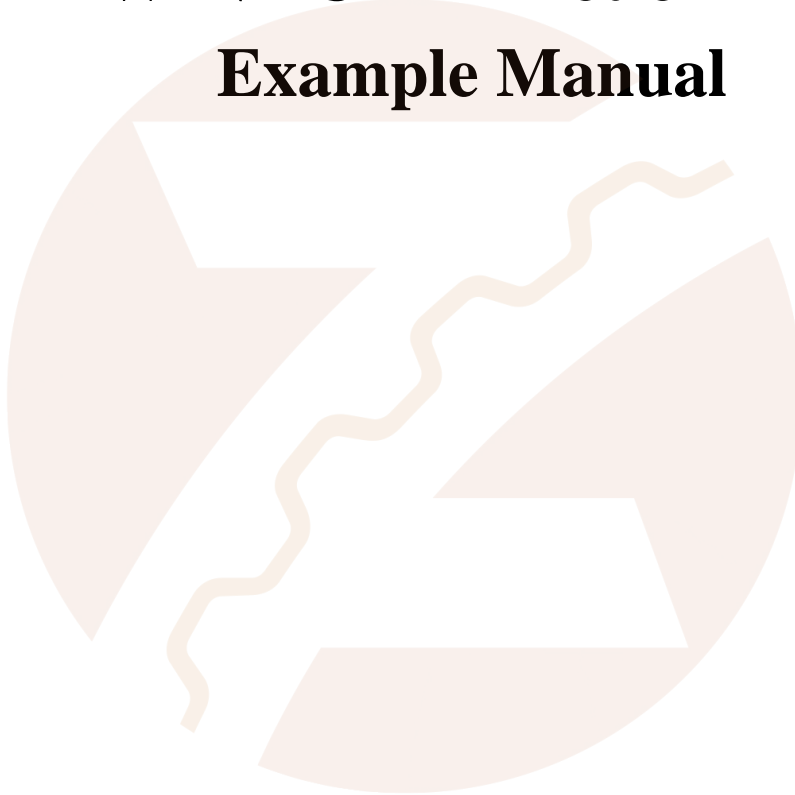


# **HIWIN PCI-4P Motion Library Example Manual**



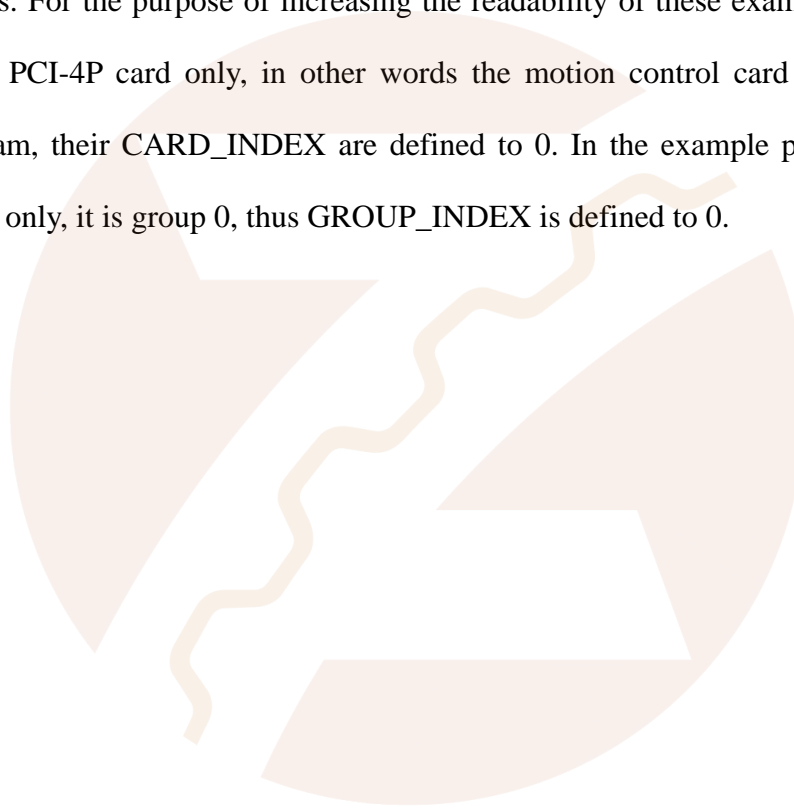
**ZETEK®**  
**Mar. 3. 2005**

## Contents

1. Introduction.....	3
2. Group, mechanism parameter setting .....	4
3. Initialize and close motion library .....	5
4. Set system status .....	7
5. Set feeding speed .....	9
6. Line, arc, circular motion.....	10
7. Point to point motion .....	13
8. JOG motion.....	15
9. Getting speed, position and command position .....	16
10. Motion hold, continue, abort.....	19
11. Motion status.....	21
12. Speed override .....	22
13. Software limit check and limit switch check .....	24
14. Speed blend function.....	26
15. Get and clear error status .....	27
16. Homing .....	28
17. Acceleration, deceleration step setting.....	30

## 1. Introduction

The examples provided are for console mode, user may integrate these examples into his own application program. MCCL can support up to 12 PCI-4P cards and 72 groups. For the purpose of increasing the readability of these examples, it is assumed to use 1 PCI-4P card only, in other words the motion control card used in the example program, their `CARD_INDEX` are defined to 0. In the example program we defined 1 group only, it is group 0, thus `GROUP_INDEX` is defined to 0.



## 2. Group, mechanism parameter setting

### Related functions

MCC\_SetSysMaxSpeed()  
MCC\_GetSysMaxSpeed()  
MCC\_SetGroupConfig()  
MCC\_SetMachParam()  
MCC\_GetMachParam()  
MCC\_UpdateMachParam()

### Example program

InitSys.cpp

### Explanation

This example explains the setup process of group parameter and mechanism parameter. Use MCC\_SetSysMaxSpeed() to set the upper limit of feed speed first ,then use MCC\_SetMachParam() to set mechanism parameter of each axis. Finally use MCC\_SetGroupConfig() to set group parameter.

The more detailed explanation of related group parameter and mechanism parameter, please refer to “**PCI-4P motion library user’s manual**”.



### 3. Initialize and close motion library

**Related functions**

MCC\_InitSystem()  
 MCC\_CloseSystem()  
 MCC\_GetMotionStatus()

**Example program**

InitSys.cpp

**Explanation**

After finish setting group parameter and mechanism parameter, have to use MCC\_InitSystem() to start the motion library. The parameters needed, please refer to “PCI-4P motion library user’s manual”. Used example is as follows:

```
SYS_CARD_CONFIG stCardConfig[MAX_CARD_NUM];
..
stCardConfig[CARD_INDEX].wCardType = wCardType;
```

```

// Start motion library

nRet = MCC_InitSystem (INTERPOLATION_TIME, stCardConfig,1);
// Interpolation time set to 10 ms; hardware parameter, use 1 PCI4P card only.
if (nRet == NO_ERR)// Initialization succeed.
{
/*

User can do other initialization here, for example set feeding speed, set
displacement unit.

*/
}

```

MCC\_CloseSystem() is used to close MCCL and driver function library. There are two ways to close the system:

#### **A. Finish executing all motion command then close system.**

Check if system is in stop condition, if the returned value from MCC\_GetMotionStatus() is 1, then is in stop condition.

```
while ((nRet = MCC_GetMotionStatus(GROUP_INDEX)) != 1)
{
    MCC_TimeDelay(1); // Sleep 1 ms
                    // Using "while" command, it is necessary to avoid
                    // system's being lockup.
                    // Call MCC_TimeDelay() to release CPU.
}

MCC_CloseSystem(); // Close MCCL and driver function library.
```

#### **B. Close motion library directly**

Call MCC\_CloseSystem() only, system will stop operation right away.

## 4. Set system status

### Related functions

MCC\_SetUnit()  
MCC\_GetUnit()  
MCC\_SetAbsolute()  
MCC\_SetIncrease()  
MCC\_GetCoordType()  
MCC\_SetAccType()  
MCC\_GetAccType()  
MCC\_SetDecType()  
MCC\_GetDecType()  
MCC\_SetPtPAccType()  
MCC\_GetPtPAccType()  
MCC\_SetPtpDecType()  
MCC\_GetPtpDecType()  
MCC\_SetServoOn()  
MCC\_SetServoOff()  
MCC\_EnablePosReady()  
MCC\_DisablePosReady()

### Example program

SetStatus.cpp

### Explanation

If system's status is not set specially, system will operate according to its preset status, please refer to "PCI-4P motion library user's manual". Used example is shown as follows:

MCC\_SetUnit(\_MM\_, GROUP\_INDEX);// Use mm as the unit of displacement.

MCC\_SetAbsolute(GROUP\_INDEX);// Use absolute coordinate type to show the position of each axis.

// Use 'T' curve as the acceleration type of line, arc, circular motion.

```
MCC_SetAccType('T', GROUP_INDEX);
```

// Use 'S' curve as the deceleration type of line, arc, circular motion.

```
MCC_SetDecType('S', GROUP_INDEX);
```

// Use 'T' curve as the acceleration type of point to point motion.

```
MCC_SetPtPAccType('T', 'T', 'T', 'T', 0, 0, GROUP_INDEX);
```

// Use 'S' curve as the deceleration type of point to point motion.

```
MCC_SetPtpDecType('S', 'S', 'S', 'S', 0, 0, GROUP_INDEX);
```

```
MCC_SetServoOn(0, CARD_INDEX); // Set axis 0 servo on
```

// Set on position ready output

```
MCC_EnablePosReady(CARD_INDEX);
```

The maximum value of feeding speed can not be greater than  $\text{RPM} \times \text{pitch} / \text{gear\_ratio}$ , otherwise MCCL will use  $\text{RPM} \times \text{pitch} / \text{gear\_ratio}$  as the maximum value for each axis's feeding speed automatically in its trajectory planning. RPM, pitch, gear\_ratio are each axis's mechanism parameters.



## 5. Set feeding speed

### Related functions

MCC\_SetFeedSpeed()  
MCC\_GetFeedSpeed()  
MCC\_SetPtPSpeed()  
MCC\_GetPtpSpeed()

### Example program

SetSpeed.cpp

### Explanation

It is necessary set feeding speed before doing any motion. Special notice must be taken that the feeding speed can not be greater than the value set by MCC\_SetSysMaxSpeed().

Use MCC\_SetFeedSpeed() to set the feeding speed of line, arc, circular motion. For example when call MCC\_SetFeedSpeed (20, GROUP\_INDEX), means feeding speed is 20 mm/sec or 20 inch /sec, depends on the unit used.

For point to point motion instead of calling MCC\_SetFeedSpeed(), use MCC\_SetPtPSpeed() to set speed, but it's values are set as percent of maximum speed for each axis., range is 0 ~ 100. For example when execute MCC\_SetPtPSpeed (50, GROUP\_INDEX), the speed for each axis of point to point motion is  $(wRPM \times dfPitch / dfGearRatio) \times 50\%$ .

## 6. Line, arc, circular motion

### Related functions

MCC\_SetAbsolute()  
 MCC\_SetFeedSpeed()  
 MCC\_Line()  
 MCC\_ArcXY()  
 MCC\_CircleXY()

### Example program

GeneralMotion.cpp

### Explanation

After finish setting mechanism parameter, starting system, setting maximum feeding speed, and setting feed speed it becomes ready to execute line, arc, circular motion. When executing arc function, there is a special notice that the parameters needed are within the reasonable range. Example is as follows:

```
MCC_SetAbsolute(GROUP_INDEX);           // Use absolute coordinate to express
                                         the position of each axis

MCC_SetFeedSpeed(20, GROUP_INDEX); // Set feeding speed
MCC_Line(10, 10, 0, 0, 0, 0, GROUP_INDEX);
                                         // Notice that start point, reference point and end point should not
                                         be colinear

nRet = MCC_ArcXY(10, 20, 20, 20, GROUP_INDEX);

if (nRet != NO_ERR)
{
    /*
```

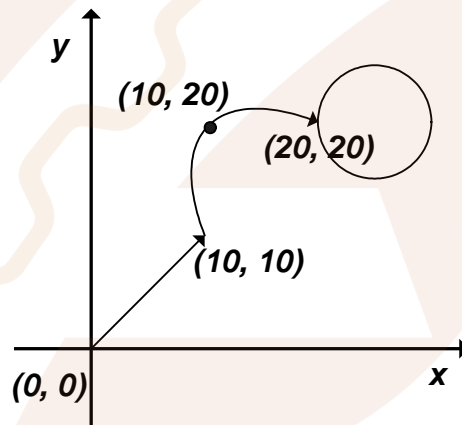
Use returned value to identify the reason of error, if parameter error then send back

value PARAMETER\_ERR is returned.

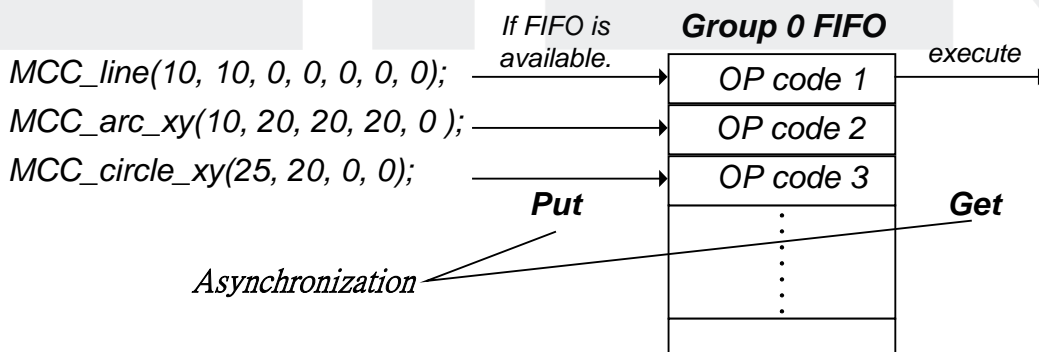
```
*/
}
```

```
MCC_CircleXY(25, 20, 0, GROUP_INDEX);
```

Use returned value to identify the reason of error. Regarding the meaning of returned error, please refer to “PCI-4P motion library user’s manual”. Trajectory of motion is shown as follows:



The process of executing motion function is that motion function put op code in each FIFO of each group first, then MCCL get the commands from FIFO’s of different groups simultaneously and execute sequentially. It doesn’t have to wait for one motion to finish to execute another motion. If FIFO is full, the returned value of function is COMMAND\_BUFFER\_FULL\_ERR, the size of each FIFO buffer is 10000. The figure shown below is the operation of FIFO for a certain group.



Thus, commands of the same group will be executed sequentially; but commands of different groups can be executed concurrently.



# ZETEK®

## 7. Point to point motion

### Related functions

```
MCC_SetAbsolute()
MCC_SetPtPSpeed()
MCC_PtP();
```

### Example program

PtPMotion.cpp

### Explanation

After finish setting mechanism parameter, starting system, setting maximum feeding speed, and setting feeding speed, it is possible to execute point to point motion. Used function example is as follows:

```
MCC_SetAbsolute(GROUP_INDEX);           // Use absolute coordinate to express
                                         the position of each axis
```

```
MCC_SetFeedSpeed(20, GROUP_INDEX); // Set feeding speed.
```

```
                                         // Motion with 20% of the maximum speed for each axis i.e. (RPM
                                         × pitch / gear_ratio) × 20%
```

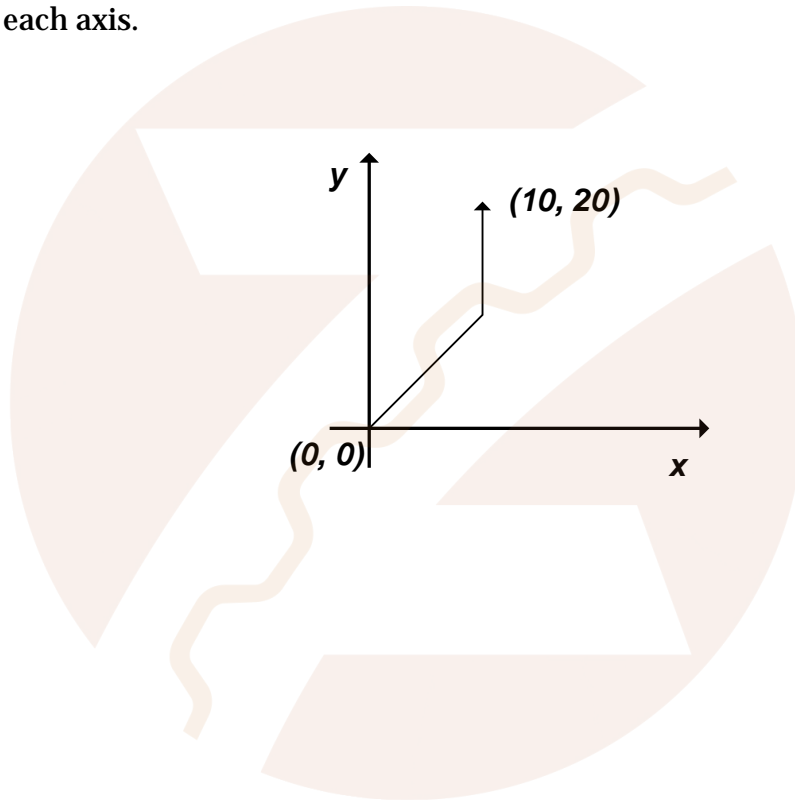
```
MCC_SetPtPSpeed(20, GROUP_INDEX);
```

```
// Move to (10,20),however motion of each axis start at same time and could arrive at
target at different time.
```

```
MCC_PtP(10, 20, 0, 0, 0, 0, GROUP_INDEX);
```

For point to point motion, each axis has independent motion profile. They

won't reach the target position at the same time though they start simultaneously, this is different from general motion. General motion are synchronized, it starts and ends a motion at the same time. The figure below is a point to point motion trajectory, assuming the maximum speed are the same for each axis.



# ZETEK®

## 8. JOG motion

### Related functions

```
MCC_SetUnit()  
MCC_JogPulse()  
MCC_JogSpace()  
MCC_JogConti()
```

### Example program

```
JogMotion.cpp
```

### Explanation

MCC\_JogPulse() uses pulse as unit, do small stroke motion for specific axis, but pulse value can not exceed 2048. MCC\_JogSpace() uses actual distance as unit(mm or inch), do also small stroke motion for specific axis. MCC\_JogConti() move continuously to the right/left limit position set by mechanism parameter. The speed parameters of MCC\_JogSpace() and MCC\_JogConti() are set as velocity percent. Setting is similar to that of point to point motion. Used example is as follows:

```
MCC_SetUnit(_MM_, GROUP_INDEX);    // Use mm as displacement unit.  
MCC_JogPulse(0, 100, GROUP_INDEX); // Axis 0 move 100 pulses  
  
// Move axis 0 a distance with speed of (wRPM × dfPitch/dfGearRation) ×10%  
MCC_JogSpace(-1, 10, 0, GROUP_INDEX);  
  
// Move axis 0 continuously with speed of (wRPM × dfPitch/dfGearRation) ×10%  
to right limit.  
MCC_JogConti(1, 10, 0, GROUP_INDEX);
```

## 9. Getting speed, position and command position

### Related functions

MCC\_GetCurFeedSpeed()  
 MCC\_GetFeedSpeed()  
 MCC\_GetCurPos()  
 MCC\_GetPulsePos()  
 MCC\_GetCurCommand()  
 MCC\_GetCommandCount()

### Example program

GetStatus.cpp

### Explanation

MCC\_GetCurFeedSpeed() is used to get the current feeding speed, MCC\_GetSpeed() is used to get the feeding speeds of each axis. MCC\_GetCurPos() is used to get the cartesian coordinates of current position of each axis, MCC\_GetPulsePos() is used to get the motor coordinates(or pulse coordinates) of current position of each axis. The relationship between cartesian coordinates and motor coordinates is motor coordinate = cartesian coordinate × (dfGearRatio /dfPitch) × dwPPR. Use MCC\_GetCurPos and MCC\_GetPulsePos() only when the channel of the group designated by GROUP\_INDEX is enabled , i.e. not -1, for the coordinate value of that axis to be effective. For example channel of a certain group is disabled when

```
group_config.stGroupInfo[GROUP_INDEX].nchannel[2] = -1;
```

In this case z axis coordinate got via MCC\_GetCurPos() or MCC\_GetPulsePos() will always be 0. Used example is shown as follows:

```
double dfCurPosX, dfCurPosY, dfCurPosZ, dfCurPosU ,dfCurSpeed, dfNOT_USED;
```



```
double dfCurSpeedX, dfCurSpeedY, dfCurSpeedZ, dfCurSpeedU;
long lCurPulseX, lCurPulseY, lCurPulseZ, lCurPulseU, lNOT_USED;

// Get current feeding speed.
dfCurSpeed = MCC_GetCurFeedSpeed(GROUP_INDEX);

// Get feeding speeds of each axis.
MCC_GetSpeed(&dfCurSpeedX, &dfCurSpeedY, &dfCurSpeedZ, &dfCurSpeedU,
&dfNOT_USED, &dfNOT_USED, GROUP_INDEX);

// Get cartesian coordinates of current position of each axis
MCC_GetCurPos(&dfCurPosX, &dfCurPosY, &dfCurPosZ, &dfCurPosU,
&dfNOT_USED, &dfNOT_USED, GROUP_INDEX);

// Get motor coordinates of current position of each axis
MCC_GetPulsePos(&lCurPulseX, &lCurPulseY, &lCurPulseZ, &lCurPulseU,
&lNOT_USED, &lNOT_USED, GROUP_INDEX);
```

Use MCC\_GetCurCommand() can obtain the information of motion command in execution, including motion command type 、 motion command encode 、 feed speed and target position etc. Use MCC\_GetCommandCount() can obtain the number of motion command which is saved and has not been executed in motion buffer area. User can compare motion command encode to the number of saved motion command. Used example is as follows:

```
Int nCommandCount;
```

```
COMMAND_INFO    stCommandInfo;

// Get motion command being executed

MCC_GetCurCommand( &stCommandInfo, GROUP_INDEX);

/*

    stCommandInfo.nType;      motion command type

                                0  point to point motion
                                1  linear motion
                                2  clockwise arc, circular motion
                                3  counterclockwise arc,circular motion

    stCommandInfo.nCommandIndex; Motion command encode

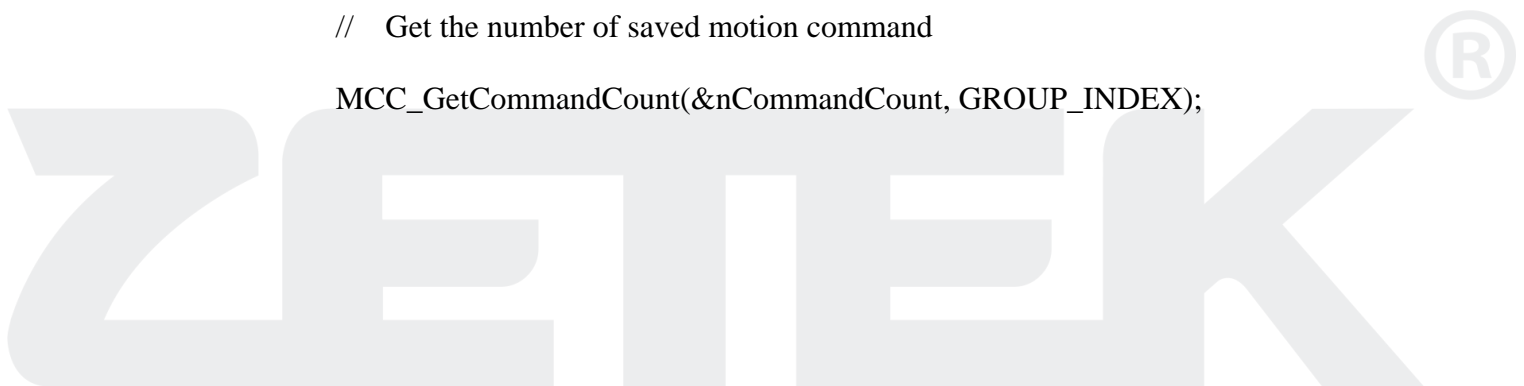
    stCommandInfo. DfFeedSpeed      feed speed

    stCommandInfo. dfPos[]          Target position coordinate

*/

// Get the number of saved motion command

MCC_GetCommandCount(&nCommandCount, GROUP_INDEX);
```

The ZETEK logo is displayed in a large, light gray, sans-serif font. The letters are bold and blocky. A registered trademark symbol (®) is located to the upper right of the 'K'.

## 10. Motion hold, continue, abort

### Related functions

MCC\_HoldMotion()  
 MCC\_ContiMotion()  
 MCC\_AbortMotion()

### Example program

CtrlMotion.cpp

### Explanation

MCC\_HoldMotion() is used to hold the current command in execution; MCC\_ContiMotion() is used to continue executing the current hold command. Use MCC\_ContiMotion() together with MCC\_HoldMotion() for using and they must be used for the same group. MCC\_AbortMotion() is used to abort the current command in execution ,it also clear rest motion commands in the buffer.

If there is no motion in execution, when call MCC\_AbortMotion(), the return value of the function will be ABORT\_ILLGEGAL\_ERR. If there is no motion in execution, when call MCC\_HoldMotion(), the return value of the function will be HOLD\_ILLGEGAL\_ERR; if calling MCC\_HoldMotion() before unsuccessfully, call MCC\_ContiMotion() the return value of the function will be CONTI\_ILLGEGAL\_ERR. Usually we will use the value returned by the function to judge if these functions are used not in the right time. Used example is shown as follows:

```
int nRet;

nRet = MCC_AbortMotion(GROUP_INDEX);

if (nRet != NO_ERR)

    printf("Erroneous return value : %d ", nRet);// Use the function not at the right
                                                time.
```

```
nRet = MCC_HoldMotion(GROUP_INDEX);  
  
if (nRet != NO_ERR)  
{  
    nRet = MCC_motion_abort(GROUP_INDEX);  
    if (nRet != NO_ERR)  
        printf("Erroneous return value : %d ", nRet); // Use the function not at the  
                                                    right time.  
}  
else  
    printf("Erroneous return value : %d ", nRet); // Use the function not  
                                                    at the right time.
```



# ZETEK®

## 11. Motion status

### Related functions

MCC\_GetMotionStatus()

### Example program

MotionFinished.cpp

### Explanation

Use return value of MCC\_GetMotionStatus() to check motion status. If returned value is 0, it means in motion; if returned value is 1, machine is in stop status, and there is no command in the motion buffer; After a successful call to MCC\_HoldMotion(), the returned value of MCC\_GetMotionStatus() is 2, which means machine is in hold status, and there are still unfinished commands in the buffer. Used example is shown as follows:

```
int nStatus;
```

```
MCC_Line(50, 50, 0, 0, 0, 0, GROUP_INDEX);
```

```
// Wait for motion finish
```

```
while (MCC_GetMotionStatus(GROUP_INDEX) != 1);
```

```
MCC_Line(20, 20, 0, 0, 0, 0, GROUP_INDEX);
```

```
nStatus = MCC_GetMotionStatus(GROUP_INDEX);
```

```
printf("Motion status : %d \r", nStatus);
```

## 12. Speed override

### Related functions

MCC\_SetOverSpeed()  
 MCC\_GetOverSpeed()  
 MCC\_SetPtPOverSpeed()  
 MCC\_GetPtPOverSpeed()

### Example program

OverSpeed.cpp

### Explanation

MCC\_SetOverSpeed() is used to set speed override for general motion. It is set is percent of former speed. MCC\_GetOverSpeed() is used to get the current over speed ratio. For point to point motion speedoverride ratio need to call MCC\_SetPtPOverSpeed() and MCC\_GetPtPOverSpeed(). Used example is as follows:

```
// Set general motion's feeding speed to 20 mm/ sec
MCC_SetFeedSpeed(20, GROUP_INDEX);
MCC_Line(10, 10, 0 ,0 ,0 ,0 ,0 ,0, GROUP_INDEX)

// Set speed override ratio, i.e. current speed
// will become 20 X150% = 30 mm/ sec
MCC_SetOverSpeed(150, GROUP_INDEX);

nRate = MCC_GetOverSpeed( GROUP_INDEX);// nRate should be 150

// Set point to point motion's feeding speed to 50 % of maximum speed of each axis.
// i.e. equal to (wRPM * dfPitch / dfGearRatio) x50 %
```

```
MCC_SetPtPSpeed(50, GROUP_INDEX);
```

```
MCC_PtP(10, 10, 0 ,0 ,0 ,0 ,0, GROUP_INDEX)
```

```
// Set point to point motion speed override ratio
```

```
// i. e. current speed will become (wRPM * dfPitch / dfGearRatio) 50 % × 150%
```

```
MCC_SetPtPOverSpeed(150, GROUP_INDEX);
```

```
nRate = MCC_GetPtPOverSpeed(GROUP_INDEX);// nRate should be 150
```



ZETEK®

### 13. Software limit check and limit switch check

**Related functions**

MCC\_SetOverTravelCheck()  
 MCC\_GetOverTravelCheck()  
 MCC\_EnableLimitSwitchCheck()  
 MCC\_DisableLimitSwitchCheck ()

**Example program**

CheckOT.cpp

**Explanation**

MCCL provides software limit check function. When software limit check is enabled, if travel range of any axis passes limit value, system will stop movement. To resume from this situation, it is necessary to disable software limit check, and clear system's error record (refer to the explanation later). dfHighLimit, dfLowLimit, dfHighLimitOffset, dfLowLimitOffset of mechanism parameters to set software limit's ; MCC\_SetOverTravelCheck() is used to enable and disable this, MCC\_GetOverTravelCheck() is used to check current status. Used example is shown as follows:

```
// Enable X axis software limit check.
MCC_SetOverTravelCheck(1, 0, 0, 0, 0, 0, GROUP_INDEX);

// For OT0 ~ OT3 a value of 1 means software limit check is enabled for the axis, 0
means disabled.
MCC_GetOverTravelCheck(&OT0, &OT1, &OT2, &OT3, &NOT_USED,
&NOT_USED, GROUP_INDEX);
```



```
nErrCode = MCC_GetErrCode(GROUP_INDEX);// Get error code
```

Use `MCC_GetErrorcode()` to judge if software limit is reached it cause system not to move. The returned value of 0xf301~0xf304 correspond to X axis ~ U axis. If this situation appears, please follow the example showed below to put system back to normal motion, but be very careful to avoid motor's collision.

```
// Disable software limit check
MCC_SetOverTravelCheck(0, 0, 0, 0, 0, 0, GROUP_INDEX);

// Clear error record in system , let it continue to do the motion normally
MCC_ClearError(GROUP_INDEX);
```

MCCL provides hardware limit switch check also, use `MCC_EnableLimitSwitchCheck()` to enable it. If `wOverTravelUpSensor` and `wOverTravelDownSensor` mode are set 2, it does not make sense to call `MCC_EnableLimitSwitchCheck()`, Calling `MCC_EnableLimitSwitchCheck(0)` or `MCC_EnableLimitSwitchCheck(1)` enable limit switch check . When limit switch is touched, motion control card will stop outputting pulse, but the motion command will keep execution internally, user must call `MCC_AbortMotion()` to stop current motion command in execution. If limit switch check function is not needed, call `MCC_DisableLimitSwitchCheck()` Call `MCC_GetLimitSwitchStatus()` to check if limit switch is touched.

## 14. Speed blend function

### Related functions

MCC\_EnableBlend()  
 MCC\_DisableBlend()  
 MCC\_CheckBlend()

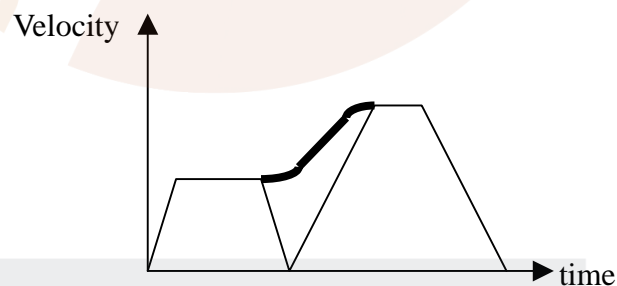
### Example program

SetBlend.cpp

### Explanation

MCCL provides speed blend function, the description of trajectory and speed blend in motion, please refer to “**PCI-4P Motion Library User’s Manual**”.

It is used to connect two different velocity profile smoothly. In the drawing the speed of first move command transfer continuously to the second move command. Thus the motion time is shorter.



Calling `MCC_EnableBlend()` and `MCC_DisableBlend()` enables and disables speed blend function separately, call `MCC_CheckBlend()` get current setting. A returned value of 0 means enabled ; 1 means disabled.

## 15. Get and clear error status

### Related functions

```
MCC_GetErrorCode()  
MCC_ClearError()
```

### Example function

```
ErrorStatus.cpp
```

### Explanation

Error occurred in system, if its cause has been cleared/removed, it is still necessary to call `MCC_ClearError()` to clear error record in the system, otherwise motion won't continue. It is advised to get current error code regularly to check if there is any error. Used example is shown below. Also refer to software limit check and limit switch check.

```
nErrcode = MCC_GetErrCode(GROUP_INDEX);  
if (nErrCode != 0)  
{  
  
    /*  
    Do necessary procedure  
    */  
  
    MCC_ClearError(GROUP_INDEX); // Clear system's error record.  
}
```

Exception: while using arc motion command, a error code of 0xf104 means illegal calling parameter (points can not be collinear). In this case, there is no need to call `MCC_ClearError`.

## 16. Homing

### Related functions

MCC\_SetGohomeAccStep()  
MCC\_GetGohomeAccStep()  
MCC\_GoHome()  
MCC\_GetGohomeStatus()  
MCC\_AbortGoHome()

### Example program

GoHome.cpp

### Explanation

Homing parameters are set in HOME\_CONFIG of mechanism parameter (please refer to “**PCI-4P Motion Library User’s Manual**”). The default steps of homing acceleration and deceleration are 10 steps, i. e. the preset time for acceleration, deceleration are  $10 \times$  interpolation Time. To change them for smother motion use MCC\_SetGohomeAccStep().

Use MCC\_GetGoHomeStatus() to check if homing motion is finished, Use MCC\_AbortGoHome() in the process of homing to abort it.

The homing provided by MCCL currently can only respond to one motion control card at a time. In case of homing with several cards, call MCC\_GetGoHomeStatus() to check if finished , then proceed next card to call MCC\_GoHome() for homing.

Used example is shown below:

```
// Set acceleration, deceleration steps of homing procedure, if not being set, the default step is 10.
```

```
MCC_SetGoHomeAccStep(15, 15, 15, 15, 0, 0);
```

```
// 0xff means t axis won't do homing.
```

```
MCC_GoHome(10, 10, 0, 0, 0, 0, 0, 0, 0, 0xff, 0xff, 0xff, 0xff, CARD_INDEX);
```

```
•
```

```
•
```

```
MCC_AbortHome();// If needed, use this function to abort homing.
```

```
•
```

```
•
```

```
nStatus = MCC_GetGoHomeStatus();
```

```
// Use return value to check if homing procedure is done, if
```

```
// nStatus value is 1, it has finished
```



ZETEK®

## 17. Acceleration, deceleration step setting

### Related functions

```
MCC_SetAccStep()  
MCC_SetDecStep()  
MCC_GetAccStep()  
MCC_GetDecStep()  
MCC_SetPtPAccStep()  
MCC_SetPtPDecStep()  
MCC_GetPtPAccStep()  
MCC_GetPtPDecStep()
```

### Example program

```
AccStep.cpp
```

### Explanation

The default value is 10 steps for acceleration, deceleration of general motion (including line, arc, circular motion) and point to point motion, i.e. preset time for acceleration, deceleration is  $10 \times$  interpolation Time, Use `MCC_SetAccStep()`, `MCC_SetDecStep()`, `MCC_SetPtPAccStep()`, `MCC_SetPtPDecStep()` to change settings.

After setting feeding speed (for example call `MCC_SetFeedSpeed ()` or `MCC_SetPtPSpeed()`), acceleration and deceleration steps should be adjusted ; for example when higher feeding speed is set, bigger acceleration and deceleration steps are better, especially for stepping motors.

In `AccStep.cpp`, as speed varies, different acceleration and deceleration steps are used. Calculate the acceleration into  $m/s^2$  would help designing the velocity profile.