HIWIN®

PDL Examples for D-series Drives

User Manual

www.hiwinmikro.tw

Revision History

Release Date	Version	Applicable Product	Revision Contents
Jun. 30th, 2014	3.0	D-series drive	First edition.
Sep. 11th, 2015	4.0	D-series drive	Update the information to match the source document.
Jan. 14th, 2019	5.0	D-series drive	Revise example 3 and 8.

Table of Contents

1. Homing by directly searching index 1
2. Homing by touching hard stop 5
3. Use I/O to trigger a move 11
4. Unit conversion 15
5. Teaching function 19
6. Re-enable to resume an unfinished absolute positioning 25
7. Jog motion by digital inputs 29
8. Jog motion by analog inputs 33
9. Continuous motion in force / torque mode 37

(This page is intentionally left blank.)

1. Homing by directly searching index

Example number	1
File name	1.pdl
Drive model	D1-series drive
Firmware requirement	Lightening 0.117 MDP 0.168 and above
Function	Trigger digital input I2 to execute the built-in function of homing. As soon as homing is done, digital output O2 outputs a pulse to the host controller.
Used input	I2
Used output	O2

Sample code

```
// ============================ Macro ============================
#define pulse_width      20                    //  Set output pulse width (ms)
// ========================= Main Program =========================
#task/1;                                       //  Create task #1
_input_home_loop:
till(~I2);                                     //  Wait for I2 to be OFF
setoff O2;                                     //  Set O2 to be OFF
till(I2);                                      //  Wait for I2 to be ON
call _go_home;
goto _input_home_loop;
ret;
// ========================= Homing Process =========================
_go_home:
call _X_init_exec;                             //  Call the homing function built in the drive
sleep 100;                                     //  Delay 100 ms
till(X_I_flag<>1);                             //  Wait for the homing process to end
if(X_I_flag<>2) do                             //  If X_I_flag is not 2, homing fails.
    retprintl/101("Homing process failed.");
goto _input_home_loop;
end;
// After homing succeeds, the drive outputs a pulse to the host controller
seton O2;                                      //  Set O2 to be ON
sleep pulse_width;                             //  Delay a pulse width
setoff O2;                                     //  Set O2 to be OFF
ret;
```

How to use

This sample code is to execute the built-in homing function by triggering digital input I2. As the homing process finishes, digital output O2 outputs a pulse to the host controller. Pulse width can be changed by users. The flow chart of this sample code is given in Figure 1-1.

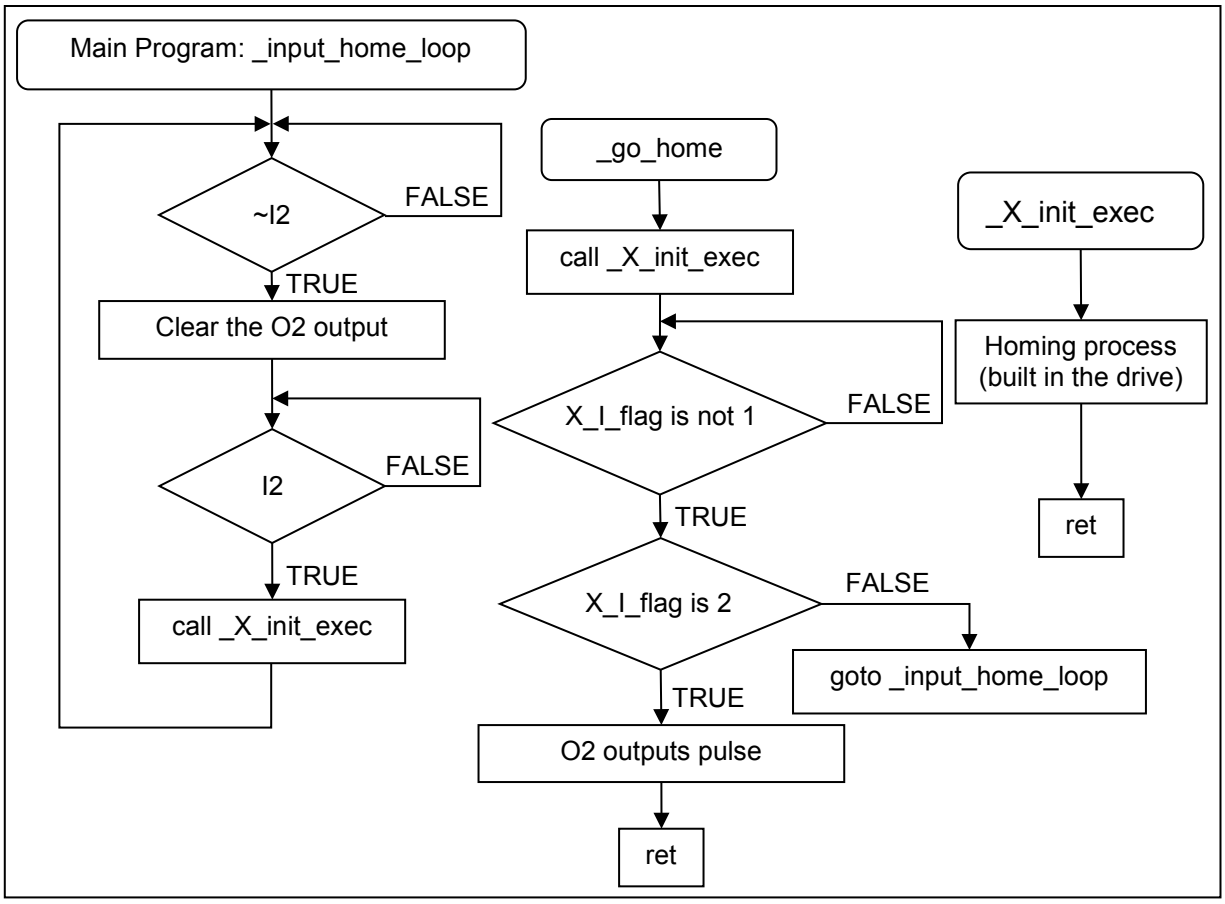


Figure 1-1

Application explanation

At the beginning of main program, a task is initiated, called **#task/1**. This is to assign a task number in user-edited PDL application program in the multitasking process of drive. For D-series drives, PDL is able to execute up to the maximum 4 tasks (#0~#3) at the same time. In these four tasks, #task/0 is occupied by the drive's operation system (sys.pdl). Therefore, there are only three tasks (#1~#3) available for users.

In the drive's operation system (OS), there is a function named **_X_init_exec**. This function is for homing, and is built in the drive. By using **call _X_init_exec** in the programming code, the homing method is performed by directly searching the index signal. Variables for homing can be set at the "Homing" page in the "Application center" of Lightening program, as Figure 1-2 shows.

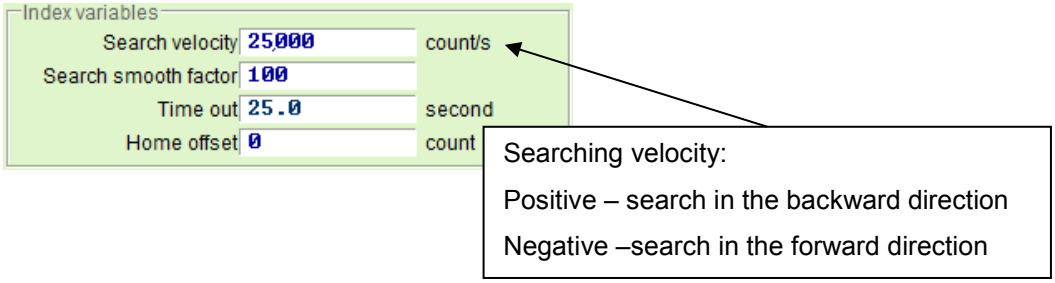


Figure 1-2 Homing variables setting

In this sample code, there is a variable named **X_I_flag** for showing the homing status, as Table 1-1 shows.

X_I_flag	Homing status
-1	Homing fails
0	Homing not executed
1	Homing
2	Homing succeeds

Table 1-1 Definition of **X_I_flag**

When coding a PDL application program, if a digital input is for triggering an event, users should set this input to be “Not Configured” to avoid a conflict.

In this sample code, the setting of pulse width is used to extend the period of output signal to the host controller. By doing so, the host controller can avoid losing the signal sent by O2.

2. Homing by touching hard stop

Example number	2
File name	2.pdl
Drive model	D1-series drive
Firmware requirement	Lightening 0.117 MDP 0.168 and above
Function	During homing, the motor first moves toward the negative direction until it touches hard stop and output current reaches pre-defined threshold. After that, the motor turns around and moves toward the positive direction to search index and end homing.
Used input	I2
Used output	null

Sample code

```
// ============================ Macro ============================
#define BackDistance      10000                  //  Set the distance of leaving hard stop
#define CurrThreshold      0.05                  //  Set the threshold for output current
#define JogVel      25000                        //  Set the searching hard stop speed
// ========================= Main Program =========================
#task/1;                                         //  Create task #1
#long OriginalHomeVel;
#long HomeVel;
_input_home_loop:
till(~I2 & X_ready);                             //  Wait for I2 to be OFF and servo ready
till(I2);
call _go_home;
goto _input_home_loop;
ret;
// ========================= Homing Process =========================
_go_home:
OriginalHomeVel = X_index_vel;                   //  Record the original homing speed
//   Process of searching hard stop
printl/101("Start to search negative hard stop.");
X_jvl = -JogVel;                                 //  Do jog toward the negative direction
till (X_curr_abs > CurrThreshold);               //  Wait until the current threshold is reached
X_stop_m = 1;                                    //  Stop motion
if(X_en=0) do
    printl/101("Search negative hard stop failed!");
    goto _input_home_loop;
end;
```

```
till (~X_run);
printl/101("Negative hard stop found, going back.");
X_trg = X_ref_pos + BackDistance;          //  Move the pre-defined distance away from the hard stop
sleep 100;
till (~X_run);
//  Process of searching index
printl/101("Start to search index.");
HomeVel = abs (X_index_vel);
X_index_vel = -HomeVel;                    //  Set the homing speed and direction
call _X_init_exec;                         //  Call the built-in homing function
sleep 100;
till(X_I_flag<>1);
if(X_I_flag<>2) do
    printl/101("Homing process failed!");
else do
    printl/101("Homing process successful.");
end;
ret;
```

How to use

In the homing process, users are able to select between limit switch or touching hard stop. This makes sure that the index searching can start at the same reference point with the same direction every time. If there is no limit switch, users can choose the method of touching hard stop as the reference point.

The method of touching hard stop is to touch hard stop at the same side, and then go to reverse direction to search index signal. This can obtain a well home repeatability. The flow chart for homing method of touching hard stop is shown in Figure 2-1.

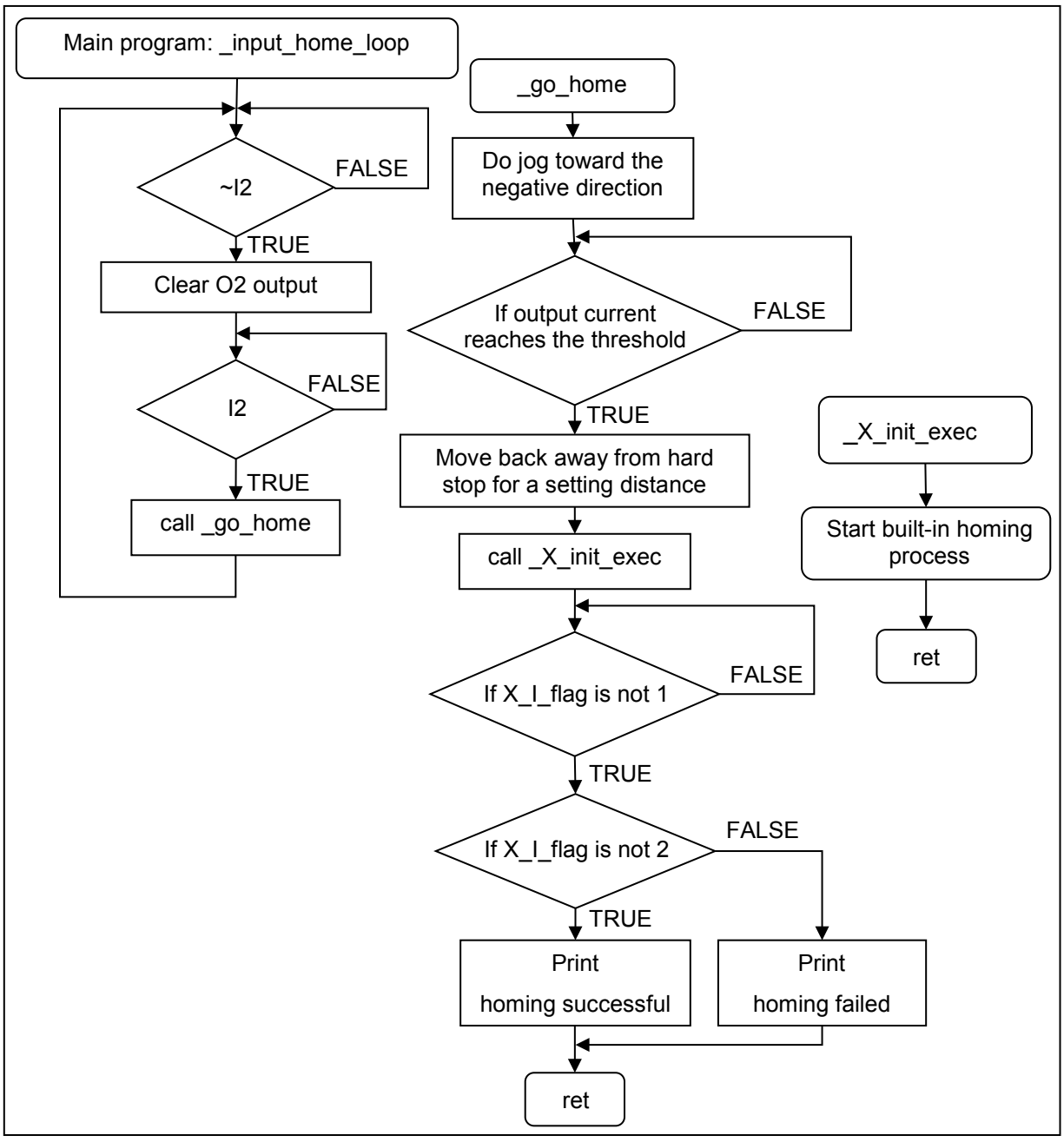


Figure 2-1

At the beginning, users have to set the threshold for output current (**CurrThrshold**), which is effected by loadings and frictions (e.g. screw and guideway). Therefore, users can set this threshold by observing the actual output current in the “Scope” of Lightening. The setting process for current threshold is described as follows.

Step 1. Complete the gain tuning.
Step 2. Open the Scope, and set to observe the Actual Current.
Step 3. Run the whole stroke with the speed of searching hard stop (**JogVel**).

Step 4. Record the maximum value of Actual Current during motor moving.

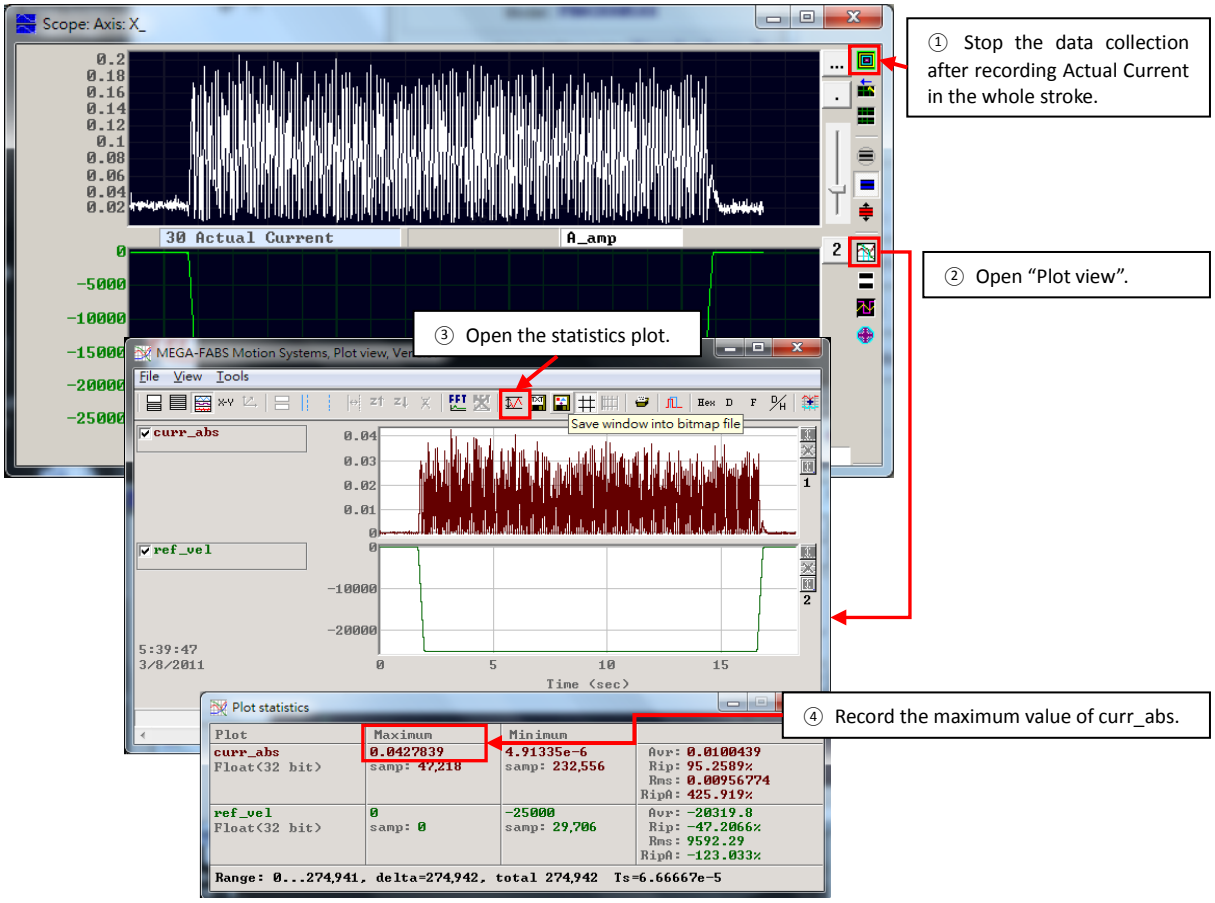


Figure 2-2 Observe the Scope and set the current threshold

Step 5. Set the current threshold slightly larger than the **curr_abs** (unit: A^2). In the example of Figure 2-2, the **CurrThreshold** can be set as 0.05.

Note: The current threshold should not be set too large. Otherwise, the error message of "Soft thermal threshold reached" may happen as the motor touches hard stop.

Application explanation

If users want to search index after the limit switch is triggered, just modify this sample code slightly and set the input function of limit switch in Lightening as the following ways.

(1) Lightening setting

Set the input function of limit switch in the “I/O Center”. The default setting of D1 drive is shown in Figure 2-3.

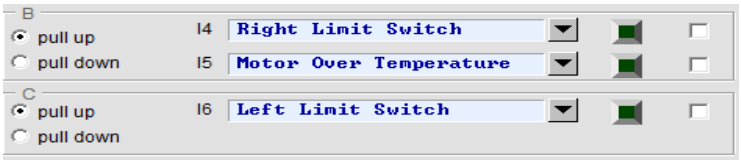


Figure 2-3 The default setting of digital input

(2) Modify this sample code

Since the left limit switch is set at I6 in Fig.1.5, users have to modify **till (X_curr_abs > CurrThreshold)** as **till (I6)**. In this case, there is no need to set current threshold.

In this sample code, **X_stop_m** was used to stop motor and **X_index_vel** was used to set the speed and direction of searching index. Take this sample code as an example. **X_stop_m = 1** is used to stop motor, and **X_index_vel** is the same as the value of “Search velocity” in Figure 1-2.

3. Use I/O to trigger a move

Example number	3
File name	3.pdl
Drive model	D1-series drive
Firmware requirement	Lightening 0.117 MDP 0.168 and above
Function	Use input I2 and I3 to trigger a motor to move a pre-defined distance. I2 and I3 are used to let motors move forward and backward respectively.
Used input	I2, I3
Used output	null

Sample code

```
// ============================ Macro ============================
#define distance      131072                    //  Set moving distance (count)
// ========================= Main Program ========================
#task/1;                                        //  Create task #1
_trigger_loop:
till(X_ready);                                  //  Wait for motor to be enabled

if (I2& ~I3) do                                 //  If I2 is ON and I3 is OFF, move forward.
    X_trg = X_ref_pos + distance;               //  Move forward a pre-defined distance
    sleep 50;                                   //  Delay 50 ms
    till (~X_run);                              //  Wait for motor to stop
    printl/101("Positive motion ended.");
end;

if (I3 & ~I2) do                                //  If I3 is ON and I2 is OFF, move backward.
    X_trg = X_ref_pos - distance;               //  Move backward a pre-defined distance
    sleep 50;
    till (~X_run);
    printl/101("Negative motion ended.");
end;

till(~I2 & ~I3);                                //  Wait for I2 and I3 to be OFF
goto _trigger_loop;
ret;
```

How to use

Users should set the moving distance first. Then, trigger the digital input I2 or I3 to let motor move in the forward or backward direction, respectively. Before **_trigger_loop** starts running every time, the statuses of I2 and I3 must be set to **OFF** and the motor must be enabled. Functions of I2 and I3 are given in Table 3-1.

I2	I3	Motion direction
ON	OFF	Forward
OFF	ON	Backward

Table 3-1 Functions of I2 and I3

Application explanation

In PDL, **X_trg** is the command to trigger an absolute positioning. When the value of **X_trg** is different from the feedback value of encoder, the motor will move to the position of **X_trg**. In addition, users are allowed to change the value of **X_trg** when the motor is moving. In this case, the motor will move to the latest setting position of **X_trg**.

Using **X_trg** to realize a relative positioning, a reference point has to be set at coding. There are two examples given as follows.

(1) Using feedback position

Feedback position is used as the reference point.

X_trg = X_enc_pos + distance;

In this case, the accumulation of position error will be a problem due to jitter issues. If users adopt feedback position as reference point, an unsuitable reference point will be obtained, since the motor may move over or short of position (jitter issue). Therefore, the cumulative error becomes obvious after running for many times.

(2) Using reference position

To solve the problem in the above example, the reference position is used as the reference point.

X_trg = X_ref_pos + distance;

A final tip in this example is **printl/retprint** command. They are used to show messages on “Message + command prompt” window.

(This page is intentionally left blank.)

4. Unit conversion

Example number	4
File name	4.pdl
Drive model	D1-series drive
Firmware requirement	Lightening 0.117 MDP 0.168 and above
Function	Automatically convert the unit from “mm” or “deg” to “count”. Thus application engineers do not have to bother unit conversion in later applications.
Used input	null
Used output	null

Sample code

```
// ============================ Macro ============================
#define distance      45                          //  Set moving distance (mm or deg)
// ========================= Main Program =========================
// =   Description:  For linear motor – convert mm to count
// =                 For AC servo motor with ball screw – convert mm to count
// =                 For torque motor – convert deg to count
// =   Arguments:   null
// ================================================================
#task/1;                                          //  Create task #1
#float LMCntPerMM;
#float disTemp;
if (X_rotaryType=2) do                            //  If the motor is an AC servo motor
    disTemp = distance;
    disTemp = disTemp / X_pitchScrew;             //  Unit conversion - mm to rev
    disTemp = disTemp * X_cntperunit;             //  Unit conversion - rev to count
end;

if (X_rotaryType=1) do                            //  If the motor is a torque motor
    disTemp = distance;
    disTemp = disTemp / 360;                      //  Unit conversion– deg to rev
    disTemp = disTemp * X_cntperunit;             //  Unit conversion– rev to count
end;

if (X_rotaryType=0) do                            //  If the motor is a linear motor
    LMCntPerMM = X_cntperunit / 100;
    //   Convert the resolution unit from count / 100mm to count / mm
    disTemp = distance;
```

```
    disTemp = disTemp * LMCntPerMM;          //  Unit conversion - mm to count
end;

ret;
```

How to use

In PDL, all motion variables (i.e. speed **X_vel_max**, acceleration **X_acc**, deceleration **X_dcc**, and so on) and position variables use “count” as unit. This sample code demonstrates unit conversion before executing any motion.

For example, assume that a linear motor needs to move 45 mm from the current position (as this sample code shows, **#define distance 45**). Since the calculated result is saved into **disTemp**, the following command can make motor move 45 mm.

X_trg = X_ref_pos + disTemp;

Application explanation

In this sample code, there is a built-in variable named **X_pitchScrew**. This variable represents the screw pitch of AC servo motor. In addition, **X_rotaryType** represents the motor type, as Table 4-1 shows.

X_rotaryType	Motor type
0	Linear motor
1	Torque motor
2	AC servo motor

Table 4-1 Definition of **X_rotaryType**

In this example, there is another built-in variable **X_cntperunit**. This variable corresponds to the resolution of encoder. The definition of **X_cntperunit** for each motor type is given in Table 4-2.

Motor type	X_cntperunit
Linear motor	count / 100mm
Torque motor	count / rev
AC servo motor	count / rev

Table 4-2 Definition of **X_cntperunit** for each motor type

(This page is intentionally left blank.)

5. Teaching function

Example number	5
File name	5.pdl
Drive model	D1-series drive
Firmware requirement	Lightening 0.117 MDP 0.168 and above
Function	The example allows users to teach and record desired positions by triggering digital inputs. After that, users can make the motor move to taught points by triggering the corresponding digital inputs.
Used input	I2, I3, I4, I6
Used output	null

Sample code

```
// ============================ Macro ============================
#define BufferSize      4                       //  Set moving distance (mm or deg)
// ========================= Global Variable =========================
#short BufferNum;                               //  Position buffer index
#long PosBuff[BufferSize];                      //  Position buffer for storing each taught point
// ========================= Main Program =========================
#task/1;                                        //  Create task #1
BufferNum=0;                                    //  Initialize the index of position buffer

_trigger_loop:
if (I2) do                                      //  If I2 is ON, call the process of teaching point.
    call _PosMem;                               //  Call the process to teach point
    till (~I2);
end;
if (I6) do                                      //  If I6 is ON, call the process of moving.
    if (X_en=0) do                              //  If motor is disabled
        printl/101("Motor is not enabled!");    //  Print the warming message for disabled motor
        till (~I6);
        goto _trigger_loop;                     //  When I6 = 0, go back to _trigger_loop.
    end;
    call _Move2Tar;                             //  Call the process of moving to a teaching point
    till (~I6);
end;
goto _trigger_loop;
ret;
```

```
// ======================= Teaching (PosMem) =======================
// =   Description:  Save position in position registers PosBuff according to position index BufferNum
// =   Arguments:   null
// =================================================================
_PosMem:
#long posTemp;
posTemp=X_enc_pos;
PosBuff[BufferNum]= posTemp;                    //   Save feedback position to PosBuff
printl/103("Teaching point %ld, Position=%ld", BufferNum, posTemp);
//   Print the saved number and position of taught point
BufferNum+=1;                                   //   Move to next index of PosBuff
      if (BufferNum=BufferSize) do              //   If index exceeds the buffer size
            BufferNum=0;                        //   Go back to the 1st buffer
      end;
ret;
// ================== Move to taught point (Move2Tar) ==================
// =   Description:  Choose the target position by I4 and I3
// =   Arguments:   null
// =====================================================================
_Move2Tar:
#short index;
if (~I4 & ~I3) do                               //   (I4, I3) = (0, 0)
      index =0;
end;
if (~I4 & I3) do                                //   (I4, I3) = (0, 1)
      index=1;
end;
if (I4 & ~I3) do                                //   (I4, I3) = (1, 0)
      index=2;
end;
if (I4 & I3) do                                 //   (I4, I3) = (1, 1)
      index=3;
end;
X_trg=PosBuff[index];                           //   Move to selected taught point
till (~X_run);                                  //   Wait for motor stop
printl/103("Move to point %ld ended.", index);  //   Print the in-position information
printl/103("Position=%ld", PosBuff[index]);     //   Print the position
sleep 100;
ret;
```

How to use

The teaching function is triggered by digital inputs. Functions of used inputs are given in Table 5-1.

Digital input	Function
I2	Rising-edge trigger. Teach position. Record the current position into buffer.
I3	Choose a teaching point.
I4	
I6	Rising-edge trigger. Move to the selected taught point.

Table 5-1 Functions of used inputs

As I2 is triggered (rising-edge trigger) once, the teaching process is performed once. Table 5-2 shows the relation of the triggering times of I2 to teaching point. If I2 has been triggered over 4 times, the later trigger will sequentially overwrite the previous teaching point beginning from the position 0. If users want to restart the teaching process, they can only do it by resetting the drive or restarting the 24V power supply.

Note: All previous teaching point will be cleared after doing reset. In addition, all initial teaching points are set to be 0.

Index	Teaching point
0	Record the current position to teaching point 0
1	Record the current position to teaching point 1
2	Record the current position to teaching point 2
3	Record the current position to teaching point 3

Table 5-2 Relation of the triggering times of I2 to teaching point

Before starting a motion to teaching point, users have to set the moving speed, acceleration, and corresponding motion variables in “Performance center” first. Then, use I4 and I3 to select teaching point as target position, and use I6 to trigger the motion. Table 5-3 shows the relation of digital inputs to action.

Input			Action
I4	I3	I6	
OFF	OFF	Rising-edge trigger	Move to teaching point 0
OFF	ON		Move to teaching point 1
ON	OFF		Move to teaching point 2
ON	ON		Move to teaching point 3

Table 5-3 Relation of digital inputs to action

Application explanation

To avoid conflict with input functions, users have to set the used inputs to be “Not Configured” when these inputs are used in PDL.

(This page is intentionally left blank.)

6. Re-enable to resume an unfinished absolute positioning

Example number	6
File name	6.pdl
Drive model	D1-series drive
Firmware requirement	Lightening 0.117 MDP 0.168 and above
Function	As a motor moves in the absolute coordinate, it may be disabled during the motion due to the occurrence of an error. After the error is fixed and the motor is enabled again, it will move to the original target position, which was set before disabling.
Used input	I2, I3
Used output	null

Sample code

```
// ============================ Macro ============================
#define TargetPos1      0                       //  Target position 1 (count)
#define TargetPos2      100000                  //  Target position 2 (count)
// ========================= Main Program =========================
#task/1;                                        //  Create task #1
#long targetpos;
#long MotionCompleteFg;                 //  Flag is used to identify whether motor is at target or not
MotionCompleteFg = 1;                   //  Motor arrives at the target position
_TargetMove:
till (~I2 & ~I3);
till (I2 | I3);
if (I2 & ~I3) do
    targetpos = TargetPos1;
    MotionCompleteFg = 0;               //  Motor does not arrive at the target position
end;
if (~I2 & I3) do
    targetpos = TargetPos2;
    MotionCompleteFg = 0;
end;
while (MotionCompleteFg = 0) do
    till (X_ready);                     //  If motor stops due to an error, wait here until it is re-enabled.
    X_trg = targetpos;                  //  Start to move to the target position.
    sleep 100;
    till (~X_run);                      //  Wait for motion to be completed, or stopped due to an error.
    if (X_ready) do
        MotionCompleteFg = 1;           //  If motor successfully arrives, leave the while loop.
```

```
    end;
end;
goto _TargetMove;
ret;
```

How to use

After users set absolute target positions (**TargetPos1**, **TargetPos2**), they can choose the target position by triggering I2 and I3 (Table 6-1). After that, the motor will move toward the chosen position. If the motor stops and is disabled during the motion due to occurrence of an error, it will wait until the error is fixed. After the motor is enabled, **X_trg** will be executed again to keep moving toward the target position. The motor stops motion until it arrives at the target position.

I2	I3	Variable name	Target position
ON	OFF	TargetPos1	0 count
OFF	ON	TargetPos2	100000 count

Table 6-1 Functions of digital inputs

Application explanation

In this and previous sample codes, **X_trg** is used to do motion in the absolute coordinate. When the value of **X_trg** is not equal to the value of **X_enc_pos**, the motor will move toward **X_trg** until **X_enc_pos** is equal to **X_trg.** Then, the motor stops motion.

When a motor stops due to disable, **X_trg** will be automatically set to the value of current **X_ref_pos**. At this time, any move of motor only changes the value of encode feedback **X_enc_pos**, but **X_ref_pos** and **X_trg** remain the same until the motor receives an enable command again. As the motor receives an enable command again, **X_ref_pos** and **X_trg** will be set to the current encoder feedback **X_enc_pos**. Therefore, when an error occurs to disable motor, in the normal condition, the re-enabled motor will stay instead of moving.

Moreover, this sample code designs a “while loop” and declares a variable **MotionCompleteFg** to represent the situation of motor arriving at the target position. When the motion of motor is completed, the program will check whether the motor is enabled or disabled. If the motor is enabled, it means that the motor indeed arrives at the target position. If the motor is disabled, it means that the motor does not arrive at the target position.

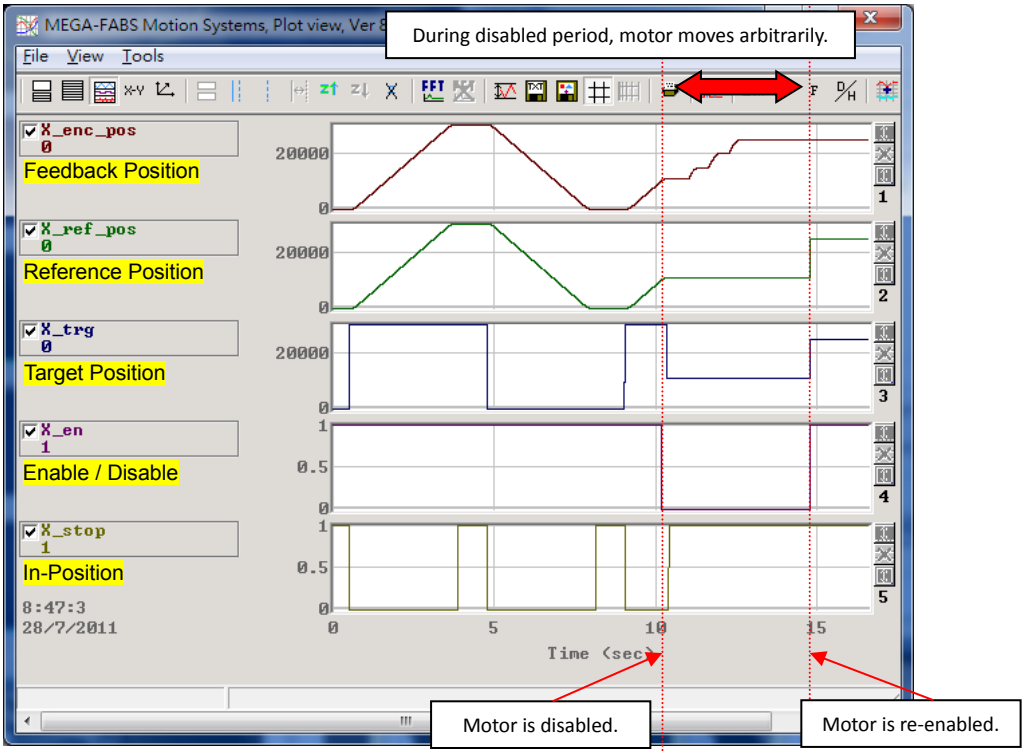


Figure 6-1 The variation of target position under different conditions

7. Jog motion by digital inputs

Example number	7
File name	7.pdl
Drive model	D1-series drive
Firmware requirement	Lightening 0.117 MDP 0.168 and above
Function	Let users trigger Jog motion through drive's inputs.
Used input	I2, I3
Used output	null

Sample code

```
// ============================ Macro ============================
#define CW_JOG I2                                   //  Define I2 as the port for forward jog
#define CCW_JOG I3                                  //  Define I3 as the port for backward jog
// ======================== Main Program ========================
#task/1;                                            //  Create task #1
_loop:
    if (X_en=1) do                                  //  If motor is enabled
        if (CW_JOG) do                              //  If the input for forward jog is ON
            X_jvl=X_vel_max;                        //  Do motion in the forward direction due to the value
                                                    //  of maximum speed in the "motion protection"
            till(~CW_JOG);                          //  Until the input for forward jog is released
            X_stop_m=1;                             //  Stop motion
        end;
        if (CCW_JOG) do                             //  If the input for backward jog is ON
            X_jvl=-X_vel_max;                       //  Do motion in the backward direction due to the
                                                    //  value of maximum speed in the "motion protection"
            till(~CCW_JOG);                         //  Until the input for backward jog is released
            X_stop_m=1;
        end;
    else do
        printl/101("Motor is not enabled!");        //  A warning message for disabled motor
        printl/103("Please release jog button before enabling motor.");   //  Ask to release jog command
        till(X_en=1 & ~CW_JOG & ~CCW_JOG);
        //  Until the motor is enabled and the jog input is released
    end;
goto _loop;
ret;
```

How to use

This sample code uses digital inputs to do Jog. Table 7-1 shows functions of digital input I2 and I3.

Digital input	Function
I2	Forward jog
I3	Backward jog

Table 7-1 Functions of digital input I2 and I3

The motion variables in this sample code refer to variables in “motion protection” of Lightening, as Table 7-2 shows.

Motion variable	Referring variable in “motion protection”
Velocity	Speed
Acceleration	Acc.
Deceleration	Dec. Kill

Table 7-2 Relation of motion variables to variables in “motion protection”

Note: Variables must be set before doing Jog.

Application explanation

At the beginning of this sample code, the macro of **#define CW_JOG I2** is used to make **CW_JOG** be pointed to I2. This method makes the program more flexible. Take the forward Jog (**CW_JOG**) in this sample code as an example. If users want to change the input from I2 to I12, just modify **#define CW_JOG I2** to **#define CW_JOG I12**. There are three built-in variables used in this sample code, which are illustrated in Table 7-3.

Built-in variable	Illustration
X_jvl	Set the speed of Jog. (unit: count/s)
X_vel_max	This corresponds to the maximum speed set by users in “motion protection”. (unit: count/s)
X_stop_m	Set it as 1 to stop the motor.

Table 7-3 Illustration of three built-in variables

In the following two cases, **X_trg** (mentioned in the application explanation of Chapter 3) cannot be used with **X_jvl** (mentioned in this sample code) at the same time.

(1) Case 1

When users set **jvl** to be nonzero to let a motor move, **trg** is changeable. However, **trg** is ignored by the drive and the motor keeps executing Jog with the value of **jvl**. If users want to stop the motor, the value of **stop_m** should be set to 1. Then, **jvl** will automatically be set to 0 to stop the motor, and **trg** is set to be the value of **enc_pos**.

(2) Case 2

When users set **trg** and make the motor move, **jvl** cannot be changed by users, and its value remains 0 during motion.

8. Jog motion by analog inputs

Example number	8
File name	8.pdl
Drive model	D-series drive
Firmware requirement	D1: Lightening 0.117 MDP 0.168 and above D2: Lightening 0.152 MDP 0.006 and above
Function	In position mode, it uses analog voltage input (0V~+10V) to control speed of motor and uses digital inputs to control direction of motion.
Used input	I2, I3, Ref+, Ref-
Used output	null

Sample code

```
// ============================ Macro ============================
#define JOG_VEL_SCALE      300              //  Set the ratio of input voltage to reference velocity
                                            //  as 1V = 300 rpm
// ======================= Unit Conversion proc =======================
proc Unit_Transform_Vel(float *velTemp) do
    *velTemp /= 60;
    *velTemp *= X_cntperunit;
end;
// ========================= Main Program =========================
#task/1;                                    //  Create task #1
#float JOG_VEL_VOL, JogVel_Scale;
if ( X_rotaryType <> 2 ) do                 //  Check if the motor is an AC servo motor
    goto _task_end;                         //  If it is not, finish the main program.
else do
    JogVel_Scale = JOG_VEL_SCALE;
    Unit_Transform_Vel(&JogVel_Scale);      //  Do unit conversion proc
end;
_loop:
till ( X_ready );//   Wait for servo ready
JOG_VEL_VOL = Vcmd*JogVel_Scale;            //  Read voltage from Ref+/Ref- and
                                            //  transform it to reference velocity
if ( I2 ) do
    if ( I3 ) do                            //  (I2, I3) = (1, 1)
        X_jvl = 0;                          //  Stop motion
    else do                                 //  (I2, I3) = (1, 0)
        X_jvl = JOG_VEL_VOL;                //  Do forward jog
    end;
```

```
else do
    if ( I3 ) do                        //  (I2,I3) = (0,1)
        X_jvl = -JOG_VEL_VOL;           //  Do backward jog
    else do                             //  (I2,I3) = (0,0)
        X_jvl = 0;                      //  Stop motion
    end;
end;
goto _loop;
_task_end:
ret;
```

How to use

This sample code uses built-in variable **Vcmd** to get the analog voltage input to control motion speed, and uses digital input I2 and I3 to control motion direction (forward, backward, or stop).Table 8-1 shows functions of digital input I2 and I3.

I2	I3	Function
0	0	Motion stop
0	1	Backward jog
1	0	Forward jog
1	1	Motion stop

Table 8-1 Functions of digital input I2 and I3

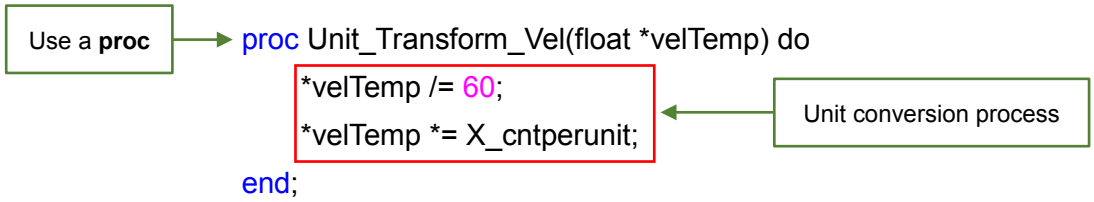
The default ratio of input voltage to reference velocity is 1V = 300 rpm. Users can change this ratio by changing the value of **JOG_VEL_SCALE**.

Application explanation

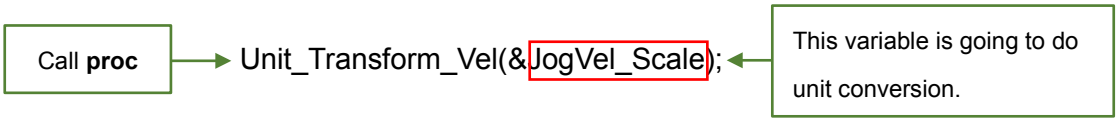
In this sample code, the analog voltage (which is input to drive from Ref+ and Ref- and can be got by the variable **Vcmd**) multiplied by the ratio (input voltage to reference velocity) equals the desired speed. Take this sample code 1V = 300 rpm as an example, voltage between 0 and 10V leads to speed between 0 and 3000 rpm.

In this sample code, it is desired to use voltage between 0 and 10V, so I2 and I3 are used to control motion direction. If users want to input analog voltage between -10 and +10V, they can directly use velocity mode. Moreover, in this sample code, the **proc** scheme of PDL is used to do unit conversion (refer to Chapter 4). Just use a **proc** in front of main program, and program the process of unit conversion in the **proc**. Then, users are able to call this **proc** for unit conversion in the main program. The **proc** can be called for many times. This reduces the complexity in the main program.

Step 1. Use a **proc** in front of main program and edit the process of unit conversion in the **proc**.



Step 2. Call **proc** when the unit conversion is needed in the main program every time.



9. Continuous motion in force / torque mode

Example number	9
File name	9.pdl
Drive model	D-series drive
Firmware requirement	D1: Lightening 0.173 MDP 0.226 and above D2: Lightening 0.173 MDP 0.027 and above
Function	Control the motor to do forward or backward motion by digital inputs in force / torque mode.
Used input	I1, I2
Used output	null

Sample code

```
// ============================ Macro ============================
#define Vol_Set      1                          //  Unit: V
// ========================= Main Program =========================
#task/1;
X_cmd_ext_i_sc = 2;                             //  Set the ratio of output current 2A to input voltage
                                                //  1V in force / torque mode
X_cmd_ext_i_sc /= 13.6;                         //  This command is set to be 3.9, 13.6, or 22.5 for
                                                //  D2-01, D2-04, or D2-10 drive respectively
X_cmd_ext_i_sc *= 1000;

_loop:
if (X_ready & ~I2) do
    till (I2);                                  //  Wait for I2 to be ON (rising-edge trigger)
    X_oper_mode1 = 3;                           //  Set the operation mode to be torque mode
    while (X_ready) do
        if (I1) do                              //  If I1 is ON
        Vcmd_offs = -Vol_Set;                   //  Do forward move
    else do                                     //  I1 is OFF
        Vcmd_offs = Vol_Set;                    //  Do backward move
        end;
    end;
else do
    sleep 200;
    Vcmd_offs = 0;                              //  Set Vcmd_offs to 0
    X_oper_mode1 = 1;                           //  Change the mode back to position mode
end;
```

```
goto _loop;
ret;
```

How to use

In this sample code, operation mode is changed to force / torque mode when users enable motor and send a rising-edge signal to input I2. At this time, motor starts running immediately. Users are able to control the motion to be forward or backward by using input I1. Table 9-1 shows the definition of digital input I1. The motor will stop moving and return to position mode when an error happens or the motor is disabled.

I1	Motion direction
0	Forward
1	Backward

Table 9-1 Definition of digital input I1

In this sample code, users are able to change output torque by changing the value of **Vol_Set** in the following command.

```
#define Vol_Set      1                  //   Unit: V
```

Here, the input voltage 1V is equal to output current 2A. By increasing or decreasing the value of **Vol_Set**, users are able to increase or decrease the output current, and thus to change the output torque. For example, setting **Vol_Set** = 2 can obtain the output current 4A if the input voltage is 2V.

Application explanation

This sample code shows how to control motion direction in force / torque mode through PDL. It can be used in the following two cases. (1) No outside voltage is available to the drive. (2) The drive (pulse-only type) cannot receive voltage command.

In this sample code, by using the built-in variable **Vcmd_offs** (bias for analog voltage input), users are able to set the value of operation torque and get the desired output current. This sample code especially sets **Vcmd_offs** to 0 when an error happens to disable the motor. This can avoid the danger from a suddenly unintended acceleration after errors are resolved and the motor is enabled again. The unit for **Vcmd_offs** is Volt (V).

The built-in variable **X_oper_mode1** is used to switch the operation mode. The definition of **X_oper_mode1** is given in Table 9-2.

Value	Operation mode
0	Stand-alone mode
1	Position mode
2	Velocity mode
3	Force / Torque mode

Table 9-2 Definition of **X_oper_mode1**

Moreover, users can set the ratio of input voltage to output current through setting the built-in variable **X_cmd_ext_i_sc**. In this sample code, the input analog voltage 1V is equal to the output torque with output current 2A.

```
X_cmd_ext_i_sc = 2;          --------(1)
X_cmd_ext_i_sc /= 13.6;      --------(2)
X_cmd_ext_i_sc *= 1000;      --------(3)
```

In (1), 2 represents that output current is 2A/V. In (2), 13.6 represents the maximum output current (**curr_drv_peak**) of D2-04 drive. For D2-01 and D2-10 drives, this value should be set to 3.9 and 22.5 respectively. In (3), 1000 is for unit conversion and should not be modified.