발생조건 = 모드 1 조건 & 모션동작이 실행될 때

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

mode

동작 모드를 설정합니다.

WIMC_SERVOONMODE_0 : 신호발생조건 = Reset후 서보알람이 아닐 때

WIMC_SERVOONMODE_1 : 신호발생조건 = Reset후 서보알람과 그 외 알람에 의한 비상실행상태가 아닐 때

WIMC_SERVOONMODE_2 : 신호발생조건 = 모드 0 조건 & 모션동작이 실행될 때

WIMC_SERVOONMODE_3 : 신호발생조건 = 모드 1 조건 & 모션동작이 실행될 때

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.1.6. WIMC_Get_ServoOnMode

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ServoOnMode
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo On 자동화 기능 동작 모드를 읽습니다.

❶ Servo On 출력신호

Servo On 신호의 자동화 기능을 설정하면 아래의 4가지 방식중 하나를 선택할 수 있습니다.

(1) 모드0 : 신호발생조건 = Reset후 서보알람이 아닐 때

(2) 모드1 : 신호발생조건 = Reset후 서보알람과 그 외 알람에 의한 비상실행상태가 아닐 때

(3) 모드2 : 신호발생조건 = 모드 0 조건 & 모션동작이 실행될 때

(4) 모드3 : 신호발생조건 = 모드 1 조건 & 모션동작이 실행될 때

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Servo On 자동화 기능 동작 모드를 반환합니다.

Return code/value	Description
WIMC_SERVOONMODE_0	신호발생조건 = Reset후 서보알람이 아닐 때
WIMC_SERVOONMODE_1	신호발생조건 = Reset후 서보알람과 그 외 알람에 의한 비상실행상태가 아닐 때
WIMC_SERVOONMODE_2	신호발생조건 = 모드 0 조건 & 모션동작이 실행될 때
WIMC_SERVOONMODE_3	신호발생조건 = 모드 1 조건 & 모션동작이 실행될 때

➲ Example

➲ See Also

31.1.7. WIMC_Set_ServoOn

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoOn
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bOn
);
```

➲ Description

특정축의 Servo On 활성화 또는 비활성화 신호를 출력합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Servo On 활성화 신호 출력

WIMC_FALSE : Servo On 비활성화 신호 출력

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.1.8. WIMC_Is_ServoOn

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_ServoOn
(
              WIMC_UINT8                axis
);
```

➲ Description

특정축의 Servo On 활성화 신호를 출력상태를 확인합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Servo On 활성화 신호 출력상태 입니다.
WIMC_FALSE	Servo On 활성화 신호 출력상태가 아닙니다.

➲ Example

➲ See Also

31.2. Sv.Reset

31.2.1. WIMC_Enable_ServoResetAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_ServoResetAuto
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 Servo Reset 자동화 기능을 Enable/Disable합니다.

servo Reset 출력신호

Servo Reset 신호의 자동화 기능으로 설정하면 원점복귀 완료 후 지정된 펄스폭의 펄스신호가 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Servo Reset 자동화 기능을 Enable 합니다.

WIMC_FALSE : Servo Reset 자동화 기능을 Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.2.2. WIMC_Is_ServoResetAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoResetAuto
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Reset 자동화 기능 설정을 확인합니다.

servo Reset 출력신호

Servo Reset 신호의 자동화 기능으로 설정하면 원점복귀 완료 후 지정된 펄스폭의 펄스신호가 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

특정축의 Servo Reset 자동화 기능설정을 반환합니다.

Return code/value	Description
WIMC_TRUE	servo reset 자동화 기능 설정상태입니다.
WIMC_FALSE	servo reset 자동화 기능 해제상태입니다.

➲ Example

➲ See Also

31.2.3. WIMC_Set_ServoResetLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoResetLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 Servo Reset 신호레벨을 설정합니다.

servo Reset 출력신호

Servo Reset 신호의 자동화 기능으로 설정하면 원점복귀 완료 후 지정된 펄스폭의 펄스신호가 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : 신호레벨 High 로 설정합니다.

WIMC_LEVEL_L : 신호레벨 Low로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.2.4. WIMC_Is_ServoResetLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoResetLevel
(
              WIMC_UINT8              aixs
);
```

➲ Description

특정축의 Servo Reset 신호레벨을 확인합니다.

servo Reset 출력신호

Servo Reset 신호의 자동화 기능으로 설정하면 원점복귀 완료 후 지정된 펄스폭의 펄스신호가 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

31.2.5. WIMC_Set_ServoPulseTime

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoPulseTime
(
              WIMC_UINT8              axis,
              WIMC_INT32              msec
);
```

➲ Description

Servo reset 신호 자동화 기능에서 Servo Reset 펄스출력 시간을 설정합니다.

servo Reset 출력신호

Servo Reset 신호의 자동화 기능으로 설정하면 원점복귀 완료 후 지정된 펄스폭의 펄스신호가 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

msec

Servo reset 펄스 출력 시간입니다. milliseconds 단위입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.2.6. WIMC_Get_ServoPulseTime

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ServoPulseTime
(
              WIMC_UINT8              axis
);
```

➲ Description

Servo reset 신호 자동화 기능에서 Servo Reset 펄스출력 시간을 읽습니다.

servo Reset 출력신호

Servo Reset 신호의 자동화 기능으로 설정하면 원점복귀 완료 후 지정된 펄스폭의 펄스신호가 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

servo reset 펄스 출력시간을 milliseconds 단위로 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

31.2.7. WIMC_Set_ServoReset

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoReset
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bOn
);
```

➲ Description

Servo Reset 활성화 신호 또는 비활성화 신호를 출력합니다.

servo Reset 출력신호

Servo Reset 신호의 자동화 기능으로 설정하면 원점복귀 완료 후 지정된 펄스폭의 펄스신호가 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Servo Reset 활성화 신호를 출력합니다.

WIMC_FALSE : Servo Reset 비활성화 신호를 출력합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.2.8. WIMC_Is_ServoReset

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoReset
(
              WIMC_UINT8              axis
);
```

➲ Description

Servo Reset 활성화 신호 출력 상태를 확인합니다.

servo Reset 출력신호

Servo Reset 신호의 자동화 기능으로 설정하면 원점복귀 완료 후 지정된 펄스폭의 펄스신호가 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Servo Reset 활성화 신호출력 상태입니다.
WIMC_FALSE	Servo Reset 활성화 신호출력 상태가 아닙니다.

➲ Example

➲ See Also

31.3. Sv.EStop

31.3.1. WIMC_Enable_ServoEStopAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_ServoEStopAuto
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 Servo 비상정지(EStop) 신호 자동화 기능을 Enable/Disable 합니다.

Servo EStop 출력신호

Servo Estop 신호의 자동화 기능으로 설정하면 모션엔진의 알람에 의한 비상실행상태일 때 신호를 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable 합니다.

WIMC_FALSE : Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.3.2. WIMC_Is_ServoEStopAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoEStopAuto
(
              WIMC_UINT8              axis
);
```

➲ Description

Servo 비상정지(EStop) 신호 자동화 기능 설정상태를 확인합니다.

Servo EStop 출력신호

Servo Estop 신호의 자동화 기능으로 설정하면 모션엔진의 알람에 의한 비상실행상태일 때 신호를 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Servo 비상정지 신호 자동화 기능이 Enable 상태입니다.
WIMC_FALSE	Servo 비상정지 신호 자동화 기능이 Disable 상태입니다.

➲ Example

➲ See Also

31.3.3. WIMC_Set_ServoEStopLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoEStopLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 Servo 비상정지(EStop) 활성화 신호레벨(high/low)을 설정합니다.

Servo EStop 출력신호

Servo Estop 신호의 자동화 기능으로 설정하면 모션엔진의 알람에 의한 비상실행상태일 때 신호를 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : 신호레벨 High로 설정합니다.

WIMC_LEVEL_L : 신호레벨 Low로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.3.4. WIMC_Is_ServoEStopLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoEStopLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo 비상정지(EStop) 활성화 신호레벨(high/low)을 확인합니다.

Servo EStop 출력신호

Servo Estop 신호의 자동화 기능으로 설정하면 모션엔진의 알람에 의한 비상실행상태일 때 신호를 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

31.3.5. WIMC_Set_ServoEStop

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Set_ServoEStop
(
             WIMC_UINT8              axis,
             WIMC_BOOL               bOn
);
```

➲ Description

특정축의 Servo 비상정지(EStop) 활성화 신호 또는 비활성화 신호를 출력합니다.

Servo EStop 출력신호

Servo Estop 신호의 자동화 기능으로 설정하면 모션엔진의 알람에 의한 비상실행상태일 때 신호를 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Servo EStop 활성화 신호를 출력합니다.

WIMC_FALSE : Servo EStop 비활성화 신호를 출력합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.3.6. WIMC_Is_ServoEStop

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoEStop
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo 비상정지(EStop) 활성화 신호 출력 상태를 확인합니다.

Servo EStop 출력신호

Servo Estop 신호의 자동화 기능으로 설정하면 모션엔진의 알람에 의한 비상실행상태일 때 신호를 발생합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Servo EStop 활성화 신호출력 상태입니다.
WIMC_FALSE	Servo Estop 활성화 신호출력 상태가 아닙니다.

➲ Example

➲ See Also

31.4. Sv.UserOut

31.4.1. WIMC_Enable_ServoUserOutAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_ServoUserOutAuto
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

Servo 사용자 신호 자동화 기능을 Enable/Disable 합니다.

Servo UserOut 출력신호

Servo UserOut신호는 사용자가 임의적으로 사용될 수 있는 출력신호로 규정되었지만 자동화 기능을 설정할 경우 Servo UserIn신호를 그대로 전달하는 역할 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable 합니다.

WIMC_FALSE : Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.4.2. WIMC_Is_ServoUserOutAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_ServoUserOutAuto
(
              WIMC_UINT8                axis
);
```

➲ Description

특정축의 Servo 사용자 신호 자동화 기능 설정상태를 확인합니다.

Servo UserOut 출력신호

Servo UserOut신호는 사용자가 임의적으로 사용될 수 있는 출력신호로 규정되었지만 자동화 기능을 설정할 경우 Servo UserIn신호를 그대로 전달하는 역할 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	설정 상태입니다.
WIMC_FALSE	해제 상태입니다.

➲ Example

➲ See Also

31.4.3. WIMC_Set_ServoUserOutLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoUserOutLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 Servo 사용자 출력 활성화 신호레벨을 설정합니다.

Servo UserOut 출력신호

Servo UserOut신호는 사용자가 임의적으로 사용될 수 있는 출력신호로 규정되었지만 자동화 기능을 설정할 경우 Servo UserIn신호를 그대로 전달하는 역할 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : Servo UserOut 활성화 신호레벨 High 설정합니다.

WIMC_LEVEL_L : Servo UserOut 활성화 신호레벨 Low 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.4.4. WIMC_Is_ServoUserOutLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_ServoUserOutLevel
(
              WIMC_UINT8                axis
);
```

➲ Description

특정축의 Servo 사용자 출력 활성화 신호레벨을 확인합니다.

Servo UserOut 출력신호

Servo UserOut신호는 사용자가 임의적으로 사용될 수 있는 출력신호로 규정되었지만 자동화 기능을 설정할 경우 Servo UserIn신호를 그대로 전달하는 역할 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

31.4.5. WIMC_Set_ServoUserOut

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoUserOut
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bOn
);
```

➲ Description

특정축의 Servo 사용자 출력 활성화 신호 또는 비활성화 신호를 출력합니다.

Servo UserOut 출력신호

Servo UserOut신호는 사용자가 임의적으로 사용될 수 있는 출력신호로 규정되었지만 자동화 기능을 설정할 경우 Servo UserIn신호를 그대로 전달하는 역할 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Servo UserOut 활성화 신호를 출력합니다.

WIMC_FALSE : Servo UserOut 활성화 신호를 출력하지 않습니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.4.6. WIMC_Is_ServoUserOut

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_ServoUserOut
(
              WIMC_UINT8                axis
);
```

➲ Description

특정축의 Servo 사용자 출력 활성화 신호 출력 상태를 확인합니다.

Servo UserOut 출력신호

Servo UserOut신호는 사용자가 임의적으로 사용될 수 있는 출력신호로 규정되었지만 자동화 기능을 설정할 경우 Servo UserIn신호를 그대로 전달하는 역할 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Servo User Out 활성화 신호 출력상태입니다.
WIMC_FALSE	Servo User Out 활성화 신호 출력상태가 아닙니다.

➲ Example

➲ See Also

31.5. Sv.Ready

31.5.1. WIMC_Enable_ServoReadyAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_ServoReadyAuto
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

서보드라이브의 준비신호 자동화기능을 설정(Enable/Disable)합니다.

Servo Ready 입력신호

서보드라이브의 준비신호로 자동화기능을 설정하게 되면, Servo Ready신호가 발생되지 않았을 때 보간동작명령을 실행하면 알람이 발생하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable 합니다.

WIMC_FALSE : Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.5.2. WIMC_Is_ServoReadyAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoReadyAuto
(
              WIMC_UINT8              axis
);
```

➲ Description

Servo ready 신호 자동화기능 설정 상태를 확인합니다.

Servo Ready 입력신호

서보드라이브의 준비신호로 자동화기능을 설정하게 되면, Servo Ready신호가 발생되지 않았을 때 보간동작명령을 실행하면 알람이 발생하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	설정 상태입니다.
WIMC_FALSE	해제 상태입니다.

➲ Example

➲ See Also

31.5.3. WIMC_Set_ServoReadyLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoReadyLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 Servo Ready 신호 자동화 기능 활성화 레벨을 설정합니다.

Servo Ready 입력신호

서보드라이브의 준비신호로 자동화기능을 설정하게 되면, Servo Ready신호가 발생되지 않았을 때 보간동작명령을 실행하면 알람이 발생하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_LEVEL_H : 신호 레벨 High로 설정합니다.

WIMC_LEVEL_L : 신호 레벨 Low로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.5.4. WIMC_Is_ServoReadyLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoReadyLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Ready 신호 자동화 기능 활성화 레벨(High/Low)을 확인합니다.

Servo Ready 입력신호

서보드라이브의 준비신호로 자동화기능을 설정하게 되면, Servo Ready신호가 발생되지 않았을 때 보간동작명령을 실행하면 알람이 발생하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

31.5.5. WIMC_Is_ServoReady

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoReady
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Ready 활성화 신호를 확인합니다.

Servo Ready 입력신호

서보드라이브의 준비신호로 자동화기능을 설정하게 되면, Servo Ready신호가 발생되지 않았을 때 보간동작명령을 실행하면 알람이 발생하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Servo Ready 활성화 신호가 발생 했습니다.
WIMC_FALSE	Servo Ready 활성화 신호가 발생하지 않았습니다.

➲ Example

➲ See Also

31.6. Sv.Inpos

31.6.1. WIMC_Enable_ServoInposAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_ServoInposAuto
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 Servo Inposition 신호의 자동화 기능을 Enable/Disable 합니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable 합니다.

WIMC_FALSE : Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.6.2. WIMC_Is_ServoInposAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoInposAuto
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Inposition 신호의 자동화 기능을 설정(Enable/Disable) 상태를 확인합니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	설정 상태입니다.
WIMC_FALSE	해제 상태입니다.

➲ Example

➲ See Also

31.6.3. WIMC_Enable_ServoInposAlarm

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_ServoInposAlarm
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 Servo Inposition 신호의 자동화기능에 대한 알람 기능을 Enable/Disable 합니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable 합니다.
WIMC_FALSE : Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.6.4. WIMC_Is_ServoInposAlarm

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoInposAlarm
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Inposition 신호의 자동화기능에 대한 알람 기능 설정(Enable/Disable) 상태를 확인합니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	설정 상태입니다.
WIMC_FALSE	해제 상태입니다.

➲ Example

➲ See Also

31.6.5. WIMC_Set_ServoInposLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoInposLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 Servo Inposition 신호의 자동화 기능 활성화 레벨(High/Low)을 설정합니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : 신호레벨 High로 설정합니다.
WIMC_LEVEL_L : 신호레벨 Low로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.6.6. WIMC_Is_ServoInposLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoInposLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Inposition 신호의 자동화 기능 활성화 레벨(High/Low)을 확인합니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

31.6.7. WIMC_Set_ServoInposTime

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoInposTime
(
              WIMC_UINT8              axis,
              WIMC_INT32              msec
);
```

➲ Description

특정축의 Servo Inpos 신호 감지 최대 대기시간을 milliseconds 단위로 설정합니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

msec

milliseconds 단위입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.6.8. WIMC_Get_ServoInposTime

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ServoInposTime
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Inpos 신호 감지 최대 대기시간을 milliseconds 단위로 읽습니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

milliseconds 단위로 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

31.6.9. WIMC_Is_ServoInpos

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoInpos
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Inpos 활성화 신호를 확인합니다.

Servo Inpos 입력신호

Servo Inpos(Inposition) 신호의 자동화 기능을 설정하면 보간명령 동작이 시작될 때 동작축의 Servo Inpos신호들이 모두 활성화 상태가 될 때까지 대기한 후 실행하게 됩니다. 대기시간이 설정된 시간을 경과하게 되면 그냥 실행하는 방식과 알람을 발생시켜 동작을 중지시키는 방식을 선택할 수 있습니다. 이때 알람의 내용은 위치알람이 됩니다.
동작을 중지시키는 알람을 발생하기 위해서 아래의 라이브러리 함수로 ServoInpos신호에 대한 알람을 설정합니다.
- WIMC_Enable_ServoInposAlarm(axis, TRUE);
Servo Inpos신호는 이전 보간명령에 대한 서보드라이브의 모터 움직임이 완전히 완료된 후에 다음명령이 실행되고자 할 때 사용됩니다.

Servo Inpos신호의 자동화 기능이 설정되면 개별축의 Complete신호는 축의 움직임이 없고 Inpos의 신호상태가 활성화 되어야만 발생합니다. 또한 스핀들 축의 inpos신호가 활성화 되었을 경우 스핀들 회전명령은 스핀들 회전이 가감속상태가 아닌 정상상태가 될 때까지 대기하게 됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

특정축 드웰타임을 milliseconds 단위로 반환합니다.

Return code/value	Description
WIMC_TRUE	Servo Inpos 활성화 신호가 발생 했습니다.
WIMC_FALSE	Servo Inpos 활성화 신호가 발생하지 않았습니다.

➲ Example

➲ See Also

31.7. Sv.Alarm

31.7.1. WIMC_Enable_ServoAlarmAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_ServoAlarmAuto
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bEnable
);
```

➲ Description

특정축의 Servo Alarm 신호의 자동화 기능을 Enable/Disable 합니다.

Servo Alarm 입력신호

Servo Alarm 신호의 자동화 기능을 설정하면 서보알람신호가 발생할 때 모션내부의 알람이 발생됩니다. 알람내용은 서보알람으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable 합니다.

WIMC_FALSE : Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.7.2. WIMC_Is_ServoAlarmAuto

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoAlarmAuto
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Alarm 신호의 자동화 기능의 설정(Enable/Disable)을 확인합니다.

Servo Alarm 입력신호

Servo Alarm 신호의 자동화 기능을 설정하면 서보알람신호가 발생할 때 모션내부의 알람이 발생됩니다. 알람내용은 서보알람으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	설정 상태입니다.
WIMC_FALSE	해제 상태입니다.

➲ Example

➲ See Also

31.7.3. WIMC_Set_ServoAlarmLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoAlarmLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 Servo Alarm 신호의 자동화 기능 활성화 레벨(High/Low)을 설정합니다.

Servo Alarm 입력신호

Servo Alarm 신호의 자동화 기능을 설정하면 서보알람신호가 발생할 때 모션내부의 알람이 발생됩니다. 알람내용은 서보알람으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : high level로 설정합니다.

WIMC_LEVEL_L : Low level로 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.7.4. WIMC_Is_ServoAlarmLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoAlarmLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Alarm 신호의 자동화 기능 활성화 레벨(High/Low)을 확인합니다.

Servo Alarm 입력신호

Servo Alarm 신호의 자동화 기능을 설정하면 서보알람신호가 발생할 때 모션내부의 알람이 발생됩니다. 알람내용은 서보알람으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

31.7.5. WIMC_Is_ServoAlarm

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoAlarm
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo Alarm 활성화 신호를 확인합니다.

Servo Alarm 입력신호

Servo Alarm 신호의 자동화 기능을 설정하면 서보알람신호가 발생할 때 모션내부의 알람이 발생됩니다. 알람내용은 서보알람으로 설정됩니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Servo Alarm 활성화 신호가 발생했습니다.
WIMC_FALSE	Servo Alarm 활성화 신호가 발생하지 않았습니다.

➲ Example

➲ See Also

31.8. Sv.UserIn

31.8.1. WIMC_Set_ServoUserInLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_ServoUserInLevel
(
              WIMC_UINT8              axis,
              WIMC_BOOL               bLevel
);
```

➲ Description

특정축의 Servo User 입력 신호 자동화 기능 활성화 레벨(High/Low)을 설정합니다.

Servo UserIn 입력신호

Servo UserIn신호는 사용자가 임의적으로 사용될 수 있는 입력신호로써 별도의 자동화 기능에 대한 동작을 실행하지 않습니다. 그러므로 이 신호에 대한 자동화 기능설정은 없으며 단지 읽기만 가능합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bLevel

WIMC_LEVEL_H : Servo UserIn 활성화 신호레벨 High 설정합니다.

WIMC_LEVEL_L : Servo UserIn 활성화 신호레벨 Low 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

31.8.2. WIMC_Is_ServoUserInLevel

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_ServoUserInLevel
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 Servo User 입력 신호 자동화 기능 활성화 레벨(High/Low)을 확인합니다.

Servo UserIn 입력신호

Servo UserIn신호는 사용자가 임의적으로 사용될 수 있는 입력신호로써 별도의 자동화 기능에 대한 동작을 실행하지 않습니다. 그러므로 이 신호에 대한 자동화 기능설정은 없으며 단지 읽기만 가능합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

특정축 드웰타임을 milliseconds 단위로 반환합니다.

Return code/value	Description
WIMC_LEVEL_H(1)	High Level
WIMC_LEVEL_L(0)	Low Level

➲ Example

➲ See Also

31.8.3. WIMC_Is_ServoUserIn

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_ServoUserIn
(
              WIMC_UINT8                axis
);
```

➲ Description

특정축의 Servo 사용자 입력 활성화 신호를 확인합니다.

Servo UserIn 입력신호

Servo UserIn신호는 사용자가 임의적으로 사용될 수 있는 입력신호로써 별도의 자동화 기능에 대한 동작을 실행하지 않습니다. 그러므로 이 신호에 대한 자동화 기능설정은 없으며 단지 읽기만 가능합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	Servo UserIn 활성화 신호가 발생했습니다.
WIMC_FALSE	Servo UserIn 활성화 신호가 발생하지 않았습니다.

➲ Example

➲ See Also

32. GIO

32.1. 범용 및 전용 입출력

WIMC 엔진은 범용입출력포트와 아날로그입력을 위한 전용 ADC포트, PWM출력을 위한 전용 포트를 갖추고 있습니다.
범용입출력포트는 일반적인 입출력기능이외에도 엔코더, 카운터, PID제어출력용 PWM기능을 포함하고 있습니다. 각 기능들을 요약하면 아래와 같습니다.

(1) 범용입력포트(GIn) = 8pin, Encoder Counter = 2Chanel, UpCounter = 2Channel
(2) 범용출력포트(GOut) = 8pin, PID제어출력용 PWM = 4Channel
(3) 아날로그입력포트(ADC) = 4Channel
(4) PWM출력포트(PWM) = 2Channel

입력신호 기능

WIMC 엔진은 8포트의 범용입력신호를 갖습니다. 8개의 핀은 엔코더카운터와 업카운터기능을 병행하고 있습니다.
이들 기능 핀에 대한 맵핑표는 아래와 같습니다.
<표> 입력포트 기능 핀맵

기능 / 포트	7	6	5	4	3	2	1	0	비고
일반 디지털입력포트	o	o	o	o	o	o	o	o	
UP COUNTER 채널#1(16bit)								UP신호	10kHz이내사용
UP COUNTER 채널#2(16bit)							UP신호		10kHz이내사용
4체배 엔코더카운터#1(32bit)					B상	A상			20kHz이내사용
4체배 엔코더카운터#2(32bit)			B상	A상					20kHz이내사용

출력 신호기능

WIMC 엔진은 8포트의 범용출력신호를 갖습니다. 8개의 핀은 PID제어용 PWM신호 출력기능과 병행하고 있습니다. PID제어용 PWM 출력신호에 대한 설명은 관련 함수들을 참조해 주세요.

이들 기능 핀에 대한 맵핑표는 아래와 같습니다.
<표> 출력포트 기능 핀맵

기능 / 포트	7	6	5	4	3	2	1	0	비고
일반 디지털입력포트	o	o	o	o	o	o	o	o	
PID #1 의 PWM 출력			2~5번 포트 중 한 개 포트선택						동작기준20kHz
PID #2 의 PWM 출력			2~5번 포트 중 한 개 포트선택						동작기준20kHz
PID #3 의 PWM 출력			2~5번 포트 중 한 개 포트선택						동작기준20kHz
PID #4 의 PWM 출력			2~5번 포트 중 한 개 포트선택						동작기준20kHz

32.1.1. WIMC_Set_Gout

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_GOut
(
              WIMC_UINT32             val
);
```

➲ Description

출력포트의 상태를 val 값으로 설정합니다.
하위 8비트만을 사용합니다.

❶ 범용 출력신호 기능

WIMC 엔진은 8포트의 범용출력신호를 갖습니다. 8개의 핀은 PID제어용 PWM신호 출력기능과 병행하고 있습니다.
PID제어용 PWM출력신호에 대한 설명은 PIC 관련 함수를 참고 하실 수 있습니다.

이들 기능 핀에 대한 맵핑표는 아래와 같습니다.
<표> 출력포트 기능 핀맵

기능 / 포트	7	6	5	4	3	2	1	0	비고
일반 디지털출력포트	o	o	o	o	o	o	o	o	
PID #1 의 PWM 출력			2~5번 포트 중 한 개 포트선택						동작기준20kHz
PID #2 의 PWM 출력			2~5번 포트 중 한 개 포트선택						동작기준20kHz
PID #3 의 PWM 출력			2~5번 포트 중 한 개 포트선택						동작기준20kHz
PID #4 의 PWM 출력			2~5번 포트 중 한 개 포트선택						동작기준20kHz

➲ Parameters

val
출력을 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

```
WIMC_Set_GOut( 1 << 7);          // 7번 포트 출력
```

➲ See Also

32.1.2. WIMC_SetBit_GOut

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_SetBit_GOut
(
              WIMC_UINT32             setMask
);
```

➲ Description

출력포트 마스크값(setMask)에 의해 포트를 개별적으로 High 레벨로 만듭니다.

❶ 바인딩된 범용 출력기능

범용 출력포트는 보간명령들과 동기화 되어 정확한 위치의 순간에 특정포트의 신호를 출력할 수 있는 기능을 갖습니다. 바인딩된 범용 출력기능은 현재 보간동작이 실행중이다면, 출력명령은 다음번 보간명령이 시작되는 시점에 실행됩니다. 만약 다음번 보간명령이 없다면 현재 보간동작명령 실행이 완전히 완료된 후에 실행됩니다.

이러한 바인딩된 범용 출력기능은 특정위치에 정확하게 신호를 보낼 수 있는 역할을 합니다. 이 기능은 위치비교기를 통해서도 실행될 수 있지만, WIMC 엔진의 위치비교기는 100usec마다 위치를 비교하는 방식이며 Trig신호핀을 공유하고 4채널로 한정되지만 바인딩된 범용 출력기능을 사용하게 되면 어떤 위치의 어떤 수만큼이든 상관없이 신호를 발생할 수 있으며 범용출력포트 전체를 사용할 수 있으므로 편리합니다.

또한 바인딩된 범용 출력용 함수들은 특정포트번호만을 Set/Clear할 수 있는 함수를 제공하며, 단일 펄스를 출력할 수도 있습니다.

➲ Parameters

setMask

출력포트 마스크 값입니다.

ex) setMask=0x03 은 포트 0번과 1번을 '1'(H 레벨)로 만들고 나머지 포트는 변화 없습니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

32.1.3. WIMC_ClrBit_GOut

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_ClrBit_GOut
(
              WIMC_UINT32             clrMask
);
```

➲ Description

출력포트 마스크값(clrMask)에 의해 포트를 개별적으로 Low 레벨로 만듭니다.

❶ 바인딩된 범용 출력기능

범용 출력포트는 보간명령들과 동기화 되어 정확한 위치의 순간에 특정포트의 신호를 출력할 수 있는 기능을 갖습니다. 바인딩된 범용 출력기능은 현재 보간동작이 실행중이다면, 출력명령은 다음번 보간명령이 시작되는 시점에 실행됩니다. 만약 다음번 보간명령이 없다면 현재 보간동작명령 실행이 완전히 완료된 후에 실행됩니다.

이러한 바인딩된 범용 출력기능은 특정위치에 정확하게 신호를 보낼 수 있는 역할을 합니다. 이 기능은 위치비교기를 통해서도 실행될 수 있지만, WIMC 엔진의 위치비교기는 100usec마다 위치를 비교하는 방식이며 Trig신호핀을 공유하고 4채널로 한정되지만 바인딩된 범용 출력기능을 사용하게 되면 어떤 위치의 어떤 수만큼이든 상관없이 신호를 발생할 수 있으며 범용출력포트 전체를 사용할 수 있으므로 편리합니다.

또한 바인딩된 범용 출력용 함수들은 특정포트번호만을 Set/Clear할 수 있는 함수를 제공하며, 단일 펄스를 출력할 수도 있습니다.

➲ Parameters

clrMask

출력포트 마스크 값입니다.

ex) clrMask=0x03 은 포트 0번과 1번을 '0'(L 레벨)로 만들고 나머지 포트는 변화 없습니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

32.1.4. WIMC_Pulse_GOut

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Pulse_GOut
(
              WIMC_UINT32             pulseMask
);
```

➲ Description

출력포트마스크값 pulseMask에 의해 포트를 개별적으로 H레벨 만든 후 특정시간(주기) 후에 자동으로 L레벨로 변화시키는 펄스형태를 만듭니다.

➲ Parameters

pulseMask

출력포트 마스크값입니다.

ex) pulseMask=0x03 이면 포트 0번과 1번을 '1'(H레벨)로 만들고 특정시간(주기) 후에 자동으로 '0'(L레벨)로 변화시키며 나머지 포트는 변화가 없습니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

32.1.5. WIMC_Get_GOut

➲ Syntax

```
WIMC_UINT32   WIMC_API                  WIMC_Get_GOut( void);
```

➲ Description

출력포트의 값을 32비트 부호 없는 값으로 읽습니다. 하위 8비트만을 사용합니다.

➲ Parameters

➲ Return Value

출력포트 값을 반환합니다.(하위 8비트만 사용합니다.)

Return code/value	Description

➲ Example

➲ See Also

32.1.6. WIMC_Get_GIn

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Get_GIn( void);
```

➲ Description

입력포트의 상태값을 32비트 부호없는 값으로 읽습니다. 하위 8비트만 사용합니다.

❶ 일반입력기능

입력신호를 읽어들이는 주기는 상태타이머의 주기에 의존합니다. 상태타이머의 정보 옵션에 GIO를 선택하고 상태타이머의 주기를 설정하면 설정된 주기마다 지정된 정보를 읽어들입니다. 상태타이머를 원하는 실시간영역내에 설정한 후 아래의 라이브러리 함수를 사용하면 범용입력포트를 읽어들일 수 있습니다.

➲ Parameters

➲ Return Value

입력폩의 상태값을 32비트 부호없는 값으로 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

32.1.7. WIMC_Enable_GIOEncoder

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_GIOEncoder
(
              WIMC_INT32              ch,
              WIMC_BOOL               bEnable
);
```

➲ Description

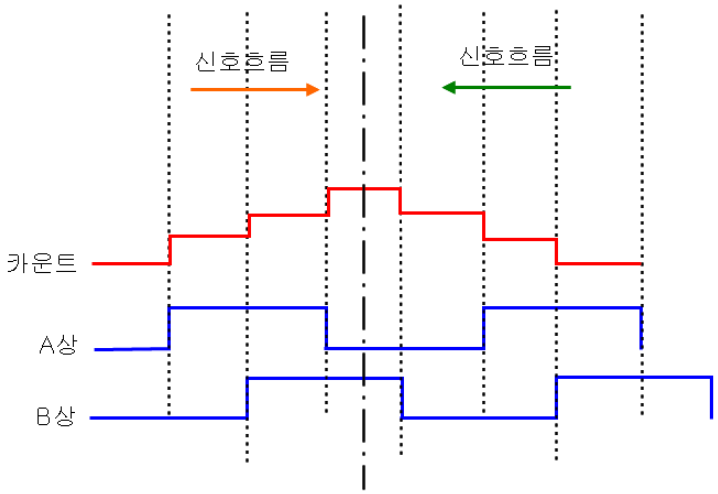
해당채널의 GIO용 엔코더 기능을 Enable/Disable 합니다.

❶ 엔코더 카운터 기능

엔코더 카운터는 A,B 위상차방식의 상대회전엔코더용 신호를 처리합니다. 기본적으로 4채배신호로 처리됩니다. WIMC 엔진의 엔코더 카운터는 입력 2번/3번 과 입력 4번/5번을 각각 한조씩 총 2개의 엔코더 카운팅을 실행합니다. 엔코더 카운터의 기능을 사용하기 위해서는 먼저 이 기능을 활성화 해주어야 합니다.

엔코더카운터는 32비트 부호있는 카운터값을 유지합니다. 아래에 상대회전엔코더의 A, B상 신호흐름에 대한 엔코더카운터의 카운팅방식을 타임그래프로 보여줍니다.

<그림> 엔코더카운터 동작



➲ Parameters

ch

채널(ch = 0 or 1)

bEnable

WIMC_TRUE : Enable 합니다.

WIMC_FALSE : Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

32.1.8. WIMC_Set_GIOUpCounter

Syntax

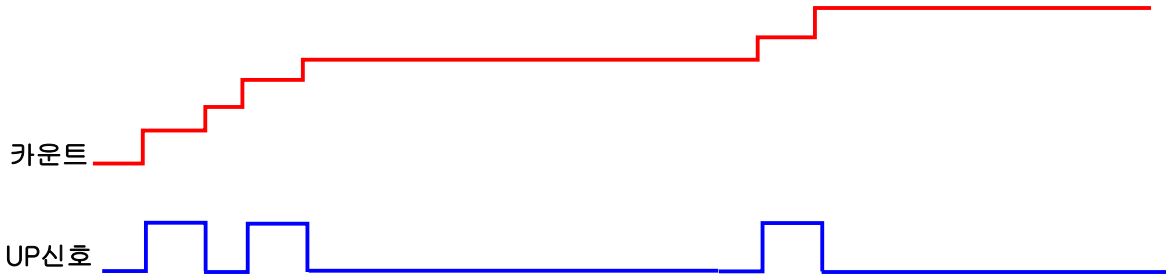
```
WIMC_BOOL     WIMC_API                WIMC_Set_GIOUpCounter
(
              WIMC_INT32              ch,
              WIMC_INT32              val
);
```

Description

업카운터의 현재카운터값을 설정합니다.

업카운터기능

입력포트 0번과 1번은 UP카운터 기능과 중첩됩니다. UP카운터로 사용할 경우 따로 설정해야하는 과정은 없습니다. UP카운터는 16비트카운터로써 입력신호 변화 때마다 증가되는 기능을 갖습니다. 그러므로 특정레벨로써 카운터할때보다 x2배값을 갖게 됩니다.



Parameters

ch

업카운터 채널번호(ch = 0,1)

val

부호없는 32비트로써 현재카운터값을 설정합니다.

Return Value

Return code/value	Description

Example

See Also

32.1.9. WIMC_Get_GIOUpCounter

Syntax

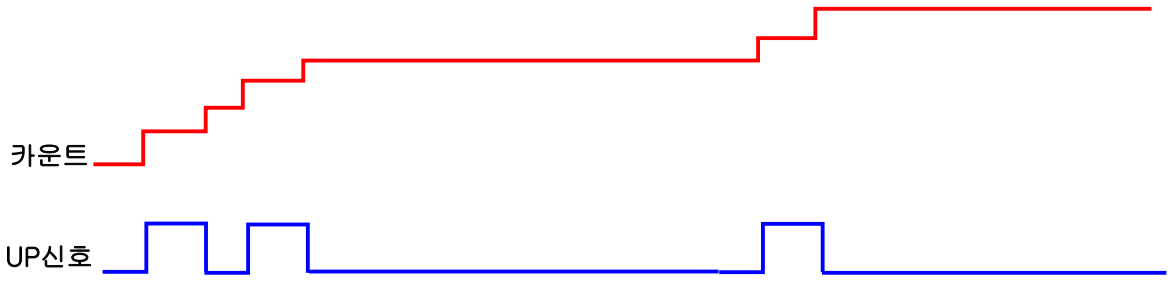
```
WIMC_INT32    WIMC_API                WIMC_Get_GIOUpCounter
(
              WIMC_INT32              ch
);
```

Description

설정된 업카운터의 현재카운터값을 읽습니다.

업카운터기능

입력포트 0번과 1번은 UP카운터 기능과 중첩됩니다. UP카운터로 사용할 경우 따로 설정해야하는 과정은 없습니다.
UP카운터는 16비트카운터로써 입력신호 변화 때마다 증가되는 기능을 갖습니다. 그러므로 특정레벨로써
카운터할때보다 x2배값을 갖게 됩니다.



Parameters

ch

업카운터 채널(ch = 0,1)

Return Value

설정된 업카운터 현재값을 리턴합니다.(0~65535)

Return code/value	Description

Example

See Also

32.1.10. WIMC_Set_GIOEncoder

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_GIOEncoder
(
              WIMC_INT32              ch,
              WIMC_INT32              val
);
```

➲ Description

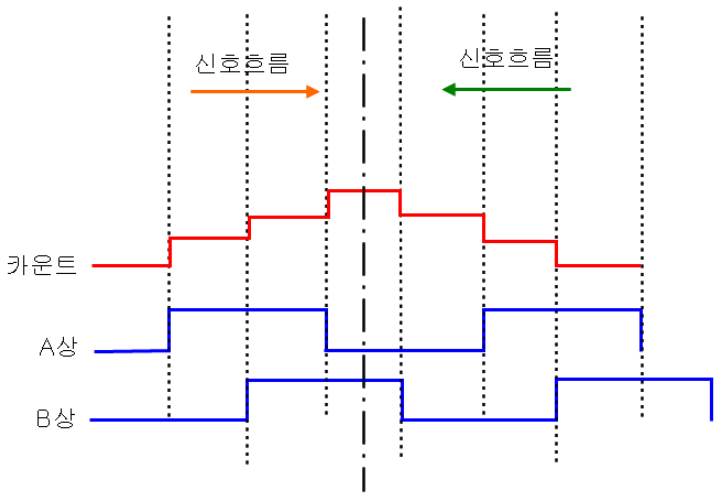
해당채널의 GIO용 엔코더 카운터값을 설정합니다.

❶ 엔코더 카운터 기능

엔코더 카운터는 A,B 위상차방식의 상대회전엔코더용 신호를 처리합니다. 기본적으로 4채배신호로 처리됩니다. WIMC 엔진의 엔코더 카운터는 입력 2번/3번 과 입력 4번/5번을 각각 한조씩 총 2개의 엔코더 카운팅을 실행합니다. 엔코더 카운터의 기능을 사용하기 위해서는 먼저 이 기능을 활성화 해주어야 합니다.

엔코더카운터는 32비트 부호있는 카운터값을 유지합니다. 아래에 상대회전엔코더의 A, B상 신호흐름에 대한 엔코더카운터의 카운팅방식을 타임그래프로 보여줍니다.

<그림> 엔코더카운터 동작



➲ Parameters

ch

채널(ch = 0,1)

val

설정할 엔코더 카운터 값입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

32.1.11. WIMC_Get_GIOEncoder

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_GIOEncoder
(
              WIMC_INT32              ch
);
```

➲ Description

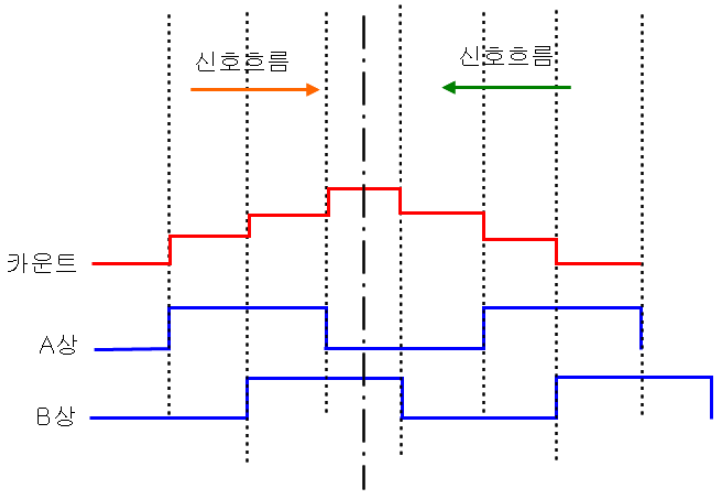
해당채널의 GIO용 엔코더 카운터 현재값을 읽습니다.

엔코더 카운터 기능

엔코더 카운터는 A,B 위상차방식의 상대회전엔코더용 신호를 처리합니다. 기본적으로 4채배신호로 처리됩니다. WIMC 엔진의 엔코더 카운터는 입력 2번/3번 과 입력 4번/5번을 각각 한조씩 총 2개의 엔코더 카운팅을 실행합니다. 엔코더 카운터의 기능을 사용하기 위해서는 먼저 이 기능을 활성화 해주어야 합니다.

엔코더카운터는 32비트 부호있는 카운터값을 유지합니다. 아래에 상대회전엔코더의 A, B상 신호흐름에 대한 엔코더카운터의 카운팅방식을 타임그래프로 보여줍니다.

<그림> 엔코더카운터 동작



➲ Parameters

ch

채널(ch = 0,1)

➲ Return Value

해당채널의 GIO용 엔코더 카운터 현재값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

32.1.12. WIMC_Get_ADC

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ADC
(
              WIMC_INT32              ch
);
```

➲ Description

해당 ch 번호의 ADC 값을 32비트 부호있는 정수 값으로 읽습니다.

ADC 전용 입력신호 기능

WIMC 엔진은 4채널의 10비트 축차비교형 아날로그 입력 전용포트(ADC: Analog to Digital Converter)를 내장합니다. ADC의 분해해상도는 10비트이지만 내부적으로 16비트로 확장되어 기록됩니다. 6비트가 시프트 됨으로 실제 최대표현값은 0xffc0 = 65472 가 됩니다. ADC의 샘플링 속도는 150SPS (Sampling per second) 로써 온도센서 등의 비교적 낮은 변화주기의 아날로그 신호값을 읽는데 사용할 수 있습니다.

➲ Parameters

ch

채널(ch= 0~3)

➲ Return Value

해당 ch 번호의 ADC 값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

32.1.13. WIMC_Set_PWM

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_PWM
(
              WIMC_INT32              ch,
              WIMC_INT32              PeriodCnt,
              WIMC_INT32              DutyCnt
);
```

➲ Description

해당 채널의 PWM 신호파형을 설정합니다.

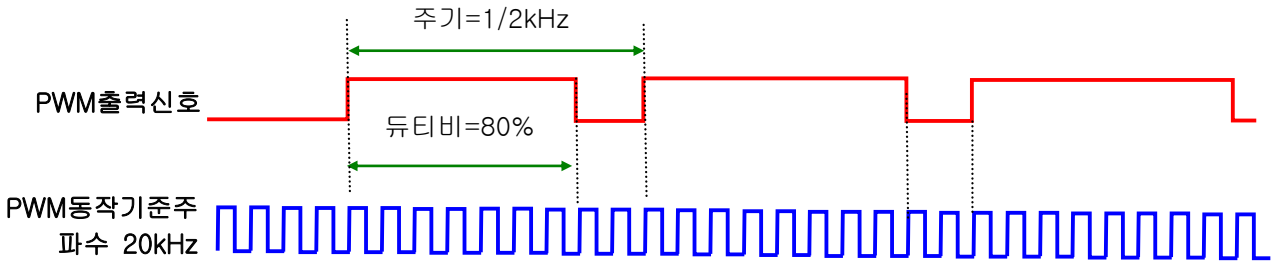
PWM 전용 출력신호 기능

WIMC 엔진은 PWM신호를 출력하는 2개의 전용포트를 갖습니다. PWM신호의 기준주파수는 20kHz입니다. ADC채널과 연결하여 온도제어용 출력 혹은 DC모터의 속도조절, PWM방식의 DAC회로를 이용하여 간단하게 아날로그 출력전압을 만드는 등의 용도로 사용될 수 있습니다. 또한 PID제어기의 출력으로 선택하여 사용될 수도 있습니다.

PWM신호를 발생하기 위해서 먼저 출력주기카운터와 ON출력카운터값을 설정합니다.
이 값을 각각 PeriodCount와 DutyCount라고 정의하면 PWM신호의 형태는 아래의 방정식으로 결정됩니다.

- PWM 신호 주기 = PeriodCount / 20kHz
- PWM Duty Ratio = 100 x DutyCount / PeriodCout [%]

<그림> PWM주기카운터 =10, PWM듀티카운터=8 일 때 PWM출력신호



➲ Parameters

ch

채널(ch = 0,1)

PeriodCnt

주기카운터 값입니다.

DutyCnt

튜티카운터 값입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

32.1.14. WIMC_Get_PWMPeriodCount

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_PWMPeriodCount
(
              WIMC_INT32              ch
);
```

➲ Description

해당 채널의 PWM 주기카운터값을 읽습니다.

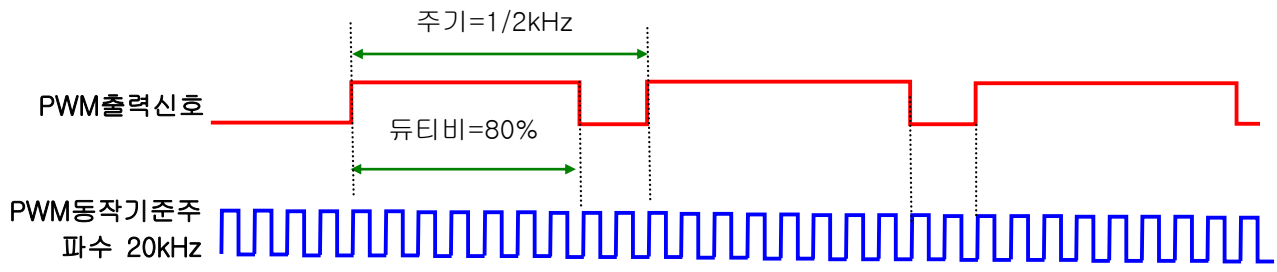
❶ PWM 전용 출력신호 기능

WIMC 엔진은 PWM신호를 출력하는 2개의 전용포트를 갖습니다. PWM신호의 기준주파수는 20kHz입니다. ADC채널과 연결하여 온도제어용 출력 혹은 DC모터의 속도조절, PWM방식의 DAC회로를 이용하여 간단하게 아날로그 출력전압을 만드는 등의 용도로 사용될 수 있습니다. 또한 PID제어기의 출력으로 선택하여 사용될 수도 있습니다.

PWM신호를 발생하기 위해서 먼저 출력주기카운터와 ON출력카운터값을 설정합니다.
이 값을 각각 PeriodCount와 DutyCount라고 정의하면 PWM신호의 형태는 아래의 방정식으로 결정됩니다.

- PWM 신호 주기 = PeriodCount / 20kHz
- PWM Duty Ratio = 100 x DutyCount / PeriodCout [%]

<그림> PWM주기카운터 =10, PWM듀티카운터=8 일 때 PWM출력신호



➲ Parameters

ch

채널(ch = 0,1)

➲ Return Value

해당 채널의 PWM 주기카운터값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

32.1.15. WIMC_Get_PWMDutyCount

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_PWMDutyCount
(
              WIMC_INT32              ch
);
```

➲ Description

해당 채널의 PWM 튜티 카운터값을 읽습니다.

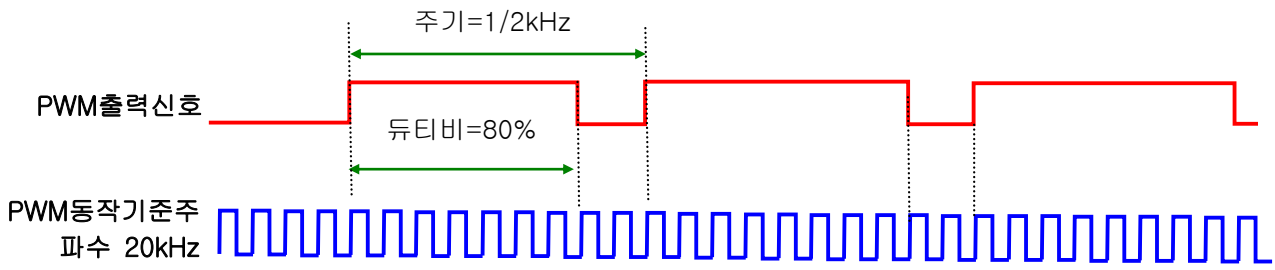
❶ PWM 전용 출력신호 기능

WIMC 엔진은 PWM신호를 출력하는 2개의 전용포트를 갖습니다. PWM신호의 기준주파수는 20kHz입니다. ADC채널과 연결하여 온도제어용 출력 혹은 DC모터의 속도조절, PWM방식의 DAC회로를 이용하여 간단하게 아날로그 출력전압을 만드는 등의 용도로 사용될 수 있습니다. 또한 PID제어기의 출력으로 선택하여 사용될 수도 있습니다.

PWM신호를 발생하기 위해서 먼저 출력주기카운터와 ON출력카운터값을 설정합니다.
이 값을 각각 PeriodCount와 DutyCount라고 정의하면 PWM신호의 형태는 아래의 방정식으로 결정됩니다.

- PWM 신호 주기 = PeriodCount / 20kHz
- PWM Duty Ratio = 100 x DutyCount / PeriodCout [%]

<그림> PWM주기카운터 =10, PWM듀티카운터=8 일 때 PWM출력신호



➲ Parameters

ch

채널(ch = 0,1)

➲ Return Value

PWM Duty Count 값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

33. EXTEND IO

33.1. 확장 입출력

확장 IO(EIO 혹은 ExtIO)보드 설정 및 동작과 관련된 함수들입니다. 확장 IO 보드에 대한 설명은 확장 IO 보드 사용자 매뉴얼을 참고 하실 수 있습니다.

33.1.1. WIMC_Enable_ExtIO

➲ Syntax

```
WIMC_BOOL      WIMC_API                WIMC_Enable_ExtIO
(
               WIMC_BOOL               bEnable
);
```

➲ Description

확장 입출력 보드(ELX36) 연결을 활성화 합니다.

➲ Parameters

bEnable

WIMC_TRUE : enable 설정합니다.

WIMC_FALSE : disable 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

```
// ExtIO 활성화
WIMC_Enable_ExtIO(TRUE); // 확장IO 연결
WIMC_Set_GIOMap( WIMC_X | WIMC_V ); // GIO IN/OUT으로 맵핑연결

// ExtIO 포트 출력
static UINT32           pin = 1 << 8;
static UINT8            n = 1;

WIMC_Enable_ExtIO(TRUE);
WIMC_Set_GIO(31, pin);
pin = pin << 1;
```

```
n++;
if (n > 12)
{
        pin = 1 << 8;
        n = 1;
}
```

➲ See Also

33.1.2. WIMC_Is_Enable_ExtIO

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Is_Enable_ExtIO(void);
```

➲ Description

확장 입출력 보드(ELX36) 활성화 상태를 확인합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	확장 입출력 보드 연결 enable 상태
WIMC_FALSE	확장 입출력 보드 연결 disable 상태

➲ Example

```
// ExtIO 활성화
WIMC_Enable_ExtIO(TRUE); // 확장IO 연결
WIMC_Set_GIOMap( WIMC_X | WIMC_V ); // GIO IN/OUT으로 맵핑연결

//WIMC_Set_GIOMap( WIMC_NONE ); // GIO IN/OUT으로 맵핑 해제

// ExtIO 포트 출력
static UINT32            pin = 1 << 8;
static UINT8             n = 1;

WIMC_Enable_ExtIO(TRUE);
WIMC_Set_ExtIO(4, pin);
pin = pin << 1;
n++;
if (n > 12)
{
        pin = 1 << 8;
        n = 1;
}
```

➲ See Also

33.1.3. WIMC_Set_GIOMap

➲ Syntax

```
WIMC_BOOL       WIMC_API                WIMC_Set_GIOMap
(
                WIMC_UINT32             mapAxis
);
```

➲ Description

확장 IO 보드의 입출력 단자로 맵핑 연결합니다.

➲ Parameters

mapAxis

WIMC_NONE : 매핑 연결을 해제합니다.

WIMC_X | WIMC_V : 확장 IO 보드 입출력 맵핑 연결

➲ Return Value

Return code/value	Description

➲ Example

```
// ExtIO 활성화
WIMC_Enable_ExtIO(TRUE); // 확장IO 연결
WIMC_Set_GIOMap( WIMC_X | WIMC_V ); // GIO IN/OUT으로 맵핑연결

// ExtIO 포트 출력
static UINT32           pin = 1 << 8;
static UINT8            n = 1;

WIMC_Enable_ExtIO(TRUE);
WIMC_Set_ExtIO(4, pin);
pin = pin << 1;
n++;
if (n > 12)
{
        pin = 1 << 8;
        n = 1;
}
```

➲ See Also

33.1.4. WIMC_Get_GIOMap

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Get_GIOMap(void);
```

➲ Description

확장 IO 보드의 입출력 단자로 맵핑 연결 설정을 확인 합니다.

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_NONE	GIO IN/OUT 맵핑 해제 상태
WIMC_X | WIMC_V	GIO IN/OUT 맵핑 연결 설정 상태

➲ Example

```
// ExtIO 활성화
WIMC_Enable_ExtIO(TRUE); // 확장IO 연결
WIMC_Set_GIOMap( WIMC_X | WIMC_V ); // GIO IN/OUT으로 맵핑연결

// ExtIO 포트 출력
static UINT32            pin = 1 << 8;
static UINT8             n = 1;

WIMC_Enable_ExtIO(TRUE);
WIMC_Set_ExtIO(4, pin);
pin = pin << 1;
n++;
if (n > 12)
{
        pin = 1 << 8;
        n = 1;
}
```

➲ See Also

33.1.5. WIMC_Enable_ExtServoIO

➲ Syntax

```
WIMC_BOOL      WIMC_API                WIMC_Enable_ExtServoIO
(
               WIMC_BOOL               bEnable
);
```

➲ Description

확장 Servo 입출력 기능을 설정(Enable/Disable) 합니다.

EIO 확장보드가 서보입출력 신호를 지원할 때 사용할 수 있습니다.

➲ Parameters

bEnable

WIMC_TRUE : enable 설정합니다.

WIMC_FALSE : disable 설정합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

33.1.6. WIMC_Is_ExtIO_Link0

➲ Syntax

```
WIMC_BOOL      WIMC_API                WIMC_Is_ExtIO_Link0
(
               WIMC_INT32              ch
);
```

➲ Description

EIO 확장보드가 연결된 EIO 채널의 최하위 byte(LSB)의 연결 유무를 확인합니다.

채널은 0~7 범위이며 하나의 채널은 32비트로 구성됩니다.
채널 0~3은 범용 입력채널이며, 채널 4~7은 범용 출력 채널입니다.
EIO 확장보드의 종류마다 임의 채널의 임의 비트 크기로 연결 될 수 있습니다.

➲ Parameters

ch
채널(ch = 0~7)

➲ Return Value

확장 IO Link0 상태를 반환합니다.

Return code/value	Description
WIMC_TRUE	연결상태입니다.
WIMC_FALSE	연결상태가 아닙니다.

➲ Example

➲ See Also

33.1.7. WIMC_Is_ExtIO_Link1

➲ Syntax

```
WIMC_BOOL      WIMC_API                WIMC_Is_ExtIO_Link1
(
               WIMC_INT32              ch
);
```

➲ Description

EIO 확장보드가 연결된 EIO 채널의 두번째 byte의 연결 유무를 확인합니다.

채널은 0~7 범위이며 하나의 채널은 32비트로 구성됩니다.
채널 0~3은 범용 입력채널이며, 채널 4~7은 범용 출력 채널입니다.
EIO 확장보드의 종류마다 임의 채널의 임의 비트 크기로 연결 될 수 있습니다.

➲ Parameters

ch
채널(ch = 0~7)

➲ Return Value

확장 IO Link1 상태를 반환합니다.

Description	Description
Link 상태 정상입니다.	Link 상태 정상입니다.
Link 상태 비정상입니다.	Link 상태 비정상입니다.

➲ Example

➲ See Also

33.1.8. WIMC_Is_ExtIO_Link2

➲ Syntax

```
WIMC_BOOL      WIMC_API                WIMC_Is_ExtIO_Link2
(
               WIMC_INT32              ch
);
```

➲ Description

EIO 확장보드가 연결된 EIO 채널의 세번째 byte의 연결 유무를 확인합니다.

채널은 0~7 범위이며 하나의 채널은 32비트로 구성됩니다.
채널 0~3은 범용 입력채널이며, 채널 4~7은 범용 출력 채널입니다.
EIO 확장보드의 종류마다 임의 채널의 임의 비트 크기로 연결 될 수 있습니다.

➲ Parameters

ch
채널(ch = 0~7)

➲ Return Value

확장 IO Link2 상태를 반환합니다.

Description	Description
Link 상태 정상입니다.	Link 상태 정상입니다.
Link 상태 비정상입니다.	Link 상태 비정상입니다.

➲ Example

➲ See Also

33.1.9. WIMC_Is_ExtIO_Link3

➲ Syntax

```
WIMC_BOOL      WIMC_API                WIMC_Is_ExtIO_Link3
(
               WIMC_INT32              ch
);
```

➲ Description

EIO 확장보드가 연결된 EIO 채널의 최상위 byte(MSB)의 연결 유무를 확인합니다.

채널은 0~7 범위이며 하나의 채널은 32비트로 구성됩니다.
채널 0~3은 범용 입력채널이며, 채널 4~7은 범용 출력 채널입니다.
EIO 확장보드의 종류마다 임의 채널의 임의 비트 크기로 연결 될 수 있습니다.

➲ Parameters

ch
채널(ch = 0~7)

➲ Return Value

확장 IO Link3 상태를 반환합니다.

Description	Description
Link 상태 정상입니다.	Link 상태 정상입니다.
Link 상태 비정상입니다.	Link 상태 비정상입니다.

➲ Example

➲ See Also

33.1.10. WIMC_Reset_ExtIO

➲ Syntax

```
WIMC_BOOL      WIMC_API                  WIMC_Reset_ExtIO( void);
```

➲ Description

설치된 EIO 확장보드에 리셋 명령을 실행합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

33.1.11. WIMC_EmgStop_ExtIO

➲ Syntax

```
WIMC_BOOL      WIMC_API                  WIMC_EmgStop_ExtIO( void);
```

➲ Description

설치된 EIO 확장보드에 비상정지 명령을 실행합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

33.1.12. WIMC_Restart_ExtIO

➲ Syntax

```
WIMC_BOOL      WIMC_API                  WIMC_Restart_ExtIO( void);
```

➲ Description

설치된 EIO 확장보드에 재시작 명령을 실행합니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

33.1.13. WIMC_Set_ExtIO

➲ Syntax

```
WIMC_BOOL       WIMC_API                WIMC_Set_ExtIO
(
                WIMC_INT32              ch,
                WIMC_UINT32             val
);
```

➲ Description

EIO 확장보드 출력채널(4~7)의 포트값을 설정합니다.

➲ Parameters

ch

채널(ch=4~7)

val

확장 IO 설정값입니다.

➲ Return Value

Return code/value	Description

➲ Example

```
// ExtIO 활성화
WIMC_Enable_ExtIO(TRUE); // 확장IO 연결
WIMC_Set_GIOMap( WIMC_X | WIMC_V ); // GIO IN/OUT으로 맵핑연결

// ExtIO 포트 출력
static UINT32           pin = 1 << 8;
static UINT8            n = 1;

WIMC_Enable_ExtIO(TRUE);
WIMC_Set_ExtIO(4, pin);
pin = pin << 1;
n++;
if (n > 12)
{
        pin = 1 << 8;
        n = 1;
```

```
}
```

➲ See Also

33.1.14. WIMC_Get_ExtIO

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Get_ExtIO
(
               WIMC_INT32              ch
);
```

➲ Description

EIO 확장보드 입력 및 출력 채널(0~7)의 포트값을 읽습니다.

➲ Parameters

ch

채널(ch = 0~7)

➲ Return Value

설정된 확장 IO 값을 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

33.1.15. WIMC_Get_ExtIO_MPG_Status

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Get_ExtIO_MPG_Status( void);
```

➲ Description

MPG 기능을 갖는 EIO 확장보드의 MPG(Manual Pulse Generator) 상태값(16비트)을 읽습니다.

➲ Parameters

➲ Return Value

확장 IO MPG(Manual Pulse Generator) 상태를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

33.1.16. WIMC_Get_ExtIO_MPG_Count

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_ExtIO_MPG_Count( void);
```

➲ Description

MPG 기능을 갖는 EIO 확장보드의 MPG(Manual Pulse Generator) 현재 카운터 값(16비트)을 읽습니다.

➲ Parameters

➲ Return Value

확장 IO MPG(Manual Pulse Generator) 카운터 값을 반환합니다.(0~65535)

Return code/value	Description

➲ Example

➲ See Also

34. 비교기

34.1. 위치비교기

WIMC 엔진은 독립된 4개의 위치비교기를 내장합니다. 위치비교기의 동작은 10kHz이며 공통된 단일의 출력신호인 TrigOut신호를 통해 비교결과를 출력합니다. 위치비교기는 현재 모션위치가 지정된 위치보다 크거나(>=) 혹은 작을 때(<=) 신호를 발생시키는 역할을 합니다. 위치비교기의 동작주파수가 10kHz이므로 고속이송동작에서 펄스를 발생시키는 시점의 위치가 다소 차이가 발생할 수 있습니다. 초당 1M로 이송하는 장비의 경우 0.1mm 오차로 신호를 출력합니다.

정해진(미리 알고있는) 위치에서 특정신호를 출력하는 기능은 위치비교기기능보다는 바인딩출력기능을 갖는 GIO함수를 사용하는 것이 정확하며 편리합니다. 이들 함수에 자세한 내용은 “바인딩된 범용출력명령” 편을 참조하실 수 있습니다.

위치비교기능은 정해지지 않고 동작중 실시간으로 비교위치가 변경할 경우에 적합합니다. 또한 위치비교기는 윈도우 메시지를 발생할 수 있습니다.
위치비교기 동작은 크게 단 1회의 비교동작을 위한 단순위치비교기능과 일정거리간격으로 반복하여 위치를 비교하는 반복위치비교기능으로 나눕니다.

34.1.1. WIMC_Enable_Comparator

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Enable_Comparator
(
              WIMC_INT32              ch,
              WIMC_BOOL               bEnable
);
```

➲ Description

채널(ch)로 지정된 비교기를 설정(enable/disable)합니다.

❶ 단순위치 비교기능

독립된 4개의 위치비교기를 개별적으로 사용할 수 있습니다. WIMC 엔진의 동작축은 6개이지만 비교기는 4개만을 갖습니다. 위치비교기는 채널번호로써 0~3번까지 존재합니다.
단순 위치 비교기능은 특정위치를 지정한 후 현재위치가 지정된 위치보다 크거나 혹은 작을 때 신호를 발생할 수 있습니다.
발생되는 신호는 WIMC 엔진의 TrigOut신호 와 함께 윈도우 메시지를 설정할 수 있습니다. 아래의 함수를 통해 각 기능들을 살펴봅니다.

➲ Parameters

ch
channel = 0~3

bEnable
WIMC_TRUE :비교기 기능 Enable.
WIMC_FALSE : 비교기 기능 Disable.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

34.1.2. WIMC_Register_Comparator_Message

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Register_Comparator_Message
(
             HWND                    h,
             WIMC_UINT32             message
);
```

➲ Description

위치비교기능에서 발생할 이벤트의 윈도우 핸들과 메시지번호를 지정합니다.

아래의 메시지 파라메터에 의해 발생된 메시지가 어떤 내용인지를 확인할 수 있습니다.

`MSG msg; // 메시지 구조체`

<msg.lParam의 비트구조>

비트번호	31~16	15~8	7~0
기능	없음	이벤트발생채널	이벤트번호
내용	없음	1ch=0000 0001 2ch=0000 0010 3ch=0000 0100 4ch=0000 1000 5ch=0001 0000 6ch=0010 0000 7ch=0100 0000 8ch=1000 0000	25 (십진수)

➲ Parameters

h

윈도우 핸들 값입니다.

message

발생시키고자 하는 메세지를 `WIMC_UINT32` 타입으로 명시합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

34.1.3. WIMC_Unregister_Comparator_Message

➲ Syntax

```
WIMC_BOOL      WIMC_API                  WIMC_Unregister_Comparator_Message( void);
```

➲ Description

비교기 메시지 등록을 삭제합니다.

WIMC_Register_Comparator_Message() 함수를 참고 하실 수 있습니다.

➲ Parameters

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

34.1.4. WIMC_Set_PosComparator

➲ Syntax

```
WIMC_BOOL    WIMC_API                WIMC_Set_PosComparator
(
             WIMC_INT32              ch,
             WIMC_UINT8              compareMode,
             WIMC_UINT8              axis,
             WIMC_INT32              comparePos,
             WIMC_UINT8              trigMode,
             WIMC_BOOL               enableEvent
);
```

➲ Description

단순비교기능을 설정합니다.

➲ Parameters

ch

채널 (ch= 0,1,2,3)

compareMode

WIMC_COMPARE_MODE_GE : >= 보다 크거나 같을 때입니다.

WIMC_COMPARE_MODE_LE : <= 보다 작거나 같을 때입니다.

axis

적용축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

comparePos

비교위치값입니다.

trigMode

트리거 신호 모드로 다음과 같이 3가지가 있습니다.

WIMC_COMPARE_TRIGMODE_DISABLE (트리거신호출력 사용안함)

WIMC_COMPARE_TRIGMODE_LEVEL (트리거신호출력방식은 레벨방식)

WIMC_COMPARE_TRIGMODE_PULSE (트리거신호출력방식은 펄스방식)

LEVEL모드는 비교조건이 만족될 때 신호가 발생(활성화)되며, 비교조건이 만족되지 않을 때 신호가 비활성화 됩니다.
PULSE모드는 비교조건이 만족되는 순간 1회 펄스가 발생됩니다. 펄스 주기는 스캔타임x(1~2)배입니다.

enableEvent

WIMC_TRUE : 비교조건이 만족될 때 Event 패킷을 전송하여 메세지를 발생 할 수 있습니다.

WIMC_FALSE : Event 패킷전송을 Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

34.1.5. WIMC_Set_PosComparatorX

➲ Syntax

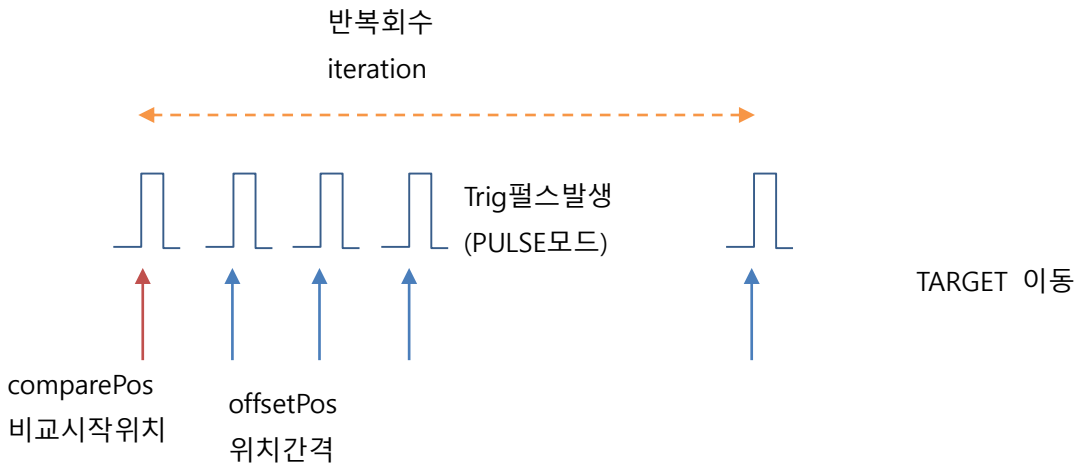
```
WIMC_BOOL    WIMC_API              WIMC_Set_PosComparatorX
(
             WIMC_INT32            ch,
             WIMC_UINT8            compareMode,
             WIMC_UINT8            axis,
             WIMC_INT32            comparePos,
             WIMC_UINT8            trigMode,
             WIMC_INT32            offsetPos,
             WIMC_UINT32           iteration,
             WIMC_BOOL             enableEvent
);
```

➲ Description

반복위치 비교기능입니다.

반복위치 비교기능은 비교위치를 특정위치에서부터 일정한 위치간격만큼 반복하여 지정할 수 있습니다.
반복위치 비교기능은 2개의 독립된 비교기를 이용하여 동작됨으로 WIMC 엔진은 최대 2채널까지만 사용 가능합니다.
채널0과 채널2번만을 사용합니다.

offsetPos인자와 iteration인자를 제외한 나머지 인자들은 단순비교기능과 동일하지만 trig펄스발생모드를 PULSE 모드로 지정하는 것이 바람직합니다.
아래 그림에서 보는바와 같이 offsetPos는 비교시작위치로부터 일정한 간격거리를 말하며 iteration는 offsetPos의 횟수를 말합니다. 비교기의 동작속도가 10kHz이므로 offsetPos의 간격이 너무 좁게 되면 정확한 펄스를 발생할 수 없게 됩니다.



➲ Parameters

ch
채널 (ch= 0,1,2,3)

compareMode

WIMC_COMPARE_MODE_GE : >= 보다 크거나 같을 때입니다.
WIMC_COMPARE_MODE_LE : <= 보다 작거나 같을 때입니다.

axis
적용축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

comparePos
비교위치값입니다.

trigMode
트리거 신호 모드로 다음과 같이 3가지가 있습니다.

WIMC_COMPARE_TRIGMODE_DISABLE (트리거신호출력 사용안함)
WIMC_COMPARE_TRIGMODE_LEVEL (트리거신호출력방식은 레벨방식)
WIMC_COMPARE_TRIGMODE_PULSE (트리거신호출력방식은 펄스방식)

LEVEL모드는 비교조건이 만족될 때 신호가 발생(활성화)되며, 비교조건이 만족되지 않을 때 신호가 비활성화 됩니다.
PULSE모드는 비교조건이 만족되는 순간 1회 펄스가 발생됩니다. 펄스 주기는 스캔타임x(1~2)배입니다.

offsetPos
위치간격을 설정합니다.

iteration
반복횟수를 설정합니다.

enableEvent
WIMC_TRUE : 비교조건이 만족될 때 Event 패킷을 전송하여 메세지를 발생 할 수 있습니다.
WIMC_FALSE : Event 패킷전송을 Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

35. 캡쳐기능

35.1. 위치 캡쳐기능

모션 라이브러리V5.14 버전 이후부터 사용 가능합니다.

위치 캡쳐 기능은 입력단자와 축을 지정하고 기능 활성화를 하면,
신호 변화를 감지와 동시에 해당축 위치값을 내부 버퍼에 저장합니다.
캡쳐 데이터 확인 함수를 이용하여 저장된 내부버퍼 데이터(위치 값)를 사용 할 수 있습니다.

1.관련 함수

1.1) WIMC_Enable_CaptureMode (WIMC_BOOL enable, WIMC_UINT32 mode = 0);
캡쳐모드 활성화 및 입력 모드 설정

1.2) WIMC_Is_CaptureMode (void);
캡쳐모드 활성화 상태 확인

1.3) WIMC_Init_Capture (void);
캡쳐데이터 초기화

1.4) WIMC_Get_CaptureCnt (void);
캡쳐데이터 카운터 확인

1.5) WIMC_Get_Capture (int n, int axisno);
캡쳐 데이터 확인

2.캡쳐기능 활성화 및 모드 설정

WIMC_Enable_CaptureMode(enable, mode);

>enable: 활성화(1), 비활성화(0)

>mode: Sync 입력(0), GIN0 입력(1), GIN1 입력(2), GIN0 or GIN1 입력(3)

3.캡쳐데이터 확인

WIMC_Get_Capture(n, axisno);

>n : 캡쳐데이터 카운터

>axisno : 축번호(WIMC_AXISNO_X , Y,...)

35.1.1. WIMC_Enable_CaptureMode

➲ Syntax

```
WIMC_UINT32   WIMC_API                WIMC_Enable_CaptureMode
(
              WIMC_BOOL               enable,
              WIMC_UINT32             mode
);
```

➲ Description

위치캡쳐 기능을 활성화 합니다.
캡쳐기능을 위한 입력 단자를 선택합니다(Sync 입력(0), GIN0 입력(1), GIN1 입력(2), GIN0 or GIN1 입력(3))

➲ Parameters

enable

위치캡쳐 기능을 활성화 합니다.

mode

0 : sync 입력 선택(기본값)
1 : GIN0 입력 선택
2 : GIN1 입력 선택
3 : GIN0 또는 GIN1 입력 변화 감지 선택

➲ Return Value

enable 상태를 반환합니다.

Return code/value	Description
1	enable
0	

➲ Example

➲ See Also

35.1.2. WIMC_Is_CaptureMode

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_CaptureMode()
```

➲ Description

캡쳐기능 활성화 상태를 확인합니다..

➲ Parameters

➲ Return Value

Return code/value	Description
WIMC_TRUE	캡쳐기능이 활성화 중입니다.
WIMC_FALSE	

➲ Example

➲ See Also

35.1.3. WIMC_Init_Capture

➲ Syntax

```
WIMC_UINT32   WIMC_API                  WIMC_Init_Capture ()
```

➲ Description

캡쳐기능을 초기화 합니다.(내부 카운터 및 버퍼)

➲ Parameters

➲ Return Value

캡쳐 내부버퍼 최대값을 반환합니다.(기본값 10000)

Return code/value	Description

➲ Example

➲ See Also

35.1.4. WIMC_Get_CaptureCnt

➲ Syntax

```
WIMC_INT32    WIMC_API                  WIMC_Get_CaptureCnt ()
```

➲ Description

캡쳐된 값의 개수를 확인합니다.

➲ Parameters

➲ Return Value

캡쳐된 값의 개수를 반환합니다.

Return code/value	Description

➲ Example

➲ See Also

35.1.5. WIMC_Get_Capture

➲ Syntax

```
WIMC_INT32    WIMC_API                WIMC_Get_Capture
(
              WIMC_INT32              n,
              WIMC_INT32              axis
);
```

➲ Description

n번째 위치캡쳐 데이터값을 읽습니다.

➲ Parameters

n

캡쳐데이터 내부버퍼 인덱스

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

캡쳐데이터 내부버퍼의 n번째 데이터 값을 반환합니다.

Return code/value	Description

➲ Example

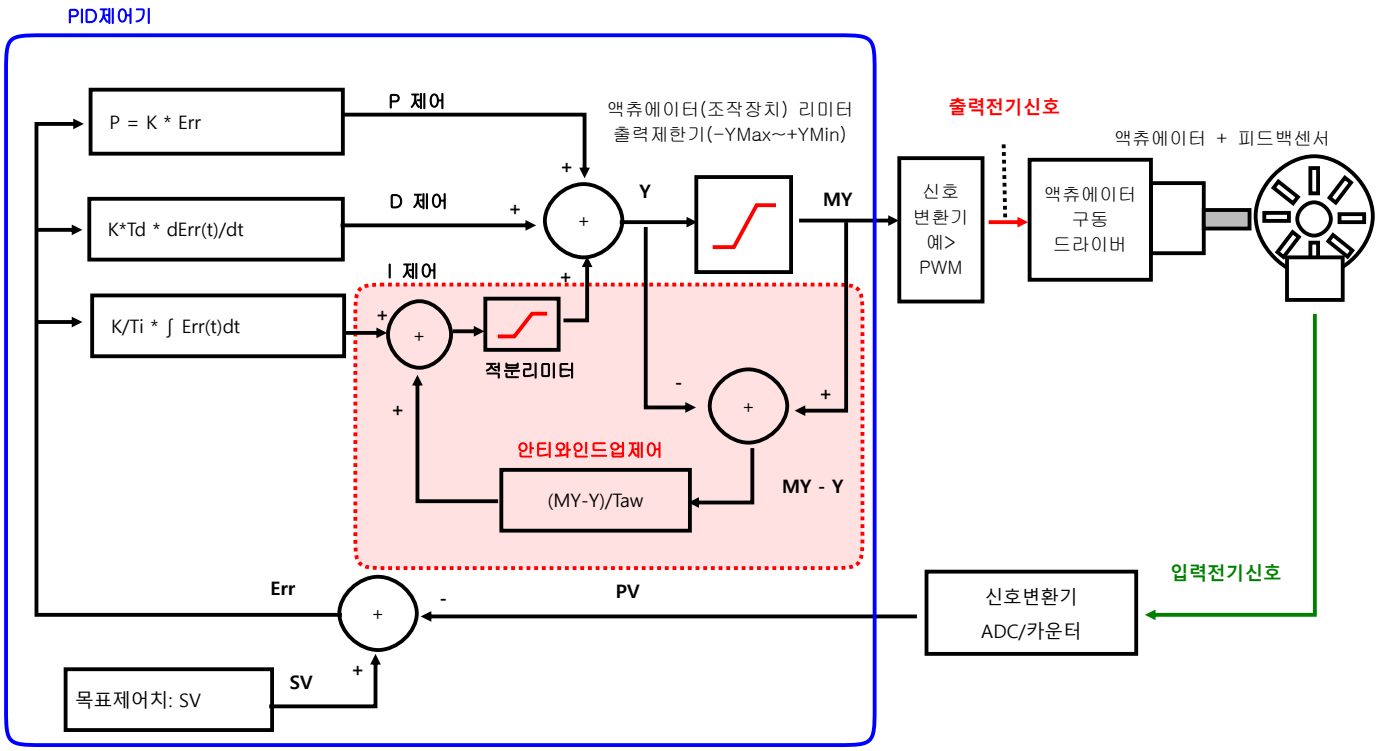
➲ See Also

36. PID

36.1. PID 제어기

WIMC 엔진은 제어주기가 100Hz이하인 온도 및 모터제어 등의 간단한 피드백제어로 사용될 수 있는 PID제어기 4채널을 내장합니다. 옵션에 따라, ON/OFF제어, P제어, PI제어, PD제어 등도 가능합니다.
여기서는 WIMC 엔진의 PID제어기 파라메터와 기능들, 그리고 관련 라이브러리 함수가 소개됩니다.
"모션마스터"프로그램을 통한 파라메터설정과 모니터링관련 내용은 프로그램설명서를 참조해 주세요.
(먼저 아래의 내용들은 PID제어에 관련된 기본적인 지식이 필요합니다.)

먼저 WIMC 엔진에 내장된 PID제어기의 기본동작에 관련된 제어흐름을 살펴보면 아래와 같습니다.
<WIMC 엔진 모션엔진 PID제어기 기본 동작 다이어그램>



그림과 같이 일반적인 PID제어기의 구성을 갖고 있습니다. 간단하게 PID제어의 기본동작과 각 제어블록에 대해서 설명합니다.
PID제어는 목표값과 피드백센서값과의 오차를 대상으로 P제어(비례제어), I제어(적분제어), D제어(미분제어)의 합성에 의해 조작값(액추에이터동작값)을 출력하게 됩니다.
만약 출력값이 모터구동장치와 같은 드라이브가 수용할 수 있는 값범위를 벗어 날 경우 출력제한기를 통해 최종조작값은 최소값과 최대값범위내로 조절됩니다.
비례제어량과 미분제어량은 대부분 특정범위내로 동작되지만, 적분제어량은 오차를 시간으로 적분하게 되므로 과도한 제어량이 발생할 수 있습니다. 대부분 제어초기에 목표값과의 오차가 크고 목표값에 도달하기 위해 시간이 많이 필요할 때 적분제어량의 과도하게 커집니다. WIMC 엔진은 적분오차가 너무 커지는 것을 제한하기 위해 안티와인드업제어 와 적분리미터를 두어 적분량을 제한하여 과도한 제어응답이 발생하지 않도록 조절합니다.

WIMC 엔진의 PID제어기는 기본적으로 ADC입력을 피드백 센서값으로 취하고, PWM출력포트와 GOut포트(범용출력포트)를 이용하는 PWM출력을 조작값으로 택하고 있습니다.

또한 독립된 4개의 PID제어기는 아래의 라이브러리 함수를 통해 제어됩니다. 모션엔진이 리셋되었을 때 모든 PID제어기의

기능은 꺼지게 됩니다.

(1) WIMC_Off_PID(ch); // ch(0~3)로 지정된 PID제어기 기능을 OFF합니다.

(2) WIMC_Reset_PID(ch); // ch(0~3)로 지정된 PID제어기의 초기화 합니다.

(3) WIMC_Setup_PID(ch, ptPid);

// ch(0~3)로 지정된 PID제어기를 구조체포인터변수 ptPid가 지정하는 pid구조체로 셋업합니다.

// 이 구조체의 파라메터값에 대해서 다음 절에서 설명합니다.

(4) WIMC_Run_PID(ch); // ch(0~3)로 지정된 PID제어기를 동작시킵니다.

(5) WIMC_Set_PIDSV(ch, val); // 제어 목표량을 재설정합니다.

// val 은 16비트 부호없는 정수입니다.

(6) WIMC_Is_PID_On(ch); // PID제어기의 ON를 확인합니다.

(7) WIMC_Get_PID_Setup(ch, ptPid) // ch(0~3)로 지정된 PID제어기의 설정들을 읽어들입니다.

// ch(0~3)로 지정된 PID제어기의 설정값들을 구조체포인터변수 ptPid가 지정하는 pid구조체로 읽습니다.

함수에 대한 사용은 아래의 순서를 따릅니다.

① PID제어기 리셋 : WIMC_Reset_PID(ch);

② PID제어기 셋업 : WIMC_Setup_PID(ch, ptPid);

③ PID제어기 동작 : WIMC_Run_PID(ch);

④ 필요시> PID제어기 목표값 재설정 : WIMC_Set_PIDSV(ch, val);

⑤ PID제어기 정지 : WIMC_Off_PID(ch);

여기서 가장 중요한 부분이 PID제어기의 셋업동작입니다. WIMC_Setup_PID()함수를 통해서 PID제어에 관련된 모든 파라메터값들을 설정합니다.

PID 제어기 Setup

PID제어기가 올바르게 작동하기 위해서는 PID제어기의 다양한 파라메터값들을 설정해야만 합니다.

이는 라이브러리 함수 WIMC_Setup_PID()를 통해서 이루어집니다.

<관련 라이브러리함수>

```
WIMC_Setup_PID( int ch, WIMC_PID_t *ptPid);
```

이 함수로 전달되는 두가지 파라메터는 각각 PID채널 번호와 PID제어기 구조체입니다.

여기에는 PID제어기 구조체의 요소들과 기능에 대해 살펴봅니다.

▶ PID 구조체 원형

```
typedef struct tag_WIMC_PID_t
{
        UINT16 SV;       // 제어목표값
        UINT16 K;        // 비례대
        UINT16 Ti;       // 적분시간
        UINT16 Td;       // 미분시간
        UINT16 Tc;       // 제어주기시간
        UINT16 Taw;      // 안티리완이드시간
        UINT16 Hys;      // 불감대역
        UINT16 AL;       // Low 알람
        UINT16 AH;       // High 알람
        UINT8 InputId;   // 피드백센서입력 번호
        UINT8 OutputId;  // 조작출력 번호
```

```
    UINT8 OutputMode;// 출력방식
    UINT8 PolarOutport;// 극성출력 GOut포트
    INT32 Extra;     // 추가 32비트 파라메터 예약
    UINT16 IL;       // 적분리미터
    UINT16 YMax;     // 최대/최소 조작출력값의 절대치
} WIMC_PID_t;
```

아래에 각 구조체 인자들의 기능들을 설명합니다.

▶ SV :제어목표값 (0~65535)

16비트 부호없는 정수값으로 단위는 없습니다.
제어목표값으로써 입력센서(PV)가 PID제어로써 도달하고자 하는 값을 설정합니다.

▶ K : 비례대 (0~65535)

16비트 부호없는 정수값으로 단위는 SV와 같이 없습니다.
비례대값은 P제어 출력이 100%가 출력되는 센서값PV와 목표제어SV과의 오차값(Err)이 됩니다.
센서와 목표치제어 값은 0~65535로써 16비트 부호없는 정수값을 기준으로 합니다.
그러므로 K = 1000 이면 오차값 Err가 1000될 때 비례제어출력 값이 100%에 도달합니다.
K값이 작을수록 비례이득이 커집니다. (다이어그램의 K값과 다르며 비율 변환되어 사용됩니다.)

※ K 값이 0 이면 PID 제어는 ON/OFF 제어로 전환됩니다.

▶ Ti : 적분시간 (Reset Time) (0~65535msec)

16비트 부호없는 정수값으로 단위는 msec입니다.
비례제어가 수렴했을 때 발생하는 Static Err(정적오차)값이 리셋되도록 적분제어를 실행하는 이득값과 관련됩니다. 제어주기 Tc보다는 커야 합니다(작으면 제어는 진동합니다). Ti값이 작을수록 적분제어의 이득값이 커져 보다 빠르게 정적오차를 제거하지만, 시스템이 진동할 수 있습니다.

※ Ti 값이 0 이면 적분제어는 동작하지 않습니다.

▶ Td : 미분시간 (0~65535msec)

16비트 부호없는 정수값으로 단위는 msec입니다.
외부의 외란(급작스러운 외부변화:노이즈, 충격, 챔버문열림/닫힘등)에 적응하는 능력을 가진 미분제어를 실행하는 이득값과 관련됩니다. Td값은 클수록 미분제어영향이 커집니다.

※ Td 값이 0 이면 미분제어는 동작하지 않습니다.

▶ Tc : 제어주기(Control period) (0~65535msec)

16비트 부호없는 정수값으로 단위는 msec입니다.
PID제어주기로써 지정된 시간단위로 연산이 이루어지며 출력도 이 주기로 발생됩니다. 이 값은 제어시스템에서 가장 먼저 선택되어야하는 파라메터로써 제어시스템이 동특성이 얼마나 빨리 응답하는가에 대한 기본적인 내용을 포함합니다.
디지털제어에서는 기본적으로 샘플링시간과 연관됩니다. 만약 제어속도가 빠르지 않는 온도제어시스템이다면 Tc는 대부분 0.1초 이상일 것입니다.

▶ Taw : 안티와인드업(Anti Windup) (0~65535msec)

16비트 부호없는 정수값으로 단위는 msec입니다.

제어출력값 Y가 액추에이터의 조작값MY 범위(-100%~+100%)를 초과할 때 적분동작을 제한하는 기능을 합니다.

적분동작을 제한하지 않으면 적분량이 지나치게 커져서 목표값에 도달하더라도 적분오차가 사라지기 전까지 출력상태는 거의 변화하지 않게 됩니다. 안티와인드업동작은 이러한 과적분에 의한 오버슈트(Overshoot)나 언더슈트(Undershoot)현상을 줄이는 기능을 합니다.

※ Taw 값이 0 이면 안티와인드업은 동작하지 않습니다.

▶ Hys : 히스테리시스(Hysterisis / Zero Margine : 불감대역) (0~65535)

16비트 부호없는 정수값으로 단위는 SV와 같이 없습니다.

센서의 노이즈나 출력동작의 미세한 진동등에 대해서 PID제어기가 동작변화를 크게 일으키지 않는 불감대영역을 결정하는 역할 합니다.

주로 센서의 기준입력값인 0~65535값 내에서 센서의 불안정한 비트들의 최대값으로 설정할 수 있습니다.

예를 들면 WIMC 엔진은 내부적으로 10비트 해상도를 갖는 ADC가 내장되었지만 출력은 16비트로 확장되어 있습니다.

그러므로 6비트값은 불안정한 값을 가질 수 있으므로 Hys = 2^6 = 32값으로 지정할 수 있습니다.

※ Hys 값이 0 이면 히스테리시스는 동작하지 않습니다.

▶ AL : 출력강제 Off 로레벨 (0~65535)

16비트 부호없는 정수값으로 단위는 SV와 같이 없습니다.

센서값(PV) 이 AL값 이하일 때 출력을 0%로 차단하는 역할을 합니다.

※ AL 값이 0 이면 동작하지 않습니다.

▶ AH : 출력강제 Off 하이레벨 (0~65535)

16비트 부호없는 정수값으로 단위는 SV와 같이 없습니다.

센서값(PV) 이 AH값 이상일 때 출력을 0%로 차단하는 역할을 합니다.

※ AH 값이 0 이면 동작하지 않습니다.

▶ InputId : 피드백센서연결 id

8비트 부호없는 정수값으로 지정된 값만 선택 가능합니다.

PV값과 연결되는 센서입력장치의 종류를 선택합니다. 모션엔진마다 각 값에 대한 연결장치의 종류가 다를 수 있습니다. WIMC 엔진은 아래와 같습니다.

① 0,1,2,3 : (기본적인 범위) 각각 ADC 채널0,1,2,3과 연결됩니다.

② 24, 25 : 각각 GIO 엔코더카운터 채널 0, 1과 연결됩니다.

▶ OutputId : 조작출력연결 id

8비트 부호없는 정수값으로 지정된 값만 선택 가능합니다.

모션엔진마다 각 값에 대한 연결장치의 종류가 다를 수 있습니다. WIMC 엔진은 아래와 같습니다.

① 0, 1 : 각각 PWM채널 0, 1과 연결됩니다.

② 2 : GIO 출력포트 bit2를 통해서 PWM신호로 연결됩니다.

3 : GIO 출력포트 bit3를 통해서 PWM신호로 연결됩니다.

4 : GIO 출력포트 bit4를 통해서 PWM신호로 연결됩니다.

5 : GIO 출력포트 bit5를 통해서 PWM신호로 연결됩니다.

▶ Output Mode : 출력모드

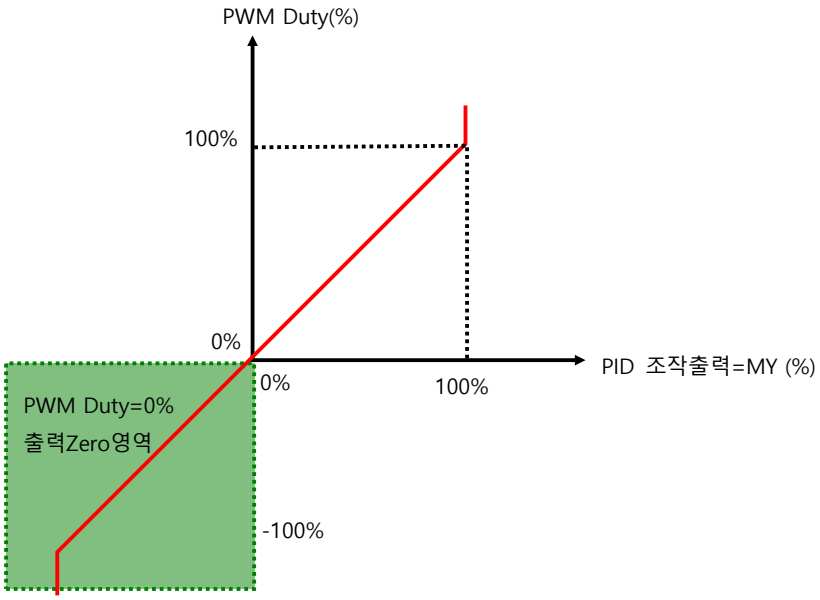
8비트 부호없는 정수값으로 지정된 값만 선택 가능합니다. (지정값=0,1,2,4,5,6)

PID제어기는 내부적으로 출력MY값이 -100.00%~+100.00%값을 출력합니다.

아래의 6가지 값방식 중 하나를 선택할 수 있습니다.

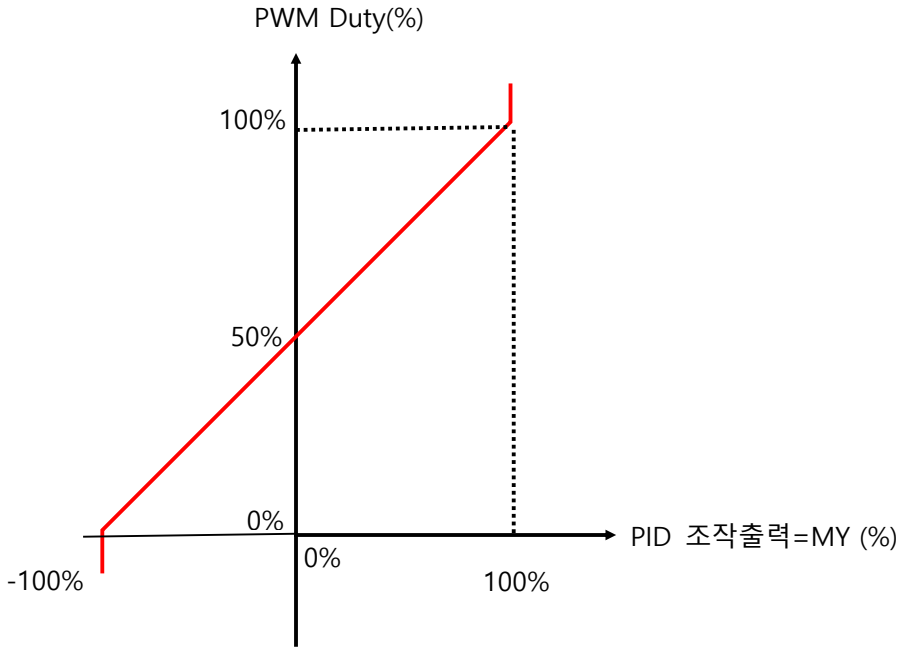
<0 방식>

PID제어기 출력중 0~100%만을 PWM의 출력(Duty)에 사용합니다.

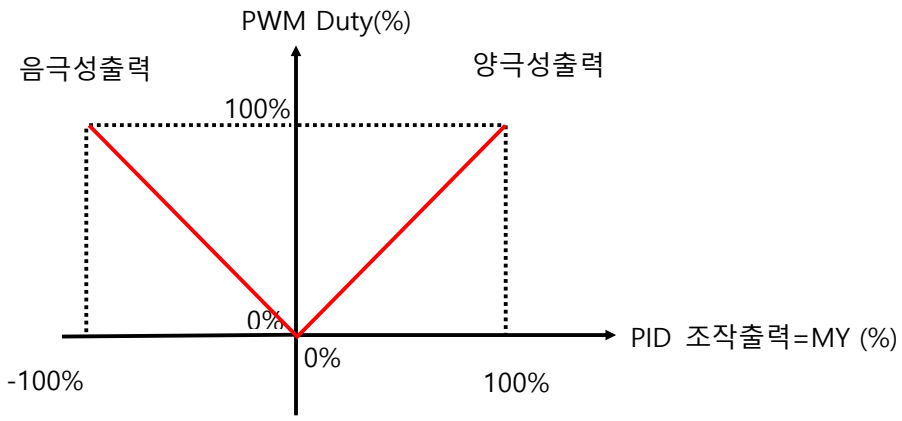


<1 방식>

PID제어기 출력 -100~100%를 PWM의 출력(Duty) 0~100%로 스케일링하여 사용합니다.



<2 방식>
별도의 극성출력포트를 설정하고 PID출력(-100 ~ +100%)의 절대값을 취해서 PWM 출력(Duty)값으로 사용합니다.
별도의 극성출력포트설정은 다음에 나오는 PID파라메터인 "PWM Polar outport"설정을 참조하세요.



<4 방식>
1 방식의 출력반전 방식입니다.
0%출력이면 100%로 출력되며, 100%출력이면 0%로 출력됩니다.
<5 방식>
2 방식의 출력반전 방식입니다.
0%출력이면 100%로 출력되며, 100%출력이면 0%로 출력됩니다.
<6 방식>
3 방식의 출력반전 방식입니다.
-100%출력이면 +100%로 출력되며, +100%출력이면 -100%로 출력됩니다.

▶ PolarOutport : 극성 출력포트 지정
8비트 부호없는 정수값입니다. (0~255)
PID 출력모드가 2일 때 극성출력을 필요로 합니다. 이때 극성출력을 GIO출력포트 중 임의로 지정할 수 있습니다. 음극성 신호(PID제어기 출력값이 -값)일 때 출력이 1(H레벨)이 됩니다. 양극성 신호(PID제어기 출력값이 +값)일 때 출력은 0(L레벨)이 됩니다.
WIMC 엔진은 GIO출력포트가 8개 존재하므로 0~7 중 하나를 선택합니다.

※ 출력포트지정값이 0~31 의 범위가 아닐 때 극성출력은 동작하지 않습니다.

▶ IL : 적분리미터
부호없는 16비트 정수값으로 단위는 %입니다. 내부적으로 최대 1000까지 값만 허용됩니다. (0~1000%)
제어시스템에 적합하지 않은 제어파라메터의 설정 혹은 외부외란에 의한 급작스런 상황변경등에서 오차Error가 커질 경우 적분량이 급수적으로 증가할 수 있습니다. 이때 적분리미터에 의해 적분제어량을 제한함으로써 예측지 못한 출력결과를 막을 수 있습니다.
적분리미터값은 적분제어에 의한 조작출력Y의 최대값100%를 기준으로 얼마나 허용할지를 결정합니다. 시스템에 따라 적분제어의 영향력은 달라질 수 있으나 낮을수록 안전합니다.

※ IL 값이 0 이면 적분리미터는 동작하지 않습니다.

▶ YMax : 최대 출력값
부호없는 16비트 정수값으로 단위는 0.1%입니다. (0.0~100.0%)

제어시스템은 내부적으로 -100.0% ~ +100.0%를 출력합니다. 이 출력값은 YMax값에 의해 -YMax ~ +YMax 로 조절됩니다.
히터제어의 경우 최대 출력값의 제한은 최대사용전력을 제한하는 효과를 가집니다.

※ YMax 값이 0 이면 출력이 차단(0)됩니다.

※ 단, 출력방식(PWM Output Mode) 이 1 일 때 YMax 가 100.0 이 아니면 0%의 출력에 도달할 수 없으므로 유의하셔야 합니다.

<3 방식>
<0 방식> 의 위상반전 출력방식입니다. 20%의 출력은 80%로 반전되어 출력되고 80%의 출력은 20%로 반전되어 출력됩니다.

<4 방식>
<1 방식의> 의 위상반전 출력방식입니다. 20%의 출력은 80%로 반전되어 출력되고 80%의 출력은 20%로 반전되어 출력됩니다.

<5 방식>
<2 방식의> 의 위상반전 출력방식입니다. 20%의 출력은 -20%로 반전되어 출력되고 80%의 출력은 -80%로 반전되어 출력됩니다.

36.1.1. WIMC_Off_PID

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Off_PID
(
              WIMC_INT32              ch
);
```

➲ Description

ch(0~3)로 지정된 PID제어기 기능을 OFF합니다.

WIMC엔진의 PID제어기는 기본적으로 ADC입력을 피드백 센서값으로 취하고, PWM출력포트와 GOut포트(범용출력포트)를 이용하는 PWM출력을 조작값으로 택하고 있습니다.
모션엔진이 리셋되었을 때 모든 PID제어기의 기능은 꺼지게 됩니다.

함수에 대한 사용은 아래의 순서를 따릅니다.
① PID제어기 리셋 : WIMC_Reset_PID(ch);
② PID제어기 셋업 : WIMC_Setup_PID(ch, ptPid);
③ PID제어기 동작 : WIMC_Run_PID(ch);
④ 필요시> PID제어기 목표값 재설정 : WIMC_Set_PIDSV(ch, val);
⑤ PID제어기 정지 : WIMC_Off_PID(ch);

관련 내용 및 설명은 (35.1절)을 참고 하실수 있습니다.

➲ Parameters

ch
채널(ch = 0,1,2,3)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

36.1.2. WIMC_Reset_PID

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Reset_PID
(
              WIMC_INT32              ch
);
```

➲ Description

ch(0~3)로 지정된 PID제어기를 초기화 합니다.

WIMC엔진의 PID제어기는 기본적으로 ADC입력을 피드백 센서값으로 취하고, PWM출력포트와 GOut포트(범용출력포트)를 이용하는 PWM출력을 조작값으로 택하고 있습니다.
모션엔진이 리셋되었을 때 모든 PID제어기의 기능은 꺼지게 됩니다.

함수에 대한 사용은 아래의 순서를 따릅니다.
① PID제어기 리셋 : WIMC_Reset_PID(ch);
② PID제어기 셋업 : WIMC_Setup_PID(ch, ptPid);
③ PID제어기 동작 : WIMC_Run_PID(ch);
④ 필요시> PID제어기 목표값 재설정 : WIMC_Set_PIDSV(ch, val);
⑤ PID제어기 정지 : WIMC_Off_PID(ch);

관련 내용 및 설명은 (35.1절)을 참고 하실수 있습니다.

➲ Parameters

ch
채널(ch = 0,1,2,3)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

36.1.3. WIMC_Run_PID

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Run_PID
(
              WIMC_INT32              ch
);
```

➲ Description

ch(0~3)로 지정된 PID제어기를 동작시킵니다.

WIMC엔진의 PID제어기는 기본적으로 ADC입력을 피드백 센서값으로 취하고, PWM출력포트와 GOut포트(범용출력포트)를 이용하는 PWM출력을 조작값으로 택하고 있습니다.
모션엔진이 리셋되었을 때 모든 PID제어기의 기능은 꺼지게 됩니다.

함수에 대한 사용은 아래의 순서를 따릅니다.
① PID제어기 리셋 : WIMC_Reset_PID(ch);
② PID제어기 셋업 : WIMC_Setup_PID(ch, ptPid);
③ PID제어기 동작 : WIMC_Run_PID(ch);
④ 필요시> PID제어기 목표값 재설정 : WIMC_Set_PIDSV(ch, val);
⑤ PID제어기 정지 : WIMC_Off_PID(ch);

관련 내용 및 설명은 (35.1절)을 참고 하실수 있습니다.

➲ Parameters

ch
채널(ch = 0,1,2,3)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

36.1.4. WIMC_Setup_PID

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Setup_PID
(
              WIMC_INT32              ch,
              WIMC_PID_t*             ptPid
);
```

➲ Description

ch(0~3)로 지정된 PID제어기를 포인터변수 ptPid가 지정하는 pid구조체로 셋업합니다.

-PID제어기 셋업(Setup)
PID제어기가 올바르게 작동하기 위해서는 PID제어기의 다양한 파라메터값들을 설정해야만 합니다.

이 함수로 전달되는 두가지 파라메터는 각각 PID채널 번호와 PID제어기 구조체입니다.

▶ PID 구조체 원형

```
typedef struct tag_WIMC_PID_t
{
        UINT16 SV;       // 제어목표값
        UINT16 K;        // 비례대
        UINT16 Ti;       // 적분시간
        UINT16 Td;       // 미분시간
        UINT16 Tc;       // 제어주기시간
        UINT16 Taw;      // 안티리완이드시간
        UINT16 Hys;      // 불감대역
        UINT16 AL;       // Low 알람
        UINT16 AH;       // High 알람
        UINT8 InputId;   // 피드백센서입력 번호
        UINT8 OutputId;  // 조작출력 번호
        UINT8 OutputMode;// 출력방식
        UINT8 PolarOutport;// 극성출력 GOut포트
        INT32 Extra;     // 추가 32비트 파라메터 예약
        UINT16 IL;       // 적분리미터
        UINT16 YMax;     // 최대/최소 조작출력값의 절대치

} WIMC_PID_t;
```

함수에 대한 사용은 아래의 순서를 따릅니다.

① PID제어기 리셋 : WIMC_Reset_PID(ch);

② PID제어기 셋업 : WIMC_Setup_PID(ch, ptPid);

③ PID제어기 동작 : WIMC_Run_PID(ch);

④ 필요시> PID제어기 목표값 재설정 : WIMC_Set_PIDSV(ch, val);

⑤ PID제어기 정지 : WIMC_Off_PID(ch);

관련 내용 및 설명은 (35.1절)을 참고 하실수 있습니다.

➲ Parameters

ch

채널 (ch= 0,1,2,3)

ptPid

PID 구조체(포인터변수 ptPid)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

36.1.5. WIMC_Set_PIDSV

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Set_PIDSV
(
              WIMC_INT32              ch,
              WIMC_INT32              val
);
```

➲ Description

ch(0~3)로 지정된 PID제어기 제어 목표량을 재설정합니다.

WIMC엔진의 PID제어기는 기본적으로 ADC입력을 피드백 센서값으로 취하고, PWM출력포트와 GOut포트(범용출력포트)를 이용하는 PWM출력을 조작값으로 택하고 있습니다.
모션엔진이 리셋되었을 때 모든 PID제어기의 기능은 꺼지게 됩니다.

함수에 대한 사용은 아래의 순서를 따릅니다.

① PID제어기 리셋 : WIMC_Reset_PID(ch);
② PID제어기 셋업 : WIMC_Setup_PID(ch, ptPid);
③ PID제어기 동작 : WIMC_Run_PID(ch);
④ 필요시> PID제어기 목표값 재설정 : WIMC_Set_PIDSV(ch, val);
⑤ PID제어기 정지 : WIMC_Off_PID(ch);

관련 내용 및 설명은 (35.1절)을 참고 하실수 있습니다.

➲ Parameters

ch
채널(ch = 0,1,2,3)

val
제어 목표량입니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

36.1.6. WIMC_Is_PID_On

➲ Syntax

```
WIMC_BOOL       WIMC_API                WIMC_Is_PID_On
(
                WIMC_INT32              ch
);
```

➲ Description

PID 제어기의 On 상태를 확인합니다.

WIMC엔진의 PID제어기는 기본적으로 ADC입력을 피드백 센서값으로 취하고, PWM출력포트와 GOut포트(범용출력포트)를 이용하는 PWM출력을 조작값으로 택하고 있습니다.
모션엔진이 리셋되었을 때 모든 PID제어기의 기능은 꺼지게 됩니다.

함수에 대한 사용은 아래의 순서를 따릅니다.
① PID제어기 리셋 : WIMC_Reset_PID(ch);
② PID제어기 셋업 : WIMC_Setup_PID(ch, ptPid);
③ PID제어기 동작 : WIMC_Run_PID(ch);
④ 필요시> PID제어기 목표값 재설정 : WIMC_Set_PIDSV(ch, val);
⑤ PID제어기 정지 : WIMC_Off_PID(ch);

관련 내용 및 설명은 (35.1절)을 참고 하실수 있습니다.

➲ Parameters

ch
채널(ch = 0,1,2,3)

➲ Return Value

PID 제어기의 On 상태를 반환합니다.

Return code/value	Description
WIMC_TRUE	PID 제어기 On 상태입니다.
WIMC_FALSE	PID 제어기 Off 상태입니다.

➲ Example

➲ See Also

36.1.7. WIMC_Get_PID_Setup

➲ Syntax

```
WIMC_BOOL       WIMC_API                WIMC_Get_PID_Setup
(
                WIMC_INT32              ch,
                WIMC_PID_t*             ptPid
);
```

➲ Description

ch(0~3)로 지정된 PID제어기의 설정들을 읽어들입니다.

▶ PID 구조체 원형

```
typedef struct tag_WIMC_PID_t
{
        UINT16 SV;       // 제어목표값
        UINT16 K;        // 비례대
        UINT16 Ti;       // 적분시간
        UINT16 Td;       // 미분시간
        UINT16 Tc;       // 제어주기시간
        UINT16 Taw;      // 안티리완이드시간
        UINT16 Hys;      // 불감대역
        UINT16 AL;       // Low 알람
        UINT16 AH;       // High 알람
        UINT8 InputId;   // 피드백센서입력 번호
        UINT8 OutputId;  // 조작출력 번호
        UINT8 OutputMode;// 출력방식
        UINT8 PolarOutport;// 극성출력 GOut포트
        INT32 Extra;     // 추가 32비트 파라메터 예약
        UINT16 IL;       // 적분리미터
        UINT16 YMax;     // 최대/최소 조작출력값의 절대치

} WIMC_PID_t;
```

함수에 대한 사용은 아래의 순서를 따릅니다.

① PID제어기 리셋 : WIMC_Reset_PID(ch);

② PID제어기 셋업 : WIMC_Setup_PID(ch, ptPid);

③ PID제어기 동작 : WIMC_Run_PID(ch);

④ 필요시> PID제어기 목표값 재설정 : WIMC_Set_PIDSV(ch, val);

⑤ PID제어기 정지 : WIMC_Off_PID(ch);

관련 내용 및 설명은 (35.1절)을 참고 하실수 있습니다.

➲ Parameters

ch

채널 (ch= 0,1,2,3)

ptPid

PID 구조체(포인터변수 ptPid)

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

37. ALARM

37.1. 알람동작

WIMC 엔진은 여러 가지 이유로 알람이 발생할 수 있습니다. 알람는 크게 두 가지 부류로 나누어집니다. 하나는 단순통지형 알람(Simple Notification Alarm) 이며 다른 하나는 동작중지형 알람(Motion Stopping Alarm)입니다.

◉ 단순통지형 알람이란 동작을 중지시키지 않고 단지 알람상태만을 알립니다.
SoftLimit 동작에서 단순통지방식(Notify)을 선택하면 현재동작의 중지없이 알람상태만을 알려줍니다.

◉ 동작중지형 알람은 현재 실행중인 동작명령을 강제 정지시키는 알람입니다. 동작중지형 알람은 명령을 계속 실행하기에 위험요소가 많음을 나타내는 알람입니다.

알람종류

총 16종류의 알람을 발생시킵니다.

(1) 프로세스 알람 (Process Alarm) : [동작중지형알람]으로 엔진의 내부연산 오류로 발생합니다. 내부 연산중에 나누기 0 및 수신큐 혹은 명령큐의 오버플로우 등에서 발생합니다.

(2) 명령 알람 (Command Alarm) : [동작중지형알람]으로 명령패킷의 명령바이트가 정의되지 않은 명령을 지시하였거나 축지정바이트의 값이 적절하지 못할 때 발생합니다. 예로써 원호보간에서 동작 가능한 2축을 지정하지 않았을 때 발생합니다.

(3) 인터페이스 알람 (Interface Alarm) : [동작중지형알람]으로 메인직렬포트에 대한 직렬패킷이 오류를 발생했거나 직렬통신의 장애로 인해 발생합니다.

(4) 서보 알람 (Servo Alarm) : [동작중지형알람]으로 각 축마다 존재하는 인터페이스 신호 Servo Alarm 이 활성화 되었을 때 발생합니다.

(5) EMGSTOP 알람(Emergency Stop Alarm) : [동작중지형알람]으로 인터페이스 신호 EMGSTOP이 활성화 되었을 때 발생합니다. 또한 강제정지명령에 의해서도 발생됩니다. 이때에는 Command Error 오류와 같이 발생합니다.

(6) 원점복귀 알람 (Home Return Alarm) : [동작중지형알람]으로 원점탐색복귀 동작거리가 지정된 이동거리를 초과했거나, 원점복귀 후 지정된 편차이내에 수렴하지 못했을 때 발생합니다.

(7) 리미트 알람 (Limit Alarm) : [동작중지형알람]으로 각 축마다 존재하는 인터페이스 신호 LIMIT+, LIMIT-가 활성화 되었을 때 발생합니다.

(8) 소프트리미트 알람 (Soft Limit Alarm) : [동작중지형알람]과 [단순통지형알람]을 선택할 수 있습니다. 각 축마다 정해진 소프트리미트 하한위치값과 상한위치값 범위를 벗어났을 때 발생합니다.

(9) 보조인터페이스 알람 (Aux Interface Alarm) : [동작중지형알람]으로 보조직렬포트에 대한 직렬패킷이 오류를 발생했거나 직렬통신의 장애로 인해 발생합니다.

(10) 위치 알람 (Position Alarm) : [동작중지형알람]으로 보간동작의 완료시점의 위치오류나 서보의 Inpos신호 오류에 의해 발생합니다.

(11) WDT 알람 (Watch Dog Timer Alarm) : [동작중지형알람]으로 내부처리의 연산부하가 과중해서 스캔타임의 내부적으로

지정된 값보다 클 때 발생합니다.

(12) 파라메터 알람 (Parameter Alarm) : [단순통지형알람]으로 축설정 및 칩설정 그리고 그 외 기능 설정값이 지정된 범위를 벗어났거나 예약된 영역을 지정할 때 발생합니다. (WIMC 엔진은 이 알람을 발생하지 않습니다.)

(13) 보간종점보상 알람(Endpoint Interpolation Compensation Alarm) : [동작중지형알람]으로 원호보간의 종료위치가 지정된 목표위치에서 2Pulse 이상 벗어났을 때 발생합니다.

(14) 링크 알람(Link Alarm) : [동작중지형알람]으로 모션엔진과 확장장치간의 연결상태가 불량할 때 발생합니다. (WIMC 엔진은 이 알람을 발생하지 않습니다.)

(15) 메모리 알람 (Memory Alarm) : [동작중지형알람]으로 EEPROM을 포함한 내부메모리의 읽기/쓰기동작의 오류에 의해 발생합니다. (WIMC 엔진은 이 알람을 발생하지 않습니다.)

(16) 프로그램알람 (Program Alarm) : [동작중지형알람]으로 내부의 제어프로그램 가능한 제품에 한해서 프로그램 코드 오류로 발생합니다. 또한 명령의 실행결과가 올바르지 않았을 때에도 발생합니다.

37.1.1. WIMC_Is_AlarmAxis

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_AlarmAxis
(
              WIMC_UINT8              axis
);
```

➲ Description

특정축의 알람상태를 확인합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	해당축의 알람이 발생했습니다.
WIMC_FALSE	

➲ Example

➲ See Also

37.1.2. WIMC_Is_PositionAlarm

➲ Syntax

```
WIMC_BOOL     WIMC_API                WIMC_Is_PostionAlarm
(
              WIMC_UINT8              axis
);
```

➲ Description

위치 알람상태를 확인합니다.
보간동작의 완료시점의 위치오류나 서보의 Inpos신호 오류에 의해 발생합니다.

➲ Parameters

axis
적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	해당축의 위치 알람이 발생했습니다.
WIMC_FALSE	

➲ Example

➲ See Also

37.1.3. WIMC_Get_Alarm

➲ Syntax

```
WIMC_UINT32    WIMC_API                WIMC_Get_Alarm( void);
```

➲ Description

알람발생 종류를 읽습니다. 해당비트의 값이 1이면 해당 알람이 발생한 것입니다.

<표> WIMC_Get_Alarm()의 알람상태정보

비트번호	알람종류	대표정의
(LSB) 0	PROCESSOR	WIMC_ALARM_PROCESSOR
1	COMMAND	WIMC_ALARM_COMMAND
2	MAIN INTERFACE	WIMC_ALARM_MAIN_INTERFACE
3	SERVO	WIMC_ALARM_SERVO
4	EMGSTOP	WIMC_ALARM_EMGSTOP
5	HOME RETURN	WIMC_ALARM_HOMERETURN
6	LIMIT	WIMC_ALARM_LIMIT
7	SOFT LIMIT	WIMC_ALARM_SOFTLIMIT
8	AUX INTERFACE	WIMC_ALARM_AUX_INTERFACE
9	POSITION	WIMC_ALARM_POSITION
10	WDT	WIMC_ALARM_WDT
11	PARAMETER	WIMC_ALARM_PARAMETER
12	IENDCOMP	WIMC_ALARM_IENDCOMP
13	LINK	WIMC_ALARM_LINK
14	MEMORY	WIMC_ALARM_MEMORY
15	PROGRAM	WIMC_ALARM_PROGRAM
16~31	-	정의 안됨

➲ Parameters

➲ Return Value

아래의 하위 16비트값으로 알람종류를 리턴합니다.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SoftLimit	Limit	Home Return	EmgStop	Servo	Interface	Command	Process

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Program	Memory	Link	IEndComp	Parameter	WDT	Position	Aux Inteface

<관련된 정의>

Return code/value	Description
WIMC_ALARM_PROCESS	내부 큐(Queue) 시스템이 오버플로잉되거나 내부 연산오류로 인해 모션엔전 코어에서 발생된 알람입니다. 대부분 통신을 통한 리셋명령으로 응답되지 않고 하드웨어 리셋을 통해서 회복될 수 있습니다.
WIMC_ALARM_COMMAND	잘못된 명령형태이거나, 비상정지와 같은 명령에 의한 알람이 발생되었을 때 비상실행상태로 전환됩니다.
WIMC_ALARM_INTERFACE	메인 통신포트의 통신 속도가 맞지 않거나, 노이즈 등으로 통신오류가 발생할 때 비상실행상태로 전환됩니다.
WIMC_ALARM_SERVO	서보장치의 알람신호가 발생될 때 비상실행상태로 전환됩니다.
WIMC_ALARM_EMGSTOP	이 기능은 서보구성설정에 의해 활성화/비활성화가 가능합니다.
WIMC_ALARM_HOMERETURN	모션엔진의 ESTOP핀 신호가 활성화 되었을 때 발생합니다.
WIMC_ALARM_LIMIT	원점복귀 동작에서 알람이 발생되면 비상실행상태로 전환됩니다.
WIMC_ALARM_SOFTLIMIT	모션엔진의 LIMIT신호가 검출되었을 때 비상실행상태로 전환됩니다. 이 기능은 구성설정에 의해 활성화/비활성화가 가능합니다.
WIMC_ALARM_AUXINTERFACE	현재위치가 SOFTLIMIT로 설정값에 도달했을 때 비상실행상태로 전환됩니다. 이 기능은 구성설정에 의해 비상실행상태로 전환하지 않는 단순통지형 동작방식을 변경할 수 있습니다.
WIMC_ALARM_POSITION	보조 통신포트의 통신 속도가 맞지 않거나, 노이즈 등으로 통신오류가 발생할 때 비상실행상태로 전환됩니다.
WIMC_ALARM_WDT	서보의 INPOSITION 신호동작오류나 위치제어오류가 발생했을 때 비상실행상태로 전환됩니다.
WIMC_ALARM_PARAMETER	서보의 INPOSITION 신호동작은 서보구성설정에 의해 활성화/비활성화가 가능합니다.
WIMC_ALARM_IENDCOMP	모션엔진의 스캔타임의 평균값이 내부의 지정된 시간값보다 크게 되면 비상실행상태로 전환됩니다.
WIMC_ALARM_LINK	이 기능은 구성설정에 의해 활성화/비활성화가 가능합니다.
WIMC_ALARM_MEMORY	명령의 파라메터가 형식이 잘못되었을 때 비상실행상태로 전환됩니다.
WIMC_ALARM_PROGRAM	확장연결장치에서 알람이 발생될 때 비상실행상태로 전환됩니다.

➲ Example

➲ See Also

37.1.4. WIMC_Clr_AlarmOption

➲ Syntax

```
void        WIMC_API                WIMC_Clr_AlarmOption
(
            WIMC_UINT32             nFlags
);
```

➲ Description

알람발생 방식을 조절(해제)하는 옵션입니다.

아래값으로 해제합니다.

WIMC_ALARMOPTION_EMGSTOPIMSTOP : 외부ESTOP신호에 의한 즉시정지
WIMC_ALARMOPTION_WDT : WDT 알람 설정
WIMC_ALARMOPTION_IENDCOMP : 보간종점보상 알람발생 기능 설정
WIMC_ALARMOPTION_RECOVERREG : 레지스터복구 기능설정

<리셋시 설정>
보간종점알람 설정
WDT알람 해제
레지스터복구기능 해제
EStop신호에 의한 급정지 해제(급감속정지방식)
비상정지 급감속계수 = 0

➲ Parameters

nFlags

아래 값들을 사용합니다.

WIMC_ALARMOPTION_EMGSTOPIMSTOP : 외부 ESTOP 신호에 의한 감속정지
WIMC_ALARMOPTION_WDT : WDT 알람 해제
WIMC_ALARMOPTION_IENDCOMP : 보간종점보상 알람발생 기능 해제
WIMC_ALARMOPTION_RECOVERREG : 레지스터복구 기능해제

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

37.1.5. WIMC_Set_AlarmOption

➲ Syntax

```
void        WIMC_API                WIMC_Set_AlarmOption
(
            WIMC_UINT32             nFlags
);
```

➲ Description

알람발생 방식을 조절(설정)하는 옵션입니다.

아래 값으로 설정합니다.

WIMC_ALARMOPTION_EMGSTOPIMSTOP : 외부ESTOP신호에 의한 즉시정지
WIMC_ALARMOPTION_WDT : WDT 알람 설정
WIMC_ALARMOPTION_IENDCOMP : 보간종점보상 알람발생 기능 설정
WIMC_ALARMOPTION_RECOVERREG : 레지스터복구 기능설정

<리셋시 설정>
보간종점알람 설정
WDT알람 해제
레지스터복구기능 해제
EStop신호에 의한 급정지 해제(급감속정지방식)
비상정지 급감속계수 = 0

➲ Parameters

nFlags

아래 값들을 사용합니다.

WIMC_ALARMOPTION_EMGSTOPIMSTOP : 외부ESTOP신호에 의한 즉시정지
WIMC_ALARMOPTION_WDT : WDT 알람 설정
WIMC_ALARMOPTION_IENDCOMP : 보간종점보상 알람발생 기능 설정
WIMC_ALARMOPTION_RECOVERREG : 레지스터복구 기능설정

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

37.1.6. WIMC_Get_AlarmOption

➲ Syntax

```
WIMC_UINT32     WIMC_API                WIMC_Get_AlarmOption( void);
```

➲ Description

설정된 알람설정옵션을 읽습니다.

➲ Parameters

➲ Return Value

아래 알람옵션설정 상태를 반환합니다.

Return code/value	Description
WIMC_ALARMOPTION_EMGSTOPIMSTOP	외부ESTOP신호에 의한 즉시정지
WIMC_ALARMOPTION_WDT	WDT 알람 설정
WIMC_ALARMOPTION_IENDCOMP	보간종점보상 알람발생 기능 설정
WIMC_ALARMOPTION_RECOVERREG	레지스터복구 기능설정

➲ Example

➲ See Also

37.1.7. WIMC_Enable_StopRotateByAlarm

➲ Syntax

```
void        WIMC_API                WIMC_Enable_StopRotateByAlarm
(
            WIMC_UINT8              axis,
            WIMC_BOOL               bEnable
);
```

➲ Description

알람에 의한 스핀들 자동 정지 기능을 Enable/Disable 합니다.

Stop rotate by alarm (알람발생시 회전정지기능)

WIMC 엔진은 스핀들 전용명령에 의해 스핀들축이 회전하고 있을 때 알람이 발생해도 스핀들의 회전은 정지하지 않게 됩니다. 만약 알람에 의한 비상실행상태에서는 스핀들이 자동으로 정지하고 싶다면 이 옵션을 활성화 해야 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

bEnable

WIMC_TRUE : Enable 합니다.

WIMC_FALSE : Disable 합니다.

➲ Return Value

Return code/value	Description

➲ Example

➲ See Also

37.1.8. WIMC_Is_StopRotateByAlarm

➲ Syntax

```
WIMC_BOOL     WIMC_API                  WIMC_Is_StopRotateByAlarm
(
              WIMC_UINT8                axis
);
```

➲ Description

알람에 의한 비상실행상태에서 스핀들이 자동으로 정지되었는지 확인합니다.

Stop rotate by alarm (알람발생시 회전정지기능)

스핀들 전용명령에 의해 스핀들축이 회전하고 있을 때 알람이 발생해도 스핀들의 회전은 정지하지 않게 됩니다. 만약 알람에 의한 비상실행상태에서는 스핀들이 자동으로 정지하고 싶다면 이 옵션을 활성화 해야 합니다.

➲ Parameters

axis

적용되는 축을 지정합니다.(WIMC_X/ WIMC_Y/ WIMC_Z/WIMC_U/ WIMC_V/ WIMC_W/ WIMC_S1/ WIMC_S2)

➲ Return Value

Return code/value	Description
WIMC_TRUE	알람에 의해 정지되었습니다.
WIMC_FALSE	정지하지 않았습니다.

➲ Example

➲ See Also

부록 I. WIMC 라이브러리 찾아보기

❶ 알파벳 순서입니다.

WIMC_AlarmReset	2.1.	WIMC_Enable_InterruptStatePacket	8.3.	WIMC_Get_CmdQueueLeftCmd	7.1.
WIMC_ArcCCW	27.2.	WIMC_Enable_LowerLimit	22.1.	WIMC_Get_CmdState	19.1.
WIMC_ArcCW	27.2.	WIMC_Enable_MI	28.1.	WIMC_Get_CompleteStatePacketCount	8.2.
WIMC_BitPattern	27.3.	WIMC_Enable_Notify_Complete	18.1.	WIMC_Get_CurCmdPulse	11.1.
WIMC_Cancel	2.1.	WIMC_Enable_RestrictedImCmd	9.1.	WIMC_Get_CurDelayTime	15.1.
WIMC_Change_Comm_Baudrate	1.1.	WIMC_Enable_ServoAlarmAuto	31.7.	WIMC_Get_CurPos	26.1.
WIMC_Close	1.1.	WIMC_Enable_ServoEStopAuto	31.3.	WIMC_Get_CurVel	24.2.
WIMC_Clr_AlarmOption	36.1.	WIMC_Enable_ServoInposAlarm	31.6.	WIMC_Get_CurVelAxis	24.2.
WIMC_Clr_Queue	7.1.	WIMC_Enable_ServoInposAuto	31.6.	WIMC_Get_Dec	24.6.
WIMC_Clr_StateTimer	10.1.	WIMC_Enable_ServoOnAuto	31.1.	WIMC_Get_Dec2	24.8.
WIMC_ClrBit_GOut	32.1.	WIMC_Enable_ServoReadyAuto	31.5.	WIMC_Get_Dec2Axis	24.8.
WIMC_CMove	29.2.	WIMC_Enable_ServoResetAuto	31.2.	WIMC_Get_DecAxis	24.6.
WIMC_DecStop	2.1.	WIMC_Enable_ServoUserOutAuto	31.4.	WIMC_Get_DllVer	1.1.
WIMC_Delay	15.1.	WIMC_Enable_StepExe	14.1.	WIMC_Get_DriveVel	24.3.
WIMC_EmgStop_ExtIO	33.1.	WIMC_Enable_StopRotateByAlarm	36.1.	WIMC_Get_DriveVelAxis	24.3.
WIMC_Enable_AuxPortAllCommand	4.3.	WIMC_Enable_UpperLimit	22.1.	WIMC_Get_DwellTime	25.4.
WIMC_Enable_AuxPortStatePacket	4.3.	WIMC_Enable_VelForceConvergence	25.6.	WIMC_Get_DwellTimeAxis	25.4.
WIMC_Enable_AuxPortStateTimer	4.3.	WIMC_Enable_VelPosAutoTransition	25.8.	WIMC_Get_EncPos	26.2.
WIMC_Enable_Backlash	26.4.	WIMC_Enter_ImCmd	7.1.	WIMC_Get_EncZCount	26.2.
WIMC_Enable_BusyStatePacket	8.1.	WIMC_Enter_QueueCmd	7.1.	WIMC_Get_EngineQueueLeftCmd	7.1.
WIMC_Enable_Cexe	13.1.	WIMC_Estop	2.1.	WIMC_Get_EventStatePacketCount	8.4.
WIMC_Enable_CLV	25.2.	WIMC_Get_Acc	24.5.	WIMC_Get_ExeCmdID	6.1.
WIMC_Enable_Comparator	34.1.	WIMC_Get_Acc2	24.7.	WIMC_Get_ExtEmgStopMode	5.1.
WIMC_Enable_CompleteStatePacket	8.2.	WIMC_Get_Acc2Axis	24.7.	WIMC_Get_ExtIO	33.1.
WIMC_Enable_DwellTimeAxis	25.4.	WIMC_Get_AccAxis	24.5.	WIMC_Get_ExtIO_Link_Status	33.1.
WIMC_Enable_ErrBeep	1.1.	WIMC_Get_ADC	32.1.	WIMC_Get_ExtIO_MPG_Count	33.1.
WIMC_Enable_EventStatePacket	8.4.	WIMC_Get_Alarm	36.1.	WIMC_Get_ExtIO_MPG_Status	33.1.
WIMC_Enable_Exe	2.1.	WIMC_Get_AlarmOption	36.1.	WIMC_Get_GearRatio	26.3.
WIMC_Enable_ExtIO	33.1.	WIMC_Get_AuxBaudrate	4.2.	WIMC_Get_GIn	32.1.
WIMC_Enable_ExtServoIO	33.1.	WIMC_Get_Backlash	26.4.	WIMC_Get_GIOEncoder	32.1.
WIMC_Enable_FFC	25.7.	WIMC_Get_BoardID	1.1.	WIMC_Get_GIOUpCounter	32.1.
WIMC_Enable_GIOEncoder	32.1.	WIMC_Get_BPU	27.3.	WIMC_Get_GOut	32.1.
WIMC_Enable_HomeDevAlarm	30.1.	WIMC_Get_BusySize	9.1.	WIMC_Get_HomeDec	30.1.
WIMC_Enable_HomeReturnAutoZero	30.1.	WIMC_Get_BusyStatePacketCount	8.1.	WIMC_Get_HomeMode	30.1.
WIMC_Enable_HomeSensorAutoZero	30.1.	WIMC_Get_CLVAxis	25.2.	WIMC_Get_HomeSeekOffsetPulse	30.1.
WIMC_Enable_HomeServoReset	30.1.	WIMC_Get_CmdID	6.1.	WIMC_Get_HomeSeekVel	30.1.
WIMC_Enable_IEndComp	9.1.	WIMC_Get_CmdPulseMax	11.1.	WIMC_Get_ImStopDec	5.1.

WIMC_Get_InportMode	9.1	WIMC_IMove	29.1.	WIMC_Is_ServoEStopAuto	31.3.
WIMC_Get_InterruptMode	9.1.	WIMC_ImStop	2.1.	WIMC_Is_ServoEStopLevel	31.3.
WIMC_Get_InterruptStatePacketCount	8.3.	WIMC_Is_Alarm	3.1.	WIMC_Is_ServoInpos	31.6.
WIMC_Get_LastErr	1.1.	WIMC_Is_Busy	20.1.	WIMC_Is_ServoInposAlarm	31.6.
WIMC_Get_LowerLimitLevel	22.1.	WIMC_Is_BusyStatePacket	8.1.	WIMC_Is_ServoInposAuto	31.6.
WIMC_Get_MainBaudrate	4.1.	WIMC_Is_CExe	13.1.	WIMC_Is_ServoInposLevel	31.6.
WIMC_Get_OutportMode	9.1	WIMC_Is_CLV	25.2.	WIMC_Is_ServoOn	31.1.
WIMC_Get_PID_Setup	35.1.	WIMC_Is_Complete	20.1.	WIMC_Is_ServoOnAuto	31.1.
WIMC_Get_PosDispMode	9.1.	WIMC_Is_CompleteAxis	20.1.	WIMC_Is_ServoOnLevel	31.1.
WIMC_Get_PosMode	9.1.	WIMC_Is_CompleteStatePacket	8.2.	WIMC_Is_ServoReady	31.5.
WIMC_Get_ProcCmdNo	9.1.	WIMC_Is_EmgExe	3.1.	WIMC_Is_ServoReadyAuto	31.5.
WIMC_Get_PulseLevel	21.1.	WIMC_Is_EnableExe	3.1.	WIMC_Is_ServoReadyLevel	31.5.
WIMC_Get_PulseMode	21.1.	WIMC_Is_Err	1.1.	WIMC_Is_ServoReset	31.2.
WIMC_Get_PWMDutyCount	32.1.	WIMC_Is_Estop	3.1.	WIMC_Is_ServoResetAuto	31.2.
WIMC_Get_PWMPeriodCount	32.1.	WIMC_Is_EventStatePacket	8.4.	WIMC_Is_ServoResetLevel	31.2.
WIMC_Get_ReadyCmdID	6.1.	WIMC_Is_ExtIO_Link0	33.1.	WIMC_Is_ServoUserIn	31.8.
WIMC_Get_S1VelOverride	25.3.	WIMC_Is_ExtIO_Link1	33.1.	WIMC_Is_ServoUserInLevel	31.8.
WIMC_Get_S2VelOverride	25.3.	WIMC_Is_ExtIO_Link2	33.1.	WIMC_Is_ServoUserOut	31.4.
WIMC_Get_ScanTime	12.1.	WIMC_Is_ExtIO_Link3	33.1.	WIMC_Is_ServoUserOutAuto	31.4.
WIMC_Get_ServoInposTime	31.6.	WIMC_Is_FFC	25.7.	WIMC_Is_ServoUserOutLevel	31.4.
WIMC_Get_ServoOnMode	31.1.	WIMC_Is_HomeDevAlarm	30.1.	WIMC_Is_StateTimer	10.1.
WIMC_Get_ServoPulseTime	31.2.	WIMC_Is_HomeDir	30.1.	WIMC_Is_StepExe	14.1.
WIMC_Get_SoftLimitMode	23.1.	WIMC_Is_HomeReturnAutoZero	30.1.	WIMC_Is_StopRotateByAlarm	36.1.
WIMC_Get_SoftLowerLimit	23.1.	WIMC_Is_HomeSensorAutoZero	30.1.	WIMC_Is_Sync	16.1.
WIMC_Get_SoftQueueLeftCmd	7.1.	WIMC_Is_HomeSensorLevel	30.1.	WIMC_Is_VelForceConvergence	25.6.
WIMC_Get_SoftQueueSize	7.1.	WIMC_Is_HomeServoReset	30.1.	WIMC_Is_VelPosAutoTransition	25.8.
WIMC_Get_SoftUpperLimit	23.1.	WIMC_Is_IEndComp	9.1.	WIMC_Leave_ImCmd	7.1.
WIMC_Get_StartVel	24.4.	WIMC_Is_InterruptStatePacket	8.3.	WIMC_Leave_QueueCmd	7.1.
WIMC_Get_StartVelAxis	24.4.	WIMC_Is_Link	1.1.	WIMC_Line	27.1.
WIMC_Get_StateTimer	10.1.	WIMC_Is_Lock_VelOverride	25.3.	WIMC_Load_ConfigFile	1.1.
WIMC_Get_UpperLimitLevel	22.1.	WIMC_Is_MI	28.1.	WIMC_Lock_VelOverride	25.3.
WIMC_Get_VelCtrlHz	25.5.	WIMC_Is_OnMotion	20.1.	WIMC_MIExe	28.1.
WIMC_Get_VelMode	24.1.	WIMC_Is_Pause	3.1.	WIMC_Notify_Complete	18.1.
WIMC_Get_VelModeAxis	24.1.	WIMC_Is_PID_On	35.1.	WIMC_Off_PID	35.1.
WIMC_Get_VelMultiplier	25.1.	WIMC_Is_PreparedCmd	19.1.	WIMC_Open_Comm	1.1.
WIMC_Get_VelOverride	25.3.	WIMC_Is_RestrictedImCmd	9.1.	WIMC_Open_Tcp	1.1.
WIMC_Get_VelState	20.1.	WIMC_Is_ServoAlarm	31.7.	WIMC_Pause	2.1.
WIMC_Get_VelStateAxis	20.1.	WIMC_Is_ServoAlarmAuto	31.7.	WIMC_Pulse_GOut	32.1.
WIMC_Home	27.4.	WIMC_Is_ServoAlarmLevel	31.7.	WIMC_Reallocate_SoftQueue	7.1.
WIMC_IHome	29.5.	WIMC_Is_ServoEStop	31.3.	WIMC_Register_Comparator_Message	34.1.

WIMC_Register_Complete_Message	18.1.
WIMC_Reset	2.1.
WIMC_Reset_ExtIO	33.1.
WIMC_Reset_PID	35.1.
WIMC_Restart_ExtIO	33.1.
WIMC_Resume	2.1.
WIMC_Run_PID	35.1.
WIMC_Save_ConfigFile	1.1.
WIMC_Set_Acc	24.5.
WIMC_Set_Acc2	24.7.
WIMC_Set_Acc2Axis	24.7.
WIMC_Set_AccAxis	24.5.
WIMC_Set_AlarmOption	36.1.
WIMC_Set_AuxBaudrate	4.2.
WIMC_Set_Backlash	26.4.
WIMC_Set_BoardID	1.1.
WIMC_Set_BPU	27.3.
WIMC_Set_BusySize	9.1.
WIMC_Set_BusyStatePacketCount	8.1.
WIMC_Set_CLVAxis	25.2.
WIMC_Set_CmdID	6.1.
WIMC_Set_CompleteStatePacketCount	8.2.
WIMC_Set_CurPos	26.1.
WIMC_Set_Dec	24.6.
WIMC_Set_Dec2	24.8.
WIMC_Set_Dec2Axis	24.8.
WIMC_Set_DecAxis	24.6.
WIMC_Set_DriveVel	24.3.
WIMC_Set_DriveVelAxis	24.3.
WIMC_Set_DwellTime	25.4.
WIMC_Set_DwellTimeAxis	25.4.
WIMC_Set_EncPos	26.2.
WIMC_Set_EventStatePacketCount	8.4.
WIMC_Set_ExtEmgStopMode	5.1.
WIMC_Set_ExtIO	33.1.
WIMC_Set_FFC_PMax	25.7.
WIMC_Set_GearRatio	26.3.
WIMC_Set_GIOEncoder	32.1.
WIMC_Set_GIOUpCounter	32.1.
WIMC_Set_Gout	32.1.
WIMC_Set_HomeDev	30.1.
WIMC_Set_HomeDir	30.1.
WIMC_Set_HomeMode	30.1.
WIMC_Set_HomeSeekOffsetPulse	30.1.
WIMC_Set_HomeSeekVel	30.1.
WIMC_Set_HomeSensorLevel	30.1.
WIMC_Set_ImStopDec	5.1.
WIMC_Set_InportMode	9.1
WIMC_Set_InterruptMode	9.1.
WIMC_Set_InterruptStatePacketCount	8.3.
WIMC_Set_LowerLimitLevel	22.1.
WIMC_Set_MainBaudrate	4.1.
WIMC_Set_OutportMode	9.1
WIMC_Set_PIDSV	35.1.
WIMC_Set_PosComparator	34.1.
WIMC_Set_PosComparatorX	34.1.
WIMC_Set_PosDispMode	9.1.
WIMC_Set_PosMode	9.1.
WIMC_Set_ProcCmdNo	9.1.
WIMC_Set_PulseLevel	21.1.
WIMC_Set_PulseMode	21.1.
WIMC_Set_PWM	32.1.
WIMC_Set_S1VelOverride	25.3.
WIMC_Set_S2VelOverride	25.3.
WIMC_Set_ServoAlarmLevel	31.7.
WIMC_Set_ServoEStop	31.3.
WIMC_Set_ServoEStopLevel	31.3.
WIMC_Set_ServoInposLevel	31.6.
WIMC_Set_ServoInposTime	31.6.
WIMC_Set_ServoOn	31.1.
WIMC_Set_ServoOnLevel	31.1.
WIMC_Set_ServoOnMode	31.1.
WIMC_Set_ServoPulseTime	31.2.
WIMC_Set_ServoReadyLevel	31.5.
WIMC_Set_ServoReset	31.2.
WIMC_Set_ServoResetLevel	31.2.
WIMC_Set_ServoUserInLevel	31.8.
WIMC_Set_ServoUserOut	31.4.
WIMC_Set_ServoUserOutLevel	31.4.
WIMC_Set_SoftLimitMode	23.1.
WIMC_Set_SoftLowerLimit	23.1.
WIMC_Set_SoftUpperLimit	23.1.
WIMC_Set_StartVel	24.4.
WIMC_Set_StartVelAxis	24.4.
WIMC_Set_StateTimer	10.1.
WIMC_Set_UpperLimitLevel	22.1.
WIMC_Set_VelCtrlHz	25.5.
WIMC_Set_VelMode	24.1.
WIMC_Set_VelModeAxis	24.1.
WIMC_Set_VelMultiplier	25.1.
WIMC_Set_VelOverride	25.3.
WIMC_SetBit_GOut	32.1.
WIMC_Setup_PID	35.1.
WIMC_Spindle	29.4.
WIMC_Sync	16.1.
WIMC_Unlock_VelOverride	25.3.
WIMC_Unregister_Comparator_Message	34.1.
WIMC_Unregister_Complete_Message	18.1.
WIMC_Update_Baudrate	4.4.
WIMC_Wait	17.1.
WIMC_Wait_GIO	32.1.
WIMC_WaitMotion	18.1.
WIMC_XDelay	15.1.
WIMC_XMove	29.3.