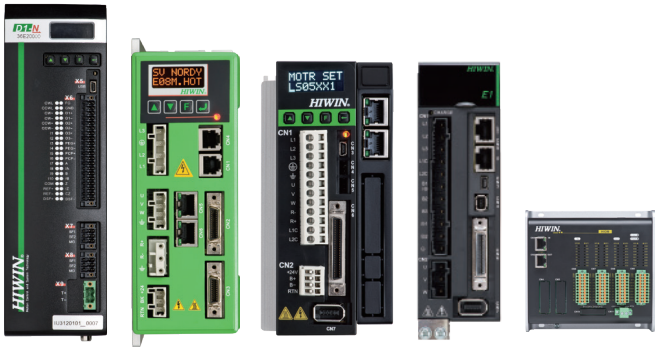
HIWIN®



MPI函式庫

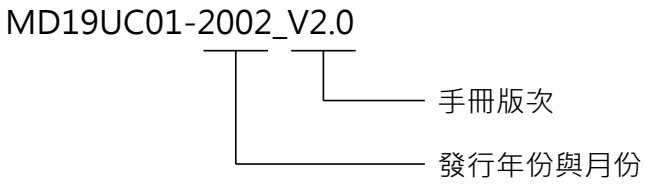
參考手冊

www.hiwinmikro.tw

MD19UC01-2002_V2.0

修訂紀錄

手冊版次資訊亦標記於手冊封面右下角。



發行日期	版次	DLL 版次	更新內容
2020/02/10	2.0	6.00	初版發行。

目錄

目錄

目錄

(此頁有意留白。)

1. 前言

1.1 關於本手冊

本手冊適用於 HIWIN D 系列驅動器與 E1 系列驅動器。MPI 函式庫，藉 DLL 檔實現，接受任何由 Microsoft Visual C++、Visual Basic .NET 或 LabVIEW 所生成的應用程式。其特色為使用 MFC，以及可在 Windows 95/98/2000/XP/7/10 Professional 上運作。透過 MPI 函式庫，使用者可完成以下項目。

- 設定通訊組態（如通訊埠編號、鮑率、RS232/USB/CAN/...）。
  註：HIWIN 台灣所製造的 D 系列驅動器不支援 CAN。
- 同時使用多個通訊埠。每個通訊埠須連接不同台驅動器，或具兩個以上 RS232 埠的同一台驅動器。
- 支援多工。多個任務可以以最小的延遲時間，透過 DLL 介面進入驅動器。
- 處理錯誤。例如：通訊錯誤發生時，MPI 函式庫會持續發送訊息，直到錯誤被清除，或次數達到 try again 變數的極限。
- 讀取 / 寫入驅動器的變數或陣列。支援 64 位元變數。
- 執行 PDL 函式。
- 設定 / 清除 / 切換 / 讀取狀態。
- 以 PDL 命令監控控制器事件。
- 支援數據收集（記錄）。
- 在兩個或多個使用 mpi 的應用程式之間建立網路連結。

MPI 函式庫由四個檔案所組成：**mpi.lib**、**mpi.dll**、**canlib32.dll** 與 **mpint.h**。至於 PDL 語言與 DCE 資料庫的相關資訊，請參閱大銀微系統官網上的《PDL Reference Manual for D-series Drives》手冊。

1.2 資料庫驗證

資料庫為驅動器裡的主程式，內含 MDP 與 PDL 檔，用來當作每個驅動器中暫存器與變數之間的參考。驅動器裡的資料庫須與 PC 裡的一致，詳細內容請參閱函式 compareFw。

使用者可利用 Wizalg 程式（Lightening 安裝包裡的工具）來驗證驅動器的資料庫。操作流程如下。

步驟1. 開啟 Wizalg 程式（路徑為 C:\HIWIN\dce\toolswin\winkmi）。

步驟2. 連線 PC 與驅動器。

步驟3. 選擇 **File** 中的 **Verify data base...**，開啟 **firmware's** 視窗。

步驟4. 點擊 **firmware's** 視窗左方 slave 列表的每一個驅動器。如圖 1.2.1 所示，將顯示一個兩欄三列的表格。欄 1 為驅動器裡的韌體，欄 2 為 PC 工作目錄裡的檔案。每個 Csum 副列裡的校對和須一致（不適用於第一個 Csum 副列 Boot Dsp Program）。

步驟5. 藍字代表校對和匹配（如圖 1.2.1 所示），紅字代表校對和不匹配（如圖 1.2.2 所示）。若校對和不匹配，請參閱 1.3 節來更新資料庫。

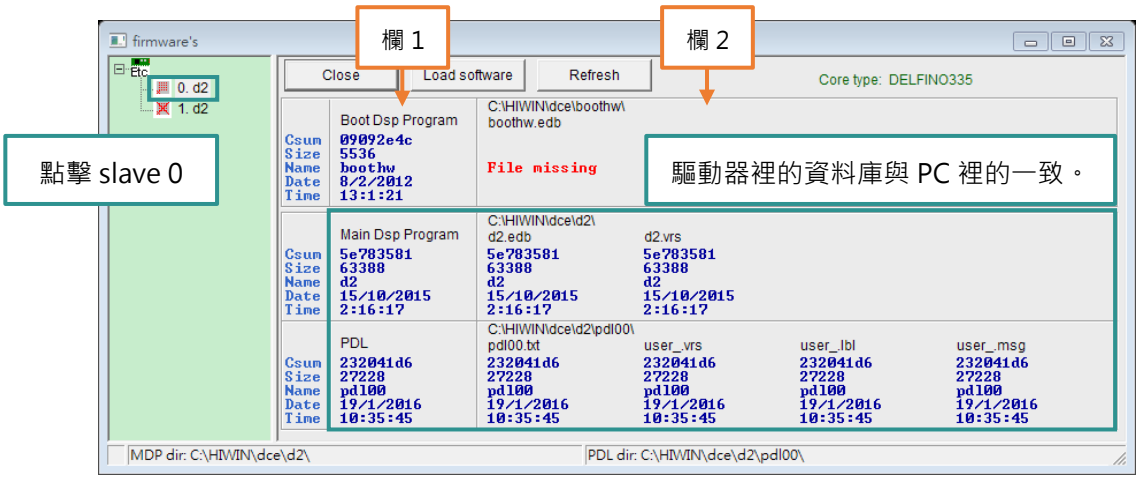


圖 1.2.1 校對和匹配

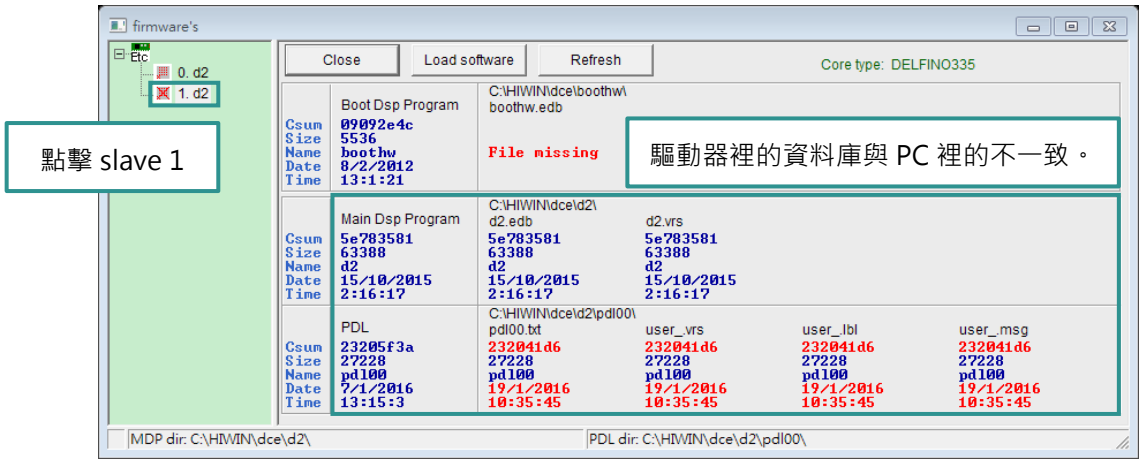


圖 1.2.2 校對和不匹配

1.3 資料庫更新

使用者可利用 Wizalg 程式來更新 MDP 與 PDL 檔。操作流程如下。

步驟1. 開啟 Wizalg 程式（路徑為 C:\HIWIN\dce\toolswin\winkmi）。

步驟2. 連線 PC 與驅動器。

步驟3. 選擇 **Boot** 中的 **Auto load programs...**，開啟 **Auto load programs** 視窗。

步驟4. 取消勾選 **Compile PDL**，再點擊 **Run** 按鈕。

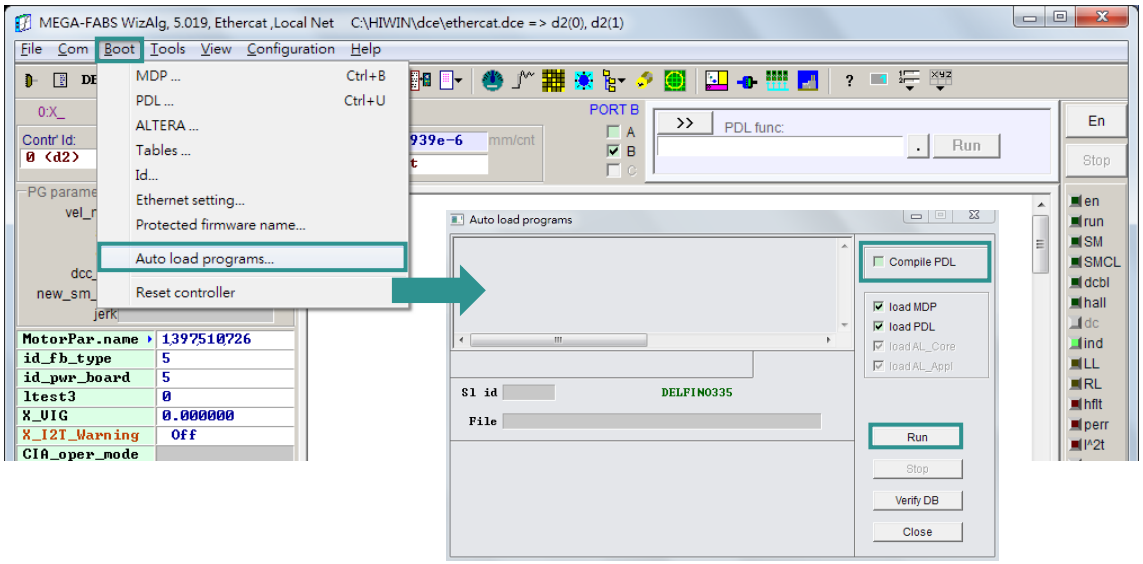


圖 1.3.1

2. 初始化

本章說明開始或結束應用程式時所需呼叫的導出函式。

2.1 openDCE

目的

建立 PC 與驅動器之間的通訊。

原型

```
MDCE *openDCE(char *pwdin, char *cnfname);
```

參數

*pwdin　　組態檔路徑。

*cnfname　　組態檔名稱。若將其設為 **NULL**，預設組態檔為 system.dce。

回傳值

指向『通訊物件』的指標，在其他函式中可被視為 *pcom*。

備註

(1) 使用者可開啟多個通訊物件。利用函式 setComPar，每個通訊物件會被設定至不同的通訊埠。每個回傳的 MDCE 指標（通常是其他函式中的最後一個參數）為通訊物件的識別符。

(2) 若只有一個通訊期，使用者可忽略回傳的 MDCE 指標，並在其他函式中設定 pcom=NULL。

(3) 呼叫函式 openDCE 後，請呼叫函式 setComPar 以定義通訊物件的參數。每個通訊物件僅能各呼叫一次函式 openDCE。

(4) 欲取得組態檔完整的路徑名稱，請呼叫函式 getDceCnfFname。

(5) 欲關閉通訊物件（約莫在結束應用程式之前），請呼叫函式 deleteDCE。

(6) 不同的驅動器須對應不同的組態檔，請參閱下表。

表 2.1.1

驅動器類型		openDCE 路徑
D2T	D2T-□□□□-S	openDCE("C:\\HIWIN\\dce", "d2.dce");
	D2T-□□□□-F	
	D2T-□□□□-E	openDCE("C:\\HIWIN\\dce", "D2COE.dce");
D1	D1-□□-S	openDCE("C:\\HIWIN\\dce", "tamuz.dce");
	D1-□□-F	
	D1-□□-E	openDCE("C:\\HIWIN\\dce", "D1COE.dce");

D1-N	D1-N-□□-S	openDCE("C:\\HIWIN\\dce", "D1N.dce");
	D1-N-□□-F	
	D1-N-□□-M	
	D1-N-□□-E	openDCE("C:\\HIWIN\\dce", "D1NCOE.dce");
E1	ED1S	openDCE("C:\\HIWIN\\dce", "D3.dce");
	ED1F	openDCE("C:\\HIWIN\\dce", "D3COE.dce");

2.2 deleteDCE

目的

結束 PC 與驅動器之間的通訊。

原型

```
void deleteDCE(MDCE *pcom=NULL, int closeMFC=FALSE);
```

參數

pcom	指向『通訊物件』的指標。
closeMFC	欲清除 MFC 內部執行緒（使用者於應用程式開始時呼叫函式 setMFCmode 才有效），請將此參數設為 **TRUE**。大部分的情況下，此參數應為 **FALSE**，因關閉 DLL 時 MFC 內部執行緒會自動被清除。

回傳值

無

備註

不再需要通訊時（約莫在結束應用程式之前），請呼叫此函式。

2.3 ver

目的

取得 DLL 版本。

原型

```
char *ver(int print);
```

參數

print 欲在訊息框中印出 DLL 版本，請將此參數設為**非零值**。

回傳值

指向『DLL 版本』的指標。

備註

DLL 會不定期更新，故建議在應用程式中使用此函式，以定期取得 DLL 版本。

2.4 disErrMsgBox

目的

關閉通訊錯誤提示訊息框。(偵測到版本衝突時，無法關閉版本比較視窗。)

原型

```
void disErrMsgBox(int dis);
```

參數

dis	欲關閉通訊錯誤提示訊息框，請將此參數設為 **1**。 欲開啟訊息框，則設為 **0**。

回傳值

無

備註

(1) 可隨時呼叫此函式來關閉 / 開啟訊息框。

(2) 第一次做 PC 與驅動器之間的資料庫確認時，請在呼叫函式 setComPar 之前呼叫此函式，以關閉訊息框。

(3) 若偵測到版本衝突，會彈出版本比較視窗。在這個情況下，此函式無法關閉此視窗。

(4) 一旦使用此函式，所有從站皆適用。

2.5 disAllErrMsgBox

目的

關閉通訊錯誤提示訊息框。(偵測到版本衝突時，可以關閉版本比較視窗。)

原型

```
void disAllErrMsgBox(int dis);
```

參數

dis	欲關閉通訊錯誤提示訊息框，請將此參數設為 **1**。 欲開啟訊息框，則設為 **0**。

回傳值

無

備註

(1) 可隨時呼叫此函式來關閉 / 開啟訊息框。

(2) 第一次做 PC 與驅動器之間的資料庫確認時，請在呼叫函式 setComPar 之前呼叫此函式，以關閉訊息框。

(3) 若偵測到版本衝突，會彈出版本比較視窗。與函式 disErrMsgBox 不同，此函式可以關閉此視窗。

(4) 一旦使用此函式，所有從站皆適用。

2.6 getFwErr

目的

確認 PC 與驅動器韌體之間是否有版本衝突。

原型

```
int getFwErr(MDCE *pcom=NULL);
```

參數

pcom 指向『通訊物件』的指標。

回傳值

偵測到 PC 與驅動器韌體之間的版本衝突時，將回傳**非零值**。

備註

進行版本比較的時機：『第一次成功建立通訊時』或『呼叫函式 compareFw 時』。

2.7 getVerErrCodeSl

目的

取得詳細的版本錯誤代碼。

原型

```
unsigned int getVerErrCodeSl(int slave=0, MDCE *pcom=NULL);
```

參數

slave 從站編號。

pcom 指向『通訊物件』的指標。

回傳值

詳細的版本錯誤代碼。

每兩位元代表各檔案資料庫的錯誤狀態。

0 – 無錯誤。

1 – 無法進行版本比較。

2 – 存在版本衝突。

位元與檔案的關係如下表所示。

位元	檔案
0、1	BOOT 韌體檔 *.edb
4、5	MDP 韌體檔 <組態檔名稱>.edb
6、7	MDP 變數列表檔 <組態檔名稱>.vrs
8、9	PDL 韌體檔 <名稱>.edb
10、11	PDL MDP 變數列表檔 <user_$>.vrs
12、13	PDL MDP 函式列表檔 <user_$>.lbl
14、15	PDL MDP 訊息列表檔 <user_$>.msg

註：不同的驅動器須對應不同的組態檔，請參閱表 2.1.1。以驅動器型號 D2T-□□□□-S 為例，其組態檔名稱為 d2。

2.8 compareFw

目的

進行 PC 與驅動器韌體之間的版本比較。

原型

```
int comepareFw(int show=0, MDCE *pcom=NULL);
```

參數

show	欲開啟版本比較視窗（即便所有韌體皆匹配），請將此參數設為 **1**。
pcom	指向『通訊物件』的指標。

回傳值

無

備註

(1) 若偵測到版本衝突，會彈出版本比較視窗（除非使用者已事先呼叫函式 disErrMsgBox）。呼叫函式 getFwErr 後，會回傳**非零值**。

(2) 進行版本比較的時機：第一次成功建立通訊時。

2.9 setMFCmode

目的

呼叫不在主執行緒中的函式 showcomstatus 與 OpenRecord。

原型

```
void setMFCmode();
```

參數

無

回傳值

無

備註

(1) 呼叫其他函式前，請先呼叫此函式。此函式僅須被呼叫一次。

(2) 若函式 showcomstatus 與 OpenRecord 在主執行緒中，使用者不須呼叫此函式。

2.10 resetController

目的

重置驅動器。

原型

```
void resetController(int slave=0, MDCE *pcom=NULL);
```

參數

slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

無

備註

呼叫此函式就如同重置硬體，先至開機模式，再回到正常模式。

3. 通訊

本章說明所有與通訊設定相關的函式。

3.1 setComPar

目的

設定通訊參數。

原型

```
int setComPar(int port, int baudrate, int mode, int trid, int rcid,
              int canbaudrate, int msgStand, int canpipelevel, int timeout,
              int locktime, int iternum, MDCE *pcom=NULL);
```

參數

port　　通訊埠編號。

baudrate　　在 RS232（USB 接頭）模式下的鮑率代碼。

代碼	0	1	2	3	4	5	6
鮑率	1200	2400	4800	9600	19200	38400	56000
代碼	7	8	9	11	12	13	
鮑率	57600	115200	128000	230400	460800	921600	

mode　　通訊模式代碼。

代碼	1	4	5
模式	RS232 （USB 接頭）	EtherCAT mega-ulink	USB （僅適用於 E1 系列驅動器）

trid　　保留。設為 0。

rcid　　保留。設為 0。

canbaudrate　　保留。設為 0。

msgStand　　保留。設為 0。

canpipelevel　　保留。設為 0。

timeout　　等待驅動器回傳訊息的時間（建議設為 100）。　　單位：毫秒

locktime　　容許通訊錯誤存在的時間（建議設為 200）。　　單位：毫秒

iternum　　重覆傳送訊息給驅動器的最大次數（建議設為 6）。

pcom　　指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

2 – 通訊埠不存在。

5 – 其他應用程式正在使用此通訊埠。

備註

呼叫函式 openDCE 後，再呼叫此函式。之後，即可隨時呼叫此函式以變更通訊設定。

3.1.1 連線至 EtherCAT mega-ulink

欲連線至 EtherCAT mega-ulink，請呼叫函式 setComPar，並設定 mode = 4。

範例

```
setMFCmode();
MDCE *pcom=openDCE("c:\\HIWIN\dce\\", NULL);
setComPar(0, 0, 4, 0, 0, 0, 0, 0, 50, 0, 8, pcom);
```

備註

(1) 呼叫函式 openDCE 前，須先呼叫函式 setMFCmode。

(2) 函式 setComPar 中，所有非相關的參數皆為 0 且第三個參數 mode 為 4。

(3) 手動選擇用於 PC 的網路裝置，如下圖所示。按下 **SetUp...**按鈕以開啟 wizalg Communication Setup 視窗，再按下 **EtherCat...**按鈕來選擇欲使用的網路。接著，按下 **Apply** 按鈕來儲存此設定。這樣一來，每次連線至 EtherCAT mega-ulink 時，皆可恢復此設定。

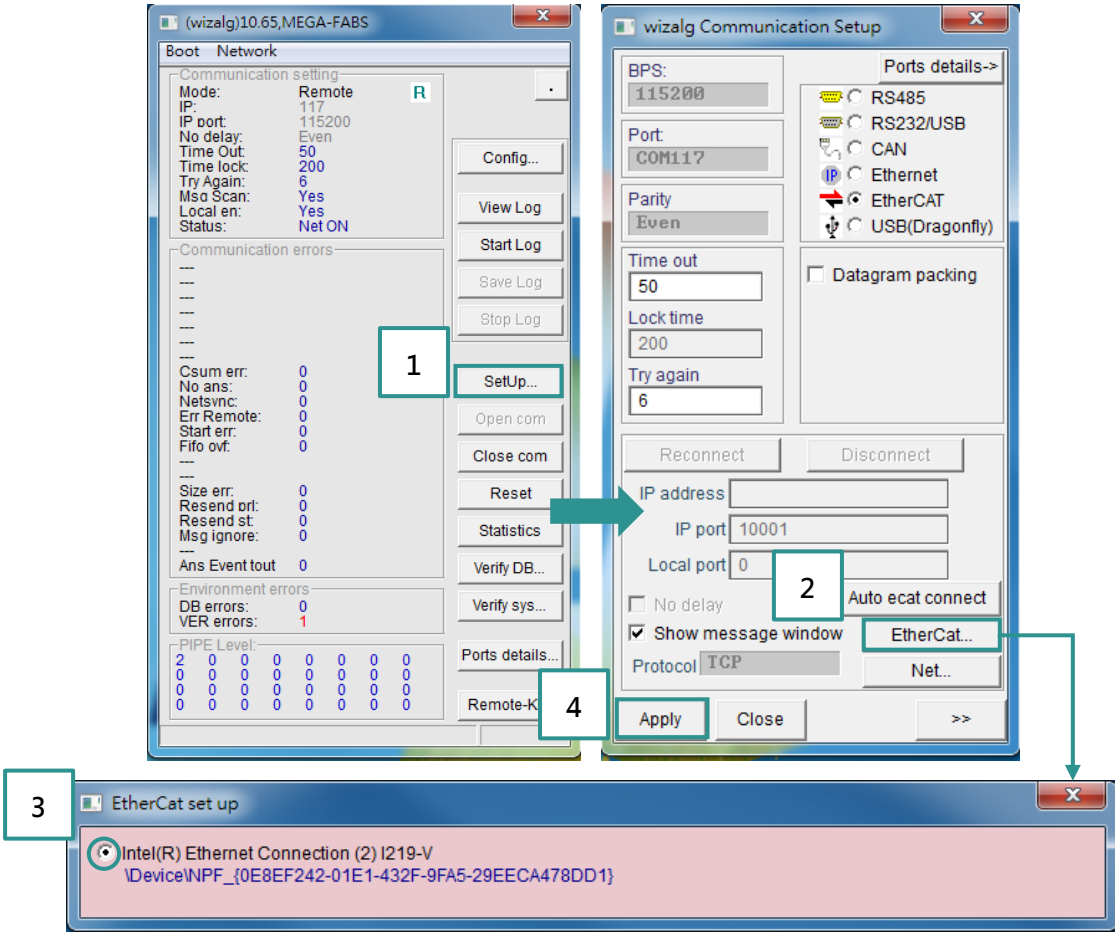

(wizalg)10.65,MEGA-FABS
Boot Network
Communication setting
Mode: Remote
IP: 117
IP port: 115200
No delay: Even
Time Out: 50
Time lock: 200
Try Again: 6
Msg Scan: Yes
Local en: Yes
Status: Net ON
Communication errors
Csum err: 0
No ans: 0
Netsync: 0
Err Remote: 0
Start err: 0
Fifo ovf: 0
Size err: 0
Resend prl: 0
Resend st: 0
Msg ignore: 0
Ans Event tout 0
Environment errors
DB errors: 0
VER errors: 1
PIPE Level:
Config...
View Log
Start Log
Save Log
Stop Log
SetUp...
Open com
Close com
Reset
Statistics
Verify DB...
Verify sys...
Ports details...
Remote-K
1
wizalg Communication Setup
BPS: 115200
Port: COM117
Parity Even
Time out 50
Lock time 200
Try again 6
Ports details->
RS485
RS232/USB
CAN
Ethernet
EtherCAT
USB(Dragonfly)
Datagram packing
Reconnect
Disconnect
IP address
IP port 10001
Local port 0
No delay
Show message window
Protocol TCP
Auto ecat connect
EtherCat...
Net...
Apply
Close
>>
2
4
3
EtherCat set up
Intel(R) Ethernet Connection (2) I219-V
\Device\NPF_{0E8EF242-01E1-432F-9FA5-29EECA478DD1}

圖 3.1.1.1

3.1.2 連線至 USB

欲連線至 USB，請呼叫函式 setComPar，並設定 mode = 5（僅適用於 E1 系列驅動器）。

範例：連線至兩台 USB 裝置

```
setMFCmode();
int port0 = 0, port1 = 1;
//  打開相對應通訊埠的通訊物件
MDCE *pcom_0 = openDCE("c:\\HIWIN\dce\\", NULL);
MDCE *pcom_1 = openDCE("c:\\HIWIN\dce\\", NULL);

setComPar(port0, 0, 5, 0, 0, 0, 0, 0, 50, 0, 8, pcom_0);  //  連線至 port0
setComPar(port1, 0, 5, 0, 0, 0, 0, 0, 50, 0, 8, pcom_1);  //  連線至 port1
```

備註

須先安裝 USB 驅動程式。

3.2 getComPar

目的

取得函式 setComPar 所設定的參數。

原型

```
int getComPar(int *pport, int *pbaudrate, int *pmode, int *ptrid, int *prcid,
              int *pcanbaudrate, int *pmsgStand, int *pcanpipelevel,
              int *ptimeout, int *plocktime, int *piternum, MDCE *pcom=NULL);
```

參數

這些引數為指標，與函式 setComPar 中的參數相同。

回傳值

0 – 函式執行成功。

-1 – 無通訊物件。

3.3 getDceCnfFname

目的

取得組態檔完整的路徑名稱。

原型

```
int getDceCnfFname(char *str, MDCE *pcom=NULL);
```

參數

*str	組態檔完整的路徑名稱。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

-1 – 無通訊物件。

備註

**str* 與 *pcom* 須依據回傳字串的長度，指向具有足夠大小的字串記憶體（建議設為 200）。

3.4 getSlaveFname

目的

取得從站名稱。

原型

```
int getSlavefFname(char *str, int slave=0, MDCE *pcom=NULL);
```

參數

*str	從站名稱。
slave	從站編號。若只有一台驅動器，則為 0。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

-1 – 無通訊物件。

3.5 getUsbHubPort

目的

取得 USB 裝置名稱。

原型

```
int getUsbHubPort(int port, char *hubport, int strlen, MDCE *pcom=NULL);
```

參數

port	通訊埠編號。 若只有一台 USB 裝置，設其為 0。
*hubport	USB 裝置名稱。
strlen	USB 裝置名稱的字串長度。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr (將 *str* 設為" ") 以取得錯誤的描述。

備註

(1) 此函式僅適用於 E1 系列驅動器。

(2) 呼叫函式 openDCE 後，再呼叫此函式。

(3) 可利用此函式得到所有連接到電腦的驅動器。

範例

```
#define MaxSlaveNum 32
MDCE *pCom[MaxSlaveNum];  //  最大連線軸數為 32 軸
int port=0;
char hubport[200];
int errcode;

//  掃描 USB 裝置
for (int i=0;i<MaxSlaveNum;i++)
{
```

```
    pCom[i]=openDCE("c:\\HIWIN\dce\\", NULL);  //  打開通訊物件
    if ((errcode=getUsbHubPort(port, hubport, sizeof(hubport), pCom[port]))==0)
    {
        printf("Device %d: %s\n", port, hubport);
        port++;
    }
    else
    {
        //  印出錯誤訊息
        char errstr[200];
        GetErrorStr(errcode, " ", 0, errstr, pCom[port]);
        printf("Device %d: %s\n", port, errstr);
        break;
    }
}
```

3.6 closePort

目的

關閉通訊埠。

原型

```
void closePort(MDCE *pcom=NULL);
```

參數

pcom 指向『通訊物件』的指標。

回傳值

無

備註

此函式僅關閉通訊埠，而不是像函式 deleteDCE 一樣刪除通訊物件。

3.7 openPort

目的

再次開啟最後一個 setComPar 所指定的通訊埠。

原型

```
int openPort(MDCE *pcom=NULL);
```

參數

pcom　　指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

2 – 通訊埠不存在。

5 – 其他應用程式正在使用此通訊埠。

備註

(1) 使用者可呼叫此函式以替代函式 setComPar。在這個情況下，將應用最後設定的通訊設定。

(2) 使用者可利用通訊狀態對話框（參閱函式 showcomstatus）來改變通訊設定。

3.8 showcomstatus

目的

開啟對話框，顯示通訊設定與相關數據的資訊。使用者可在此對話框內執行多項任務，如：關閉 / 開啟通訊埠、開啟開機對話框（PDL 與 MDP）與取得網路。

原型

```
void showcomstatus(MDCE *pcom=NULL);
```

參數

pcom 指向『通訊物件』的指標。

回傳值

無

備註

執行應用程式時，可開著或最小化此對話框。

3.9 loadDceSw

目的

更新與驅動器相關的所有軟體，如 MDP 和 PDL。

原型

```
int loadDceSw(int pdlComp, MDCE *pcom=NULL);
```

參數

pdlComp	欲編譯 PDL，請將此參數設為**非零值**。否則，請設為 **0**。
pcom	指向『通訊物件』的指標。

回傳值

若函式執行成功，將回傳 **int** 型態的值 **0**。否則，將回傳載入過程中錯誤發生的次數。

3.10 updateDataBase

目的

更新 DLL 內部資料庫。

原型

```
void updateDataBase(MDCE *pcom);
```

參數

pcom 指向『通訊物件』的指標。

回傳值

無

備註

欲刷新 DLL 內部變數 / 函式列表，不須關閉再開啟應用程式，呼叫此函式即可。

(此頁有意留白。)

4. 事件接收

4.1 waitOnMsgP

目的

接收在驅動器上運行的 PDL 程式所發送的 PDL `printl` 命令。PDL `printl` 命令的語法如下所示。

```
printl/mode1/mode2("…String…", var1, …varN);
```

此語法中，mode2 為選配 (若沒特別註明即為 0)，mode1 代表字串或其他屬性的顏色，*var1* …*varN* 則為選配變數。詳細內容請參閱手冊《PDL Reference Manual for D-series Drives》。

原型

```
int waitOnMsgP(int *pcode, char *pmsg=NULL, int *pcolor=NULL, int *pparam=NULL,
               int *pnumovrflw=NULL, int timeout=INFINITE, MDCE *pcom=NULL);
```

參數

*pcode	mode2 的值。
*pmsg	`printl` 命令中的字串；其大小應至少為 20 字元。
*pcolor	將 mode1 所指定的顏色代碼轉換成 **COLORREF** 裡相對應的值。
*pparam	一陣列。陣列中的第一項為變數的數量，其餘項目分別顯示每個變數的 32 位元值。若變數為 float 型態，須將其轉態成浮點數，而非整數。 註：字串 (*pmsg*) 裡的變數型態須與 `printl` 命令裡的變數型態相符。
*pnumovrflw	漏掉的 `printl` 訊息數量。若沒有執行緒聽取函式 waitOnMsgP，但出現了很多 `printl` 訊息，就有可能會發生溢位。函式 waitOnMsgP 又被聽取時，溢位計數器即歸零。 註：若發生溢位，收到訊息裡的屬性 (*pcode*、*pmsg*、*pcolor*、*pparam*) 皆有效。
timeout	取得 `printl` 訊息的時間。若 **timeout = -1** (INFINITE)，函式不會回傳任何值，直到 `printl` 訊息被送出 (或函式 releaseWaitMessage 被其他執行緒呼叫)。若 **timeout = 0** 且有有效訊息，函式會立刻回傳 0；若 **timeout = 0** 卻無有效訊息，函式會回傳-2。若 **timeout = 其他正數(timeout > 0)**，函式會在所設定的時間被聽取 (單位 : 微秒)。此情況下，沒訊息或呼叫函式 releaseWaitMessage，函式會回傳 5。
pcom	指向『通訊物件』的指標。

回傳值

0 – ok。

-1 – 無通訊物件。

-2 – 無有效訊息 (函式 releaseWaitMessage 被呼叫或 **timeout = 0**)，或超過一個執行緒呼叫此函式。

5 – 超時。

備註

(1) 欲忽略指標的內容，請將其設為 NULL。例如，若使用者不在意 mode2 的值，設*pcode = NULL。

(2) 此函式一次僅能被一個執行緒呼叫，否則會回傳-2，也有可能會漏掉訊息。

(3) 執行緒無法快速處理來自驅動器的訊息時，就有可能會發生溢位。mpi.dll 的佇位大小為 200。

範例一

PDL 程式執行以下 printl 命令（假設 *x_enc_pos* = 2000、*x_p_p_g* = 0.01）。

```
printl/103/00000044("x axis: enc_pos=%ld, pos loop gain=%g", x_enc_pos, x_p_p_g);
```

負責聽取函式 waitOnMsgP 的執行緒會回傳 0（代表 ok），參數如下所示。

```
*pcode=0x44;
pmsg="x axis: enc_pos=2000, pos loop gain=0.01";
*pcolor=0x02ff0000;  //  mode1 的 103 轉換成 COLORREF 的 0x02ff0000：藍色
pparm[0]=2; pparm[1]=2000; *((float *)&pparam[2])=0.01;
*pnumovrflw=0  //  假設沒溢位
```

範例二

在執行緒裡的迴圈執行函式 waitOnMsgP（變數 *endMsgTread* 設好時）。

```
int endMsgTread=0;
Uint msgThrd(LPVOID pPar)
{
    MDCE *pcom=(MDCE*)pPar;  //  指向『通訊物件』的指標
    int st, code, c, numOvrFlow, param[10];
    char msg[200], str[200];
    while(!endMsgTread) {
        st = waitOnMsgP(&code, msg, &c, param, &numOvrFlow, INFINITE, pcom);
    switch(st) {
        case 0:  //  ok
            if(numOvrFlow!=0){
                sprintf(str, "####### %ld Message lost", numOvrFlow);
            }
            strcpy(str,msg);
            break;
```

```
        case -1:
            sprintf(str, "####### No communication object");
            break;
        case -2:
            break;  //  什麼都不做
        case 5:
            sprintf(str, "####### Time out");
            break;
        default:  //  其他的回傳值
            sprintf(str, "Message return error %ld", st);
            break;
        }
    }
return(0);
}
```

4.2 releaseWaitMessage

目的

釋放負責聽取函式 waitOnMsgP 的執行緒。

原型

```
void releaseWaitMessage(MDCE *pcom);
```

參數

pcom 指向『通訊物件』的指標。

回傳值

無

備註

(1) 結束應用程式或呼叫函式 deleteDCE 之前，請呼叫此函式。

(2) 若函式 waitOnMsgP 中的 **timeout = 0**，則不須呼叫此函式。

範例

欲應用此函式，使用者可在 4.1 節範例二的 `Uint msgThrd` 中加入以下敘述。

```
endMsgTread = 1;  //  離開 while 迴圈
releaseWaitMessage(pcom);  //  釋放負責聽取函式 waitOnMsgP 的執行緒
……
```

4.3 insertEvent

目的

利用事件 ID，從眾多 PDL `printl` 命令中，篩選出欲監控的特定訊息。

原型

```
int insertEvent(int id, HANDLE h, MDCE *pcom=NULL);
```

參數

id	事件 ID；範圍：0 ~ 199。 其值須與 PDL `printl` 命令中 var1 的值一致（請參閱函式 waitOnMsgP），所以建議將事件 ID 與 *var1* 的值皆設為從站編號。
h	事件處置器；利用 Windows（32 位元）中的函式 CreateEvent 建造而成。 若將其設為 **NULL**，事件會被清除。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

-1 – 尚未呼叫函式 openDCE。

-2 – 事件 ID 超出範圍。

備註

呼叫此函式之前，請先呼叫函式 setEventCode。

4.4 closeEvent

目的

關閉事件處置器。

原型

```
int closeEvent(int id, MDCE *pcom);
```

參數

id	事件 ID；範圍：0 ~ 199。 其值須與 PDL `printl` 命令中 *var1* 的值一致（請參閱函式 waitOnMsgP），所以建議將事件 ID 與 *var1* 的值皆設為從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

-1 – 尚未呼叫函式 openDCE。

-2 – 事件 ID 超出範圍。

-3 – 函式執行失敗。

4.5 getEvent

目的

取得事件處置器。

原型

```
HANDLE getEvent(int id, MDCE *pcom);
```

參數

id	事件 ID；範圍：0 ~ 199。 其值須與 PDL `printl` 命令中 *var1* 的值一致（請參閱函式 waitOnMsgP），所以建議將事件 ID 與 *var1* 的值皆設為從站編號。
pcom	指向『通訊物件』的指標。

回傳值

事件處置器。

-1 – 尚未呼叫函式 openDCE。

-2 – 事件 ID 超出範圍。

4.6 setEventCode

目的

設定事件代碼。

此事件代碼可被視為一過篩器，從眾多 PDL `printl` 命令中，篩選出欲監控的特定訊息。

原型

```
int setEventCode(int code, MDCE *pcom);
```

參數

code	PDL `printl` 命令中 mode2 的值（請參閱函式 waitOnMsgP）。 預設值為 **1**。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

-1 – 尚未呼叫函式 openDCE。

備註

不能動態變更 PDL `printl` 命令中 mode2 的值，因為它是恆定的且已在 PDL 編譯時被定義。

4.7 getLastEventData

目的

取得最後呼叫的事件代碼的的字串與變數。

原型

```
int getLastEventData(int id, char *pmsg, int *pparam, MDCE *pcom=NULL);
```

參數

id	事件 ID；範圍：0 ~ 199。 其值須與 PDL `printl` 命令中 *var1* 的值一致（請參閱函式 waitOnMsgP），所以建議將事件 ID 與 *var1* 的值皆設為從站編號。
*pmsg	`printl` 命令中的字串；其大小應至少為 20 字元。
*pparam	一陣列。陣列中的第一項為變數的數量，其餘項目分別顯示每個變數的 32 位元值。若變數為 float 型態，須將其轉態成浮點數，而非整數。 註：字串（*pmsg*）裡的變數型態須與 `printl` 命令裡的變數型態相符。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

-1 – 尚未呼叫函式 openDCE。

-2 – 事件 ID 超出範圍。

備註

欲忽略指標（**pmsg*、**pparam*）的內容，請將其設為 NULL。

範例

```
char reportStr[200];
int code=0x00000055;
int id=1;
HANDLE hevent=CreateEvent(NULL, FALSE, FALSE, NULL);
int timeOut=10000;
setEventCode(code, pcom);
```

```
int error=insertEvent(id, hevent, pcom);
if(error){
    sprintf(reportStr, "insertEvent error = %ld", error);
    printout(reportStr);
}
else {
    error=WaitForSingleObject(hevent, timeOut);
    if(error==WAIT_TIMEOUT){
    sprintf(reportStr, "Time out");
    printout(reportStr);
    }
    else {
        int param[10];
        char message[200];
        error=getLastEventData(id, message,  param, pcom);
        if(error) {
        sprintf(reportStr, "get Data error = %ld", error);
        printout(reportStr);
        }
        else {
            sprintf(reportStr, "arrived message of id=%ld", id);
            printout(reportStr);
            printout(message);  //  印出 PDL printl 命令的訊息
            int n;
            int numPar=min(param[0],8);
            for(n=0;n<numPar;n++) {  //  印出參數
                sprintf(reportStr, "%ld %ld", n+1, param[n+1]);
                printout(reportStr);
            }
        }
    }
}
closeEvent(id, pcom);  //  關閉並清除事件
```

此範例中，PDL 程式須執行 printl 命令以觸發 mode2 的值為 0x55 且從站編號 (事件 ID) 為 1 的事件。

```
#long Z_id, X_id, Y_id;
Z_id=0; X_id=1; Y_id=2;
……
printl/103/00000055("axis id=%ld; x axis: enc_pos=%ld, velocity=%g",
X_id, x_enc_pos, x_vel_max);
```

5. 數據收集（記錄）

5.1 StartRecordData

目的

開始記錄。

原型

```
int StartRecordData(char *p[], int numVar, int rate, int numSamples,
                    double *pdata, char *errstr, MDCE *pcom=NULL);
```

參數

*p[]	一陣列，含一個或多個變數名稱；其大小應至少為 *numVar* 的值。
numVar	欲記錄的變數數量。
rate	定義採樣率為 32000/rate Hz（假設驅動器的採樣率為 32000）。 因此，每 rate*0.00003125 秒，變數就會被採樣一次。
numSamples	每個變數欲記錄的最大樣本數。
*pdata	一陣列，含變數的數據。若為多個變數，記錄完變數 1 的樣本數後，才會記錄變數 2 的樣本數。因此，**pdata[0]**為變數 1 的第一個數據，**pdata[numSamples]**為變數 2 的第一個數據，**pdata[n*numSamples]**為變數 n 的第一個數據（n = 0 … numVar-1）。 故其大小應至少為 *numVar* numSamples*。
*errstr	錯誤的描述。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請從 ***errstr*** 取得錯誤的描述。

備註

(1) 此函式會立刻回傳回傳值。若回傳 0（函式執行成功），僅代表成功開始記錄，不代表數據收集完成。此時呼叫函式 GetRecdordStatus，即可知道記錄過程是否結束、收集到多少樣本。

(2) 最終記錄到的樣本數或許會比 *numSamples* 所定義的來得少。*rate* 太小時，就有可能會發生這種情況。（因與通訊速度相比，實際記錄的頻率太快了。）即便如此，成功收集到的數據仍為有效值。

(3) 欲持續記錄（直到 numSamples），建議設 ***rate*** >= 8****numVar***。

(4) 欲停止記錄，請呼叫函式 StopRecord。

(5) 欲開啟記錄視窗，請呼叫函式 OpenRecord。使用者可查看函式 StartRecordData 中設的所有參數以及其他即時的記錄資訊。

5.2 StartRecordDataN

目的

開始記錄。

原型

```
int StartRecordDataN(char *varnames, int rate, int numSamples,
                     double *pdata, char *errstr, MDCE *pcom=NULL);
```

參數

*varnames	一識別字，含一個或多個變數名稱。 請以『空白』或『逗號』區隔變數名稱。
rate	定義採樣率為 32000/rate Hz（假設驅動器的採樣率為 32000）。 因此，每 rate*0.00003125 秒，變數就會被採樣一次。
numSamples	每個變數欲記錄的最大樣本數。
*pdata	一陣列，含變數的數據。若為多個變數，記錄完變數 1 的樣本數後，才會記錄變數 2 的樣本數。因此，**pdata[0]**為變數 1 的第一個數據，**pdata[numSamples]**為變數 2 的第一個數據，**pdata[n*numSamples]**為變數 n 的第一個數據（n = 0 … numVar-1）。 故其大小應至少為 *numVar* numSamples*。
*errstr	錯誤的描述。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請從 **errstr* 取得錯誤的描述。

備註

此函式相似於函式 StartRecordData，差別在於刪除了 *numVar*，並以 **varnames* 取代 **p[]*。

範例

```
StartRecordDataN("X_ref_pos, X_pcmd_err", 16, 5000, pdata, errstr);
```

5.3 StartRecordFileN

目的

開始記錄，並將結果存在特定的檔案裡。

原型

```
int StartRecordFileN(char *varnames, int rate, int numSamples, char *filename,
                     char *errstr, int append, MDCE *pcom=NULL);
```

參數

*varnames	一識別字，含一個或多個變數名稱。 請以『空白』或『逗號』區隔變數名稱。
rate	定義採樣率為 32000/rate Hz（假設驅動器的採樣率為 32000）。 因此，每 rate*0.00003125 秒，變數就會被採樣一次。
numSamples	每個變數欲記錄的最大樣本數。
*filename	檔案名稱。
*errstr	錯誤的描述。
append	欲將結果存在舊檔裡，設其為 **1**；欲將結果存在新檔裡，設其為 **0**。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請從 **errstr* 取得錯誤的描述。

備註

(1) 此函式相似於函式 StartRecordDataN，差別在於結果是存在特定的檔案裡，而不是存在陣列中。

(2) 結果也會被存到 gpp 檔中。
因此，下面範例中的結果會被存到"C:\\myresult.txt"與"C:\\myresult.gpp"裡。

(3) 使用者可利用 wingraph 工具查看 gpp 檔。欲從命令列操作，請輸入**>wingraph myresult.gpp**。

範例

```
StartRecordFileN("X_ref_pos, X_pcmd_err", 16, 5000, "C:\\myresult.txt",
                     pdata, errstr, 0);
```

5.4 GetRecdordStatus

目的

查看記錄過程是否結束、收集到多少樣本。

原型

```
int GetRecdordStatus(int *pnumSampColect, MDCE *pcom);
```

參數

*pnumSamColect	已收集到的樣本數。
pcom	指向『通訊物件』的指標。

回傳值

0 – 記錄過程結束。

2 – 進行中。

3 – 檔案儲存失敗。

-1 – 尚未呼叫函式 StartRecordData。

備註

使用者可在迴圈中呼叫此函式，並印出 **pnumSamColect* 以查看數據收集的過程。

5.5 StopRecord

目的

停止記錄。

原型

```
void StopRecord(MDCE *pcom=NULL);
```

參數

pcom　　指向『通訊物件』的指標。

回傳值

無

備註

(1) 若記錄過程早已結束，呼叫此函式時不會發生任何事情。

(2) 欲在呼叫此函式後開始新的記錄過程，請先呼叫函式 GetRecdordStatus，確定前一個記錄過程已確實完成。

範例

```
s=StartRecordFileN(str, rate, numsampreq, "C:\\file1.txt", errstr, 0, pcom);

Sleep(1000);
StopRecord(pcom);
int numSamp, cnt=0;
int busy=GetRecdordedStatus(&numSamp, pcom);

while(busy==2 && cnt++<2000) {
    busy=GetRecdordedStatus(&numSamp, pcom);
    Sleep(10);
}
if(busy==3) {
    //  檔案儲存失敗
}
else if(cnt>=2000) {
    //  超時
```

```
}
s=StartRecordFileN(str, rate, numsampreq, "C:\\file2.txt", errstr, 0, pcom);
```

5.6 OpenRecord

目的

開啟記錄視窗。

原型

```
void OpenRecord(int show=1, MDCE *pcom=NULL);
```

參數

show	設為 **1** 以開啟記錄視窗。
pcom	指向『通訊物件』的指標。

回傳值

無

備註

(1) 使用者可在此視窗中定義一個或多個欲記錄的變數、樣本數與速率。設定完成後，點擊 **Start(F5)**按鈕開始記錄，再點擊 **Graph** 按鈕查看結果，如下圖所示。

(2) 若呼叫函式 StartRecordData，此視窗中的變數會被更新為函式 StartRecordData 提供的變數。

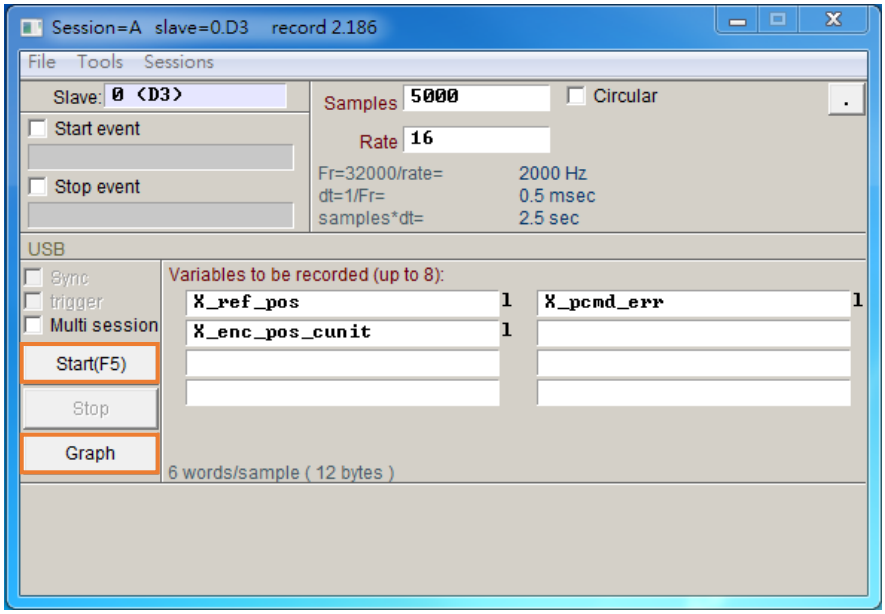


圖 5.6.1

6. 取得變數 / 陣列

當函式執行成功時，本章節中大部分的函式都會回傳 **int** 型態的值 **0**。若回傳**非零值**，請呼叫函式 GetErrorStr 以取得錯誤的描述。

所有函式皆可視變數名稱為參數，且可利用常數索引變數（以取得陣列中的其中一個項目）。
例如：`SetVarN("rec_buf[300]", 15, 0, NULL);`

6.1 GetVarAddr

目的

檢查變數的存在與大小。

原型

```
int GetVarAddr(char *varName, int slave, int *psize, MDCE *pcom=NULL);
```

參數

*varName	變數名稱。
slave	從站編號。
*psize	變數大小。 若為非陣列變數，回傳 **1**；若為陣列變數，回傳**陣列大小**。 若找不到此變數，則回傳**-1**。
pcom	指向『通訊物件』的指標。

回傳值

變數的位址。
若變數不存在，則回傳 **ADDERR**（-100）。

備註

欲取得變數的型態，請呼叫函式 GetVarAddrType。

6.2 GetVarAddrType

目的

檢查變數的存在、大小與型態。

原型

```
int GetVarAddrType(char *varName, int slave, int *psize,
                   int *ptype, MDCE *pcom=NULL);
```

參數

*varName	變數名稱。
slave	從站編號。
*psize	變數大小。 若為非陣列變數，回傳 **1**；若為陣列變數，回傳**陣列大小**。 若找不到此變數，則回傳**-1**。
*ptype	變數型態。 1：short（16 位元整數）　4：pointer（32 位元） 2：long（32 位元整數）　8：64 位元變數 3：float（32 位元浮點數）　0：找不到此變數。
pcom	指向『通訊物件』的指標。

回傳值

變數的位址。

若變數不存在，則回傳 **ADDERR**（-100）。

6.3 SetVarN

目的

設定變數值 (32 位元)。

原型

```
int SetVarN(char *varName, double data, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	變數名稱。
data	欲設定的變數值。 若使用者以型態 **float / long / short** 設定，其值會自動轉換成正確型態 **double**。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

6.4 GetVarN

目的

取得變數值 (32 位元)。

原型

```
int GetVarN(char *varName, double *pdata, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	變數名稱。
*pdata	變數值。 若使用者以型態 **float / long / short** 設定，其值會自動轉換成正確型態 **double**。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

6.5 SetVarN64

目的

設定變數值 (64 位元)。

原型

```
int SetVarN64(char *varName, _int64 data, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	變數名稱。
data	欲設定的變數值 (64 位元)。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

變數型態須為 64 位元，否則函式執行將失敗。

6.6 GetVarN64

目的

取得變數值 (64 位元)。

原型

```
int GetVarN64(char *varName, _int64 *pdata, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	變數名稱。
*pdata	變數值 (64 位元)。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

變數型態須為 64 位元，否則函式執行將失敗。

6.7 setArrayN

目的

設定陣列 (32 位元)。

原型

```
int setArrayN(char *varName, int *pdata, int num, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	陣列名稱。
*pdata	欲複製到驅動器中的應用程式陣列。
num	欲複製的變數數量。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

6.8 getArrayN

目的

取得陣列 (32 位元)。

原型

```
int getArrayN(char *varName, int *pdata, int num, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	陣列名稱。
*pdata	欲複製到應用程式中的驅動器陣列。
num	欲複製的變數數量。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

6.9 setArrayDN

目的

設定陣列 (64 位元)。

原型

```
int setArrayDN(char *varName, double *pdata, int num, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	陣列名稱。
*pdata	欲複製到驅動器中的應用程式陣列。
num	欲複製的變數數量。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

此函式相似於函式 setArrayN，差別在於 **pdata* 所指向的陣列為 64 位元。

6.10 getArrayDN

目的

取得陣列 (64 位元)。

原型

```
int getArrayDN(char *varName, double *pdata, int num, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	陣列名稱。
*pdata	欲複製到應用程式中的驅動器陣列。
num	欲複製的變數數量。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

此函式相似於函式 getArrayN，差別在於 **pdata* 所指向的陣列為 64 位元。

6.11 setArraySN

目的

設定陣列 (16 位元)。

原型

```
int setArraySN(char *varName, short *pdata, int num, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	陣列名稱。
*pdata	欲複製到驅動器中的應用程式陣列。
num	欲複製的變數數量。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

此函式相似於函式 setArrayN，差別在於 **pdata* 所指向的陣列為 16 位元。

6.12 getArraySN

目的

取得陣列 (16 位元)。

原型

```
int getArraySN(char *varName, short *pdata, int num, int slave=0, MDCE *pcom=NULL);
```

參數

*varName	陣列名稱。
*pdata	欲複製到應用程式中的驅動器陣列。
num	欲複製的變數數量。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

此函式相似於函式 getArrayN，差別在於 **pdata* 所指向的陣列為 16 位元。

6.13 getPac

目的

一次取得多個變數值。

原型

```
int getPac(char *varnames, char *l, void *data, int slave=0, MDCE *pcom=NULL);
```

參數

*varnames	一識別字，含一個或多個（至多 20 個）變數名稱。 請以『空白』或『逗號』區隔變數名稱。
*l	一字元陣列，按順序儲存與變數名稱相對應的變數型態；其大小應至少為 20。 1：short（16 位元整數）　　3：float（32 位元浮點數） 2：long（32 位元整數）　　8：64 位元變數
*data	一陣列，按順序儲存與變數名稱相對應的變數值；其大小取決於變數的數量。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

欲分別讀取此函式中的變數，請呼叫函式 GetVarN 或 GetVarN64。

範例

```
struct vargroup {
    long time;
    _int64 position;
    float velocity;
    short analoginput;
    long spare[10];
};
char vartypes[20];
```

```
char varlist[200]="fclk X_enc_pos X_vel_ff a2d[2]";
err=getPac(varlist, vartypes, &vargroup, 0, NULL);
```

6.14 getAndSetArrayN

目的

同時讀取與寫入陣列（32 位元），執行速度會比分別呼叫函式 getArrayN 與 setArrayN 還要快。

原型

```
int getArrayAndSetN(char *varrep, int numrep, void *parrrep, char *varset,
                    int numset, void *parrset, int slave, MDCE *pcom);
```

參數

*varrep	欲讀取的陣列名稱。
numrep	欲讀取的變數數量；範圍從 0 到 124 個 short 或 62 個 long。
*parrrep	欲複製到驅動器中的應用程式陣列。
*varset	欲寫入的陣列名稱。
numset	欲寫入的變數數量；範圍從 0 到 124 個 short 或 62 個 long。
*parrset	欲複製到應用程式中的驅動器陣列。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

parrrep 與 *parrset* 應指向適合當下驅動器、同樣型態的陣列（float / long / short）。

此函式不支援 64 位元。

6.15 getAndSetArrayDN

目的

同時讀取與寫入陣列（64 位元），執行速度會比分別呼叫函式 getArrayDN 與 setArrayDN 還要快。

原型

```
int getArrayAndSetDN(char *varrep, int numrep, double *parrrep, char *varset,
                     int numset, double *parrset, int slave, MDCE *pcom);
```

參數

*varrep	欲讀取的陣列名稱。
numrep	欲讀取的變數數量；範圍從 0 到 124 個 short 或 62 個 long。
*parrrep	欲複製到驅動器中的應用程式陣列。
*varset	欲寫入的陣列名稱。
numset	欲寫入的變數數量；範圍從 0 到 124 個 short 或 62 個 long。
*parrset	欲複製到應用程式中的驅動器陣列。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

此函式相似於函式 getAndSetArrayN，差別在於：

(a) 指向陣列的指標須為 double 型態。

(b) 驅動器裡的變數（**varrep* 與 **varset*）須為 float 型態。

(c) DLL 會在內部自行做型態轉換。

6.16 GetScopeData

目的

在同一 DSP 中斷讀取一組變數與一個時間計數器。

原型

```
int GetScopeData(char *varnames, double *pdata, short *pfclk, int slave=0,
                 MDCE *pcom=NULL);
```

參數

*varnames	一字串，含一個或多個變數名稱。 請以『空白』或『逗號』區隔變數名稱。
*pdata	一陣列，含變數的數據；其大小取決於變數的數量。
*pfclk	基於 DSP 取樣率的計數器；使用者可利用它將變數值畫在示波器上。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

(1) 此函式可應用於示波器。

(2) 請在延遲不超過 200 毫秒的情況下，持續呼叫此函式。否則，會回傳錯誤代碼 300。
因此，第一次呼叫此函式時，都會回傳錯誤代碼 300（在沒有其他錯誤的情況下）。

6.17 GetErrorStr

目的

取得錯誤的描述。

原型

```
void GetErrorStr(int errcode, char *str, int slave, char *errstr, MDCE *pcom=NULL);
```

參數

errcode	變數 / 狀態取得函式（SetVarN、GetVarN、setArrayN...）的回傳值。
*str	變數 / 狀態名稱。
slave	從站編號。
*errstr	錯誤的描述；其大小應至少為 200 位元組。
pcom	指向『通訊物件』的指標。

回傳值

無

備註

當其中一個變數 / 狀態取得函式回傳**非零值**時，請呼叫此函式。

(此頁有意留白。)

7. 取得狀態

狀態為 1 位元之變數 (如布林值)，表示數位輸入 / 輸出與驅動器的內部狀態 (如軸運行、軸位置錯誤標誌等)。狀態陣列中，每一個位元代表一個狀態。

7.1 setStateN

目的

設定狀態。

原型

```
int setStateN(int slave, char *stateName, int mode, MDCE *pcom=NULL);
```

參數

slave	從站編號。
*stateName	狀態名稱。
mode	欲設定的狀態。1 為觸發，0 為不觸發。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

7.2 getStateN

目的

取得狀態。

原型

```
int getStateN(int slave, char *stateName, int *state, MDCE *pcom=NULL);
```

參數

slave	從站編號。
*stateName	狀態名稱。
*state	狀態。1 為已觸發，0 為未觸發。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr 以取得錯誤的描述。

備註

(1) 讀取狀態會比讀取變數快得多，因狀態取自 DLL 的內部狀態表，不須透過通訊連結取得資訊。

(2) 狀態改變時，內部狀態表會自動更新。

(此頁有意留白。)

8. 執行 PDL 函式

8.1 RunFuncPdlN

目的

在驅動器中創造新的 task，並執行特定的 PDL 函式。

原型

```
int RunFuncPdlN(char *pdlName, char *errstr, int slave, MDCE *pcom=NULL);
```

參數

*pdlName	PDL 函式名稱。 PDL 程式中，任何以「_ (底線)」為開頭的標籤都會被上位控制器認得。
*errstr	錯誤的描述（不須呼叫函式 GetErrorStr）。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

執行此函式的 task ID（0~39）。

-1 – 函式執行失敗。請從 **errstr* 取得錯誤的描述。

備註

若已執行含有特定 PDL 函式的 task，則不會創造新的 task，且會回傳既有的 task ID。

8.2 KillTask

目的

終止 / 暫停 / 繼續 task。

原型

```
int KillTask(int st, int en, int what, int slave, MDCE *pcom=NULL);
```

參數

st	起始 task ID。
en	結束 task ID。
what	0 → 終止 task(s)；1 → 暫停 task(s)；2 → 繼續 task(s)。
slave	從站編號。
pcom	指向『通訊物件』的指標。

回傳值

0 – 函式執行成功。

非零值 – 函式執行失敗。請呼叫函式 GetErrorStr (將 *str* 設為" ") 以取得錯誤的描述。

備註

若只須終止一個 task，請設定 *st* = **n**、*en* = **n+1**，其中 n 可為函式 RunFuncPdlN 所回傳的 task ID。

(此頁有意留白。)

9. 回調函式

使用者可透過呼叫本章的函式，來定義一個可隨時被改變或解除（將參數 *func* 設為 NULL）的回調函式。

9.1 setCallBackStEvnt

目的

狀態改變時會主動通知使用者，並進行相對應的處理工作。

原型

```
void setCallBackStEvnt(void (*func)(BYTE, int), MDCE *pcom);
```

參數

type	狀態類型。
id	狀態 ID。
pcom	指向『通訊物件』的指標。

回傳值

無

備註

type 與 *id* 是<組態檔名稱.stt>檔中為每個狀態所定義的狀態屬性。當狀態改變時，須呼叫函式 PutEventSt，且 *type* 中前兩個位元的其中一位元為 1(定義送出事件方向的改變)。詳細內容請參閱手冊《PDL Reference Manual for D-series Drives》。

範例

```
void PutEventSt(BYTE type, int id)
{
    char str[200];
    sprintf(str, "state change, type=%02x, id=%ld", type, id);
    MessageBox(str);
}
……
setCallBackStEvnt(PutEventSt, NULL);   //  以回調函式指標設置 DLL
```

10. 錯誤代碼

錯誤代碼表格如下所示。

表 11.1

代碼編號	描述
31	從站編號超出範圍（0~31）
101	讀取失敗
102	寫入失敗
103	封包錯誤
104	無回應
105	檢查總和錯誤
106	大小超出範圍
107	極性錯誤
108	過載錯誤
109	緩衝器溢位
110	封包錯誤
111	中止
112	網路同步錯誤
113	Can 錯誤
114	回應大小錯誤
115	起始位元組錯誤
116	主站啟動
117	EtherCAT mega-ulink 錯誤
118	EtherCAT mega-ulink 處於測試模式
131	USB 裝置不存在
151	通訊 FIFO 已滿
152	大小轉換錯誤
153	不能通訊
154	遠端無連線
155	本地通訊被阻檔
156	通訊失敗
157	透過 Ethernet 的遠端無連線
158	EtherCAT mega-ulink 從站無回應
201	無法辨識的變數
202	從站名稱未定義
203	從站變數清單裡含一個或多個無法辨識的變數
204	變數不為 short 型態
220	變數不為 64 位元型態

221	變數為 64 位元型態
251	從站裡有無法辨識的函式
252	從站名稱未定義
260	PDL 不在從站中執行
261	從站處於啟動模式
262	Task ID 超出範圍
263	從站不處於啟動模式
270	寫入緩衝器遭拒
271	緩衝器大小的設定超出範圍
272	讀取變數遭拒
273	變數數量超出範圍
274	預期的回應長度太大
275	只支援 16 / 32 位元型態的陣列
300	重新啟動示波器
301	示波器只支援 2 個變數
500	連結忙碌
501	無支援
502	主站啟動
503	無從站
504	連結錯誤
505	從站錯誤
506	無效位址
525	伺服器忙碌
600	狀態數量超出範圍
601	從站中有無法辨識的狀態
602	從站名稱未定義
651	不支援此通訊指令碼
801	額外的參數
802	遺失的參數
803	無法辨識的指令
1000	通訊物件尚未建立

(此頁有意留白。)

11. 範例程式

11.1 初始化 MPI 函式庫

MPI 函式庫初始化的流程如下圖所示。有需要的話再呼叫函式 showcomstatus。

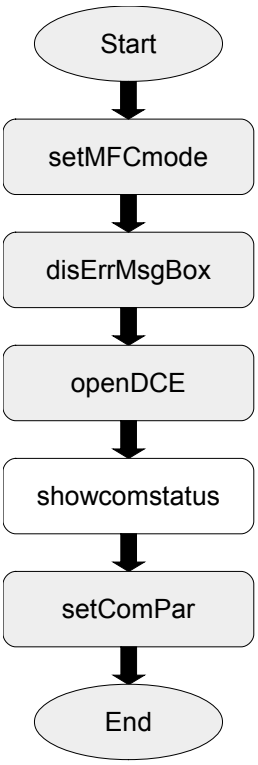


圖 12.1.1

範例

以下為驅動器型號 D2T-□□□□-S 搭配 AC 馬達做 MPI 函式庫初始化的範例。

```
setMFCmode();
disErrMsgBox(1);
pCom=openDCE("C:\\HIWIN\\dce\\", "d2.dce");
showcomstatus(pCom);
Status.AC_Connect=setComPar(
    m_AC_COM,       //  填入驅動器的通訊埠。若為 COM2：設為 2。
    nBaudrate1,     //  115200 bps：設為 8。
    nMode,          //  RS-232 ( USB 接頭 ): 設為 1；EtherCAT mega-ulink：設為 4。
    nTrid,          //  不使用：設為 0。
    nRcid,          //  不使用：設為 0。
    nCanbaudrate,   //  不使用：設為 0。
    nMsgStand,      //  不使用：設為 0。
```

```
    nCanpipelevel,  //  不使用：設為 0。
    nTimeout,
    nLocktime,
    nIternum,
    pCom);
```

11.2 編碼器回授

此範例顯示如何在主控台下取得馬達的編碼器回授。

備註

(1) 連線至驅動器前，請呼叫函式 setMFCmode 以初始化 DLL。
(2) 初始化 DLL 後，請呼叫函式 openDCE 以建立通訊物件。
(3) 建立通訊物件後，請呼叫函式 setComPar 以設定驅動器的通訊參數。
(4) PC 與驅動器的連結建立完成時，請呼叫函式 GetVarN 以取得驅動器的參數。
(5) 使用者可利用驅動器裡的馬達回授位置參數 *X_enc_pos*，來讀取馬達的編碼器回授。

所需 MPI 函式

```
void setMFCmode();
MDCE *openDCE();
int setComPar();
int GetVarN();
void deleteDCE();
```

範例

```
#include "stdafx.h"
#include <Windows.h>
#include <conio.h>
#include <stdio.h>
#include "mpint.h"  //  加入 MPI 的標頭

void main()
{
    MDCE *pCom=NULL;  //  宣告
    //  MPI 函式：DLL 初始化
    setMFCmode();
    //  MPI 函式：打開通訊物件
    pCom=openDCE("C:\\HIWIN\\dce\\", "d2.dce");
    int nPort, nBaudrate, bConnected;
    do
```

```
{
    printf("\n Input number to select Communication port (1:COM1, 2:COM2, …) \
    or -1 quit: ");
    scanf("%d", &nPort);
    if (nPort == -1)
    goto End;  //  得到-1 以結束應用程式
    //  MPI 函式：開始跟驅動器通訊
    bConnected = setComPar(
    nPort, nBaudrate=8, 1, 0, 0, 0, 0, 100, 200, 6, pCom);
}while (bConnected!=0);  //  bConnected = 0 代表通訊成功。否則，請重新選擇通訊埠。

int key;
printf("Command List:\n");
printf("Input ‘r’ : Read Encoder\n");
printf("Input ‘q’ : Quit The Application\n");

Loop :
    printf(">>Command Input: ");
    key=getche();
    printf("\n");
    switch(key)
    {
        //+++++++++++++++++ 讀取馬達回授位置 +++++++++++++++++//
        case ‘r’ :
        {
            char*  var = "X_enc_pos";
            //  驅動器裡的馬達回授位置變數
            double fpos;
            //  MPI 函式：取回馬達回授位置
            if (GetVarN(var,&fpos,0) == 0)
                printf("Motor feedback position =%.0f counts.\n", fpos);
            else
                printf("Failed to read feedback position!\n");
```

```
            }
            break;
            //+++++++++++++++++ 結束應用程式 +++++++++++++++++//
            case 'q':
                goto End;
                break;
        }
    goto Loop;
    End:
    //  MPI 函式：關閉通訊物件
    deleteDCE(pCom);
    pCom=NULL;
    system("PAUSE");
```