μPAC-5001D-CAN2 使用手冊
（使用 C 語言）

版本 1.1，2016 年 6 月

產品技術服務與使用資訊

μPAC-5001D-CAN2

保固說明

泓格科技股份有限公司（ICP DAS）所生產的產品，均保證原始購買者對於有瑕疵之材料，於交貨日起保有為期一年的保固。

免責聲明

泓格科技股份有限公司對於因為應用本產品所造成的損害並不負任何法律上的責任。本公司保留有任何時間未經通知即可變更與修改本文件內容之權利。本文所含資訊如有變更，恕不予另行通知。本公司盡可能地提供正確與可靠的資訊，但不保證此資訊的使用或其他團體在違反專利或權利下使用。此處包涵的技術或編輯錯誤、遺漏，概不負其法律責任。

版權所有

商標識別

本文件提到的所有公司商標、商標名稱及產品名稱分別屬於該商標或名稱的擁有者所有。

技術服務

如有任何問題，請與本公司客服聯絡，我們將盡速為您服務。

Email 信箱 ： service@icpdas.com

目錄

1. 簡介

μPAC-5001D-CAN2 產品是一種掌上型的可程式自動控制器(PAC)，內含豐富且多樣化的週邊與通訊埠，能夠整合 CAN bus、RS-232、RS-485 與 Ethernet 等多種不同通訊介面的設備。

為了增加模組的開放性與應用的彈性，μPAC-5001D-CAN2 產品提供了類似 DOS 作業系統的即時單工作業系統(real-time single-task operation system)。用戶能夠透過 C / C++語言開發滿足各式需求的應用程式。

在應用程序開發方面，μPAC-5001D-CAN2 產品提供各種週邊元件的函式庫與範例程式。如此豐富的函式庫，使用者也許會困惑該如何使用?為此，也提供相關函式庫的範例程式。如果使用者需要使用各種通訊埠、看門狗、即時時鐘(RTC)、七段顯示器等周邊，或於 Flash、EEPROM、MicroSD 存入與讀取資料，甚至是利用 64-bit 硬體唯一序號保護所開發的應用軟體，都可以藉由修改與合併這些範例程式，輕鬆而快速地完成符合各式需求的應用程式開發工作。

為了適應工業嚴苛的工作環境，μPAC-5001D-CAN2 產品，除了低功耗與無風扇的設計外，也於電路設計上加入各種抗干擾的保護元件，並滿足了寬操作溫度與寬工作電壓的要求。

依據上述的各項特色，μPAC-5001D-CAN2 產品非常適合運用於整合各種通訊介面的資料收集器或作為資料處理中心，亦或者將它設計成為轉換不同通訊協定的網路閘道器或橋接器。

1.1. 特色

➢ 多種 CPU 與 作業系統（OS）可供選擇

MiniOS7
80186 CPU
μPAC-5000 系列

- 類似 DOS 的嵌入式作業系統
- 啟動時間 0.4 ~ 0.8 秒
- 內建硬體診斷功能
- C 語言程式設計之標準版

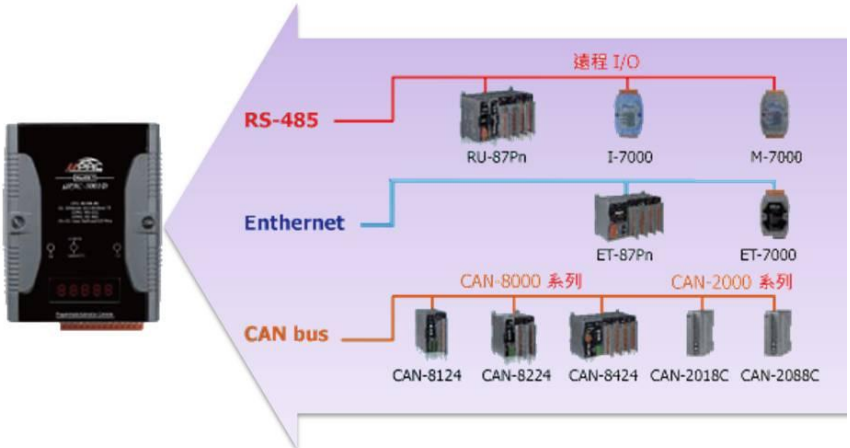
➢ 遠程 I/O 模組與 I/O 擴充單元

內建 CAN、RS-232/485 與 Ethernet 通訊埠，μPAC-5001D-CAN2 可以連接 CAN/RS-485/Ethernet 的遠端 IO 裝置，如:CAN-8x23/CAN-8x24/RU-87Pn/ET-87Pn 或遠端 IO 模組，如: CAN-2000D/CAN-2000C/I-7000/M-7000/ET-7000。

➢ 多通訊介面

支援多種的通訊介面，可用來擴展 I/O 並連接外部設備:

- Ethernet
- RS-232/485
- CAN bus



➢ 多種記憶體擴充

µPAC-5001D-CAN2 提供了多種記憶體儲存選項，例如: EEPROM，Flash，電池備援 SRAM 或 microSD，客戶可選擇適用的記憶體。

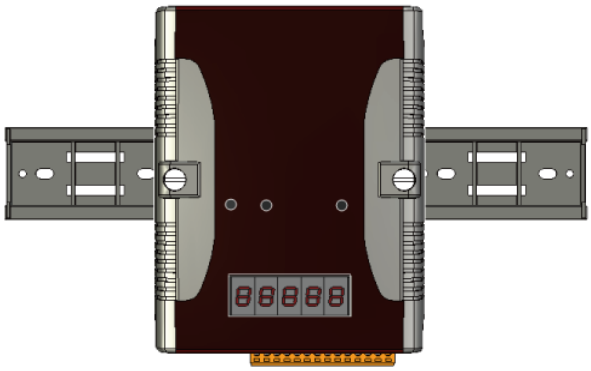
- 16 KB EEPROM: 儲存不需常更新的參數。
- microSD:
  實現可攜式資料紀錄（Data Logging）應用。
  - ✧ MiniOS7 平台：支援最大 2 GB
  - ✧ Linux and WinCE 平台：支援最大 32 GB

➢ 64 位元硬體唯一序號，以保護您的程式

可分配給每個硬體設備唯一的 64 位元序號，用來保護您軟體的著作權益。

➢ 輕巧且安裝容易

µPAC-5001D-CAN2 擁有修長的外型（91 mm x 132 mm x 52 mm）可搭配導軌（DIN-Rail ）安裝於狹窄的空間。



➢ 塑膠與金屬外殼

一般為塑膠外殼，客戶也可選用金屬外殼。

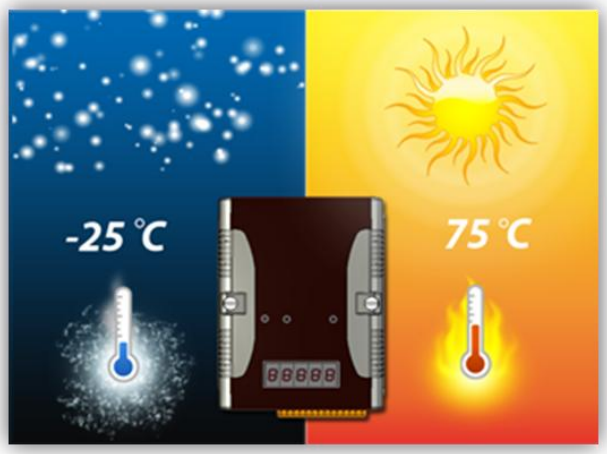
➢ 備援電源輸入

為了防止因供電不穩所造成的故障，µPAC-5001D-CAN2 設計了兩組電源輸入互為備援。若其中一組電源失效了，電源模組會切換至另一組電源輸入，而其繼電器輸出(Relay Output) 可對外界發出警告訊號。

➤ 於嚴苛環境下仍具高可靠度

μPAC-5001D-CAN2 可在廣泛的溫／濕度下運作。

- 操作溫度:
  -25˚C ~ +75 ˚C
- 儲存溫度:
  -30˚C ~ +80 ˚C
- 相對濕度:
  10 ~ 95% RH（非凝露）



1.2. μPAC-5001-CAN2 規格

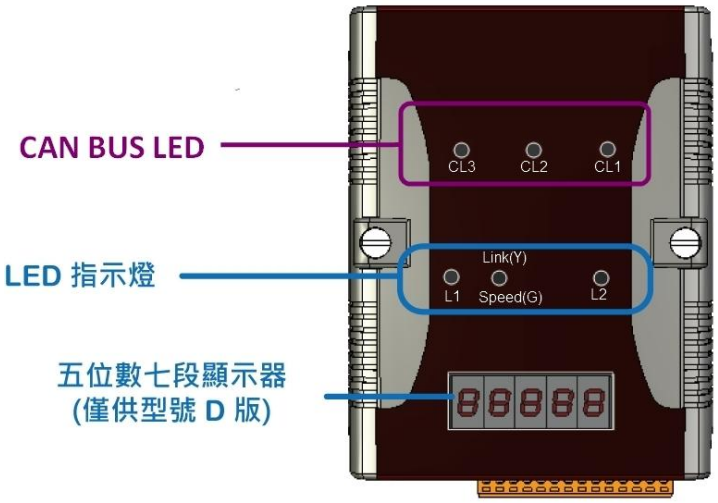
1.2.1. μPAC-5001D-CAN2

型號	μPAC-5001D-CAN2
系統軟體	
作業系統	MiniOS7（類似 DOS 的嵌入式作業系統）
程式下載介面	RS-232（COM1）或 Ethernet
編程語言	C 語言
編譯器	TC++ 1.01／TC2.01（免費軟體）／BC++3.1 ~ 5.2x／MSC 6.0／MSVC++ （1.5.2 之前版本）
CPU 模組	
CPU	80186 或相容（16-bit，80 MHz）
SRAM	512 KB
Flash	512 KB; 抹除單位為磁區（64 KB）; 可重覆讀取/寫入 100,000 次。
microSD 擴充	有，可支援 1 或 2 GB microSD
EEPROM	16 KB
NVRAM	31 Bytes（電池備援，資料可保存 5 年）
即時時鐘（RTC）	可讀/ 寫 年、月、日、時、分、秒，並提供星期資訊。
64 位元硬體序號	有
看門狗機制	有（0.8 秒）
乙太網介面	
RJ-45	RJ-45 x 1，10/100 Base-TX（Auto-negotiating，Auto MDI/MDI-X，LED indicators）
UART	
COM1	RS-232
COM2	RS-485（D2+，D2-），內含 self-tuner ASIC 晶片）
LED 指示燈	
可程式 LED	5
LED 顯示器	有五位數七段 LED 顯示器。
CAN BUS 介面	
控制器	NXP SJA1000T, 使用 16 MHz 震盪頻率
收發器	NXP TJA1042
通訊埠 數量	2
接頭	18 針螺絲端子（CAN_GND, CAN_L, CAN_GND）
通訊速率(bps)	5 k ~ 1 M(使用者自定義)
終端電阻	跳線設定 120 Ω 終端電阻
規範	ISO 11898-2, 支援 CAN 2.0A 與 CAN 2.0B
機構設計	
尺寸（W x H x D）	91 mm x 123 mm x 52 mm
安裝方式	導軌式（DIN-Rail）
工作環境	
操作溫度	-25 ~ + 75 °C
儲存溫度	-30 ~ +80 °C
相對溼度	10 ~ 90 % RH（無凝露）
電源	
保護	電源反極性保護
屏蔽地線 (Frame Ground)	有（ESD 保護）
輸入電源	+12 ~ +48 V_{DC}
隔離	–
備援電源輸入	有
功耗	3 W

1.3. 概述

以下將簡略的描述控制器的組成要件與其狀態。

前面板



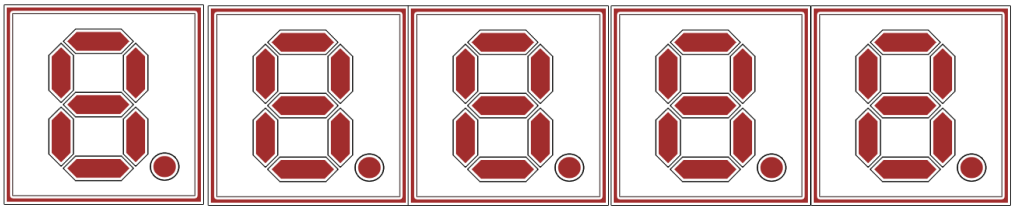
位於前面板的 LED 指示燈與五位數七段顯示器，提供您相當便利且更快速、簡易的方式來顯示診斷資訊。

LED 指示燈

LED 指示燈位於 μPAC-5001D-CAN2 的前面板，其功能如下表所示。

指示燈	狀態	意義
L1	閃爍	使用者自訂 LED
L2	暗	使用者自訂 LED
Link (G)	亮	已偵測到網路連線
	暗	未偵測到網路連線
	閃綠燈	已接收到網路封包
CL1	暗	使用者自訂 LED
CL2	暗	使用者自訂 LED
CL3	暗	使用者自訂 LED

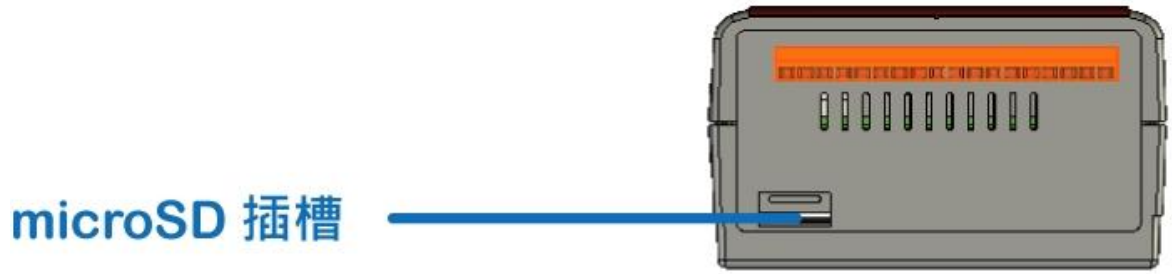
➢ 五位數七段 LED 顯示器



μPAC-5001D-CAN 配備有一個五位數七段 LED 顯示器，可用來顯示 0 ~ 9 的十進制號碼並提供了相當便利的方式，以數字形式來顯示數位資料。

頂端面板

microSD 記憶體插槽位於頂端面板，提供您簡易的擴充方式。

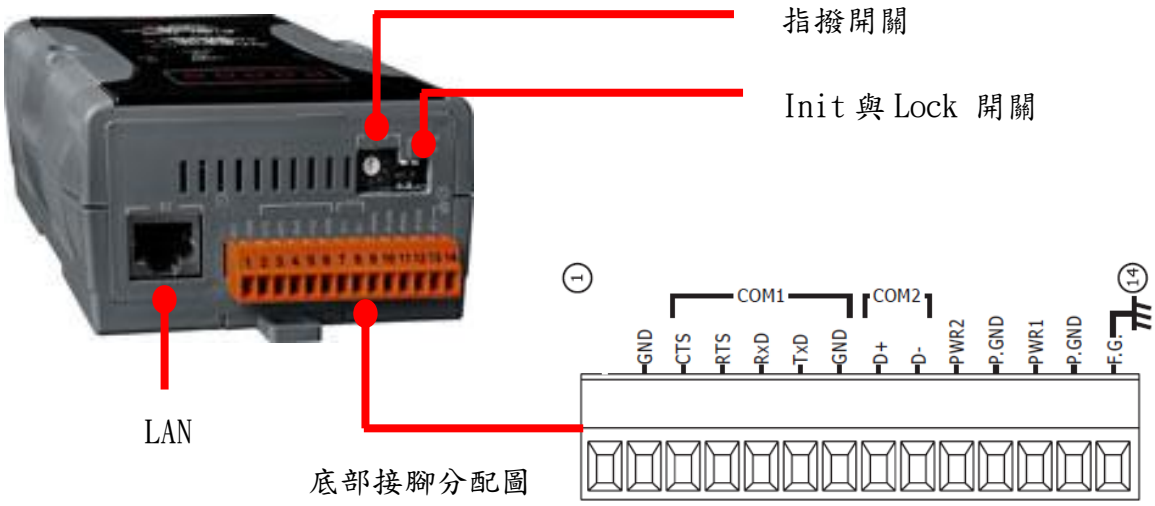


microSD 記憶體插槽

μPAC-5001D-CAN2 配備有一個 microSD 插槽並可支援最大 2 GB 容量的 microSD 卡。

底部面板

設定開關與通訊介面皆位於底部面板，提供您以簡易的方式進行系統調整與配線連接。



Init 開關: 操作模式選擇開關

ON: MiniOS7 設定模式
OFF: 韌體運行模式

於 μPAC-5001D-CAN2，切換開關固定在 OFF 位置。只有在更新 μPAC-5001D-CAN2 的韌體或作業系統時，才將開關切換至 ON。

待更新完成後，請將開關切換回 OFF 位置。

Lock 開關: Flash 記憶體的防寫開關

ON: 啟用防寫功能
OFF: 關閉防寫功能

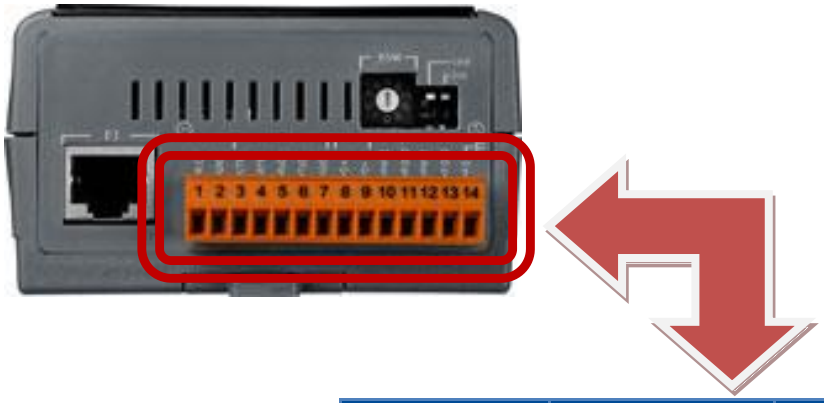
μPAC-5001D-CAN2 的 Flash 記憶體防寫功能可完全鎖定，以避免重要資料被修改或刪除。

LAN

μPAC-5001D-CAN2 含有一個供網路設備使用的乙太網路埠，可支援 RJ-45 接頭。

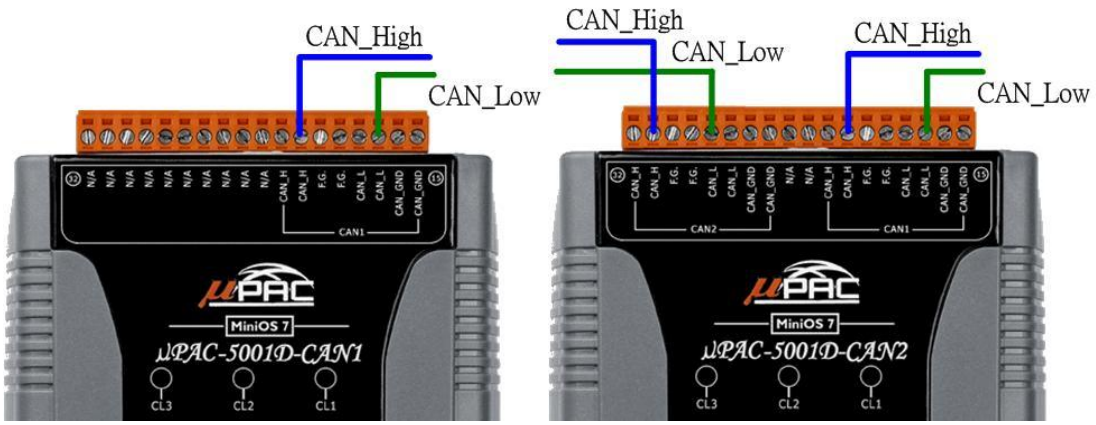
底部接腳分配圖

下列為接線端子的腳位分配：

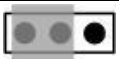
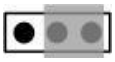


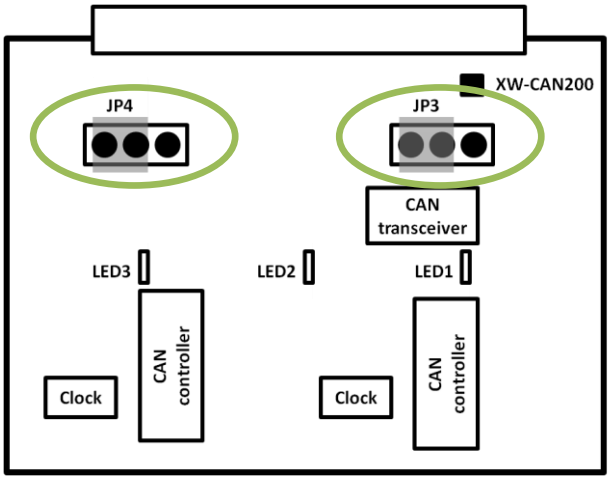
接腳	信號	說明
1	N.C	未指定
2	GND	接地
3	CTS	RS-232
4	RTS	
5	RxD	
6	TxD	
7	GND	
8	D+	RS-485
9	D-	
10	PWR2	電源輸入 1
11	P.GND	
12	PWR1	電源輸入 2
13	P.GND	
14	F.G.	屏蔽地線

CAN BUS 端接腳分配圖



終端電阻跳線選擇

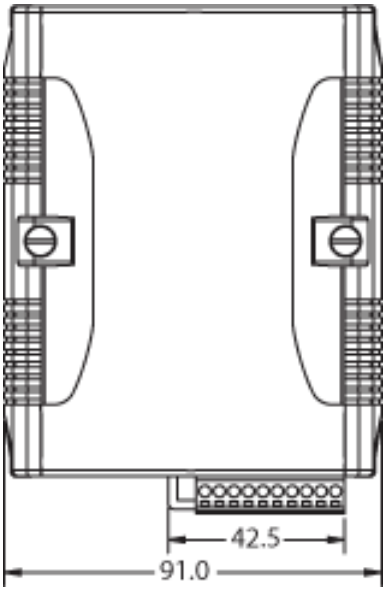
使用終端電阻(120Ω)	不使用終端電阻
	



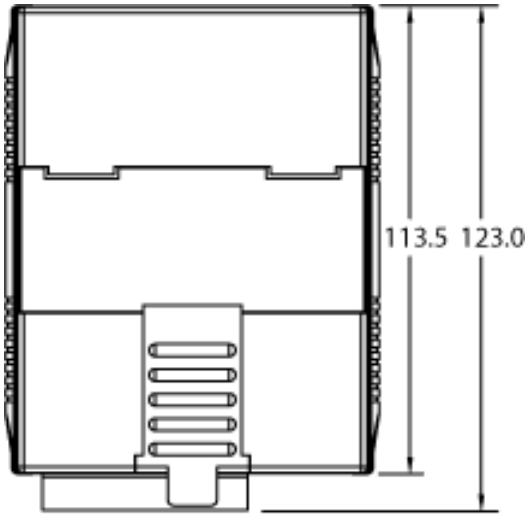
1.4. 尺寸

尺寸以毫米（mm）為單位。

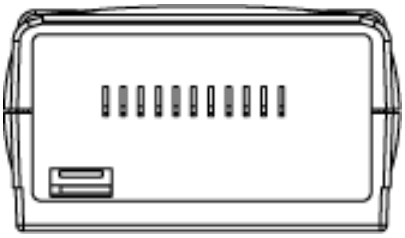
正面視圖



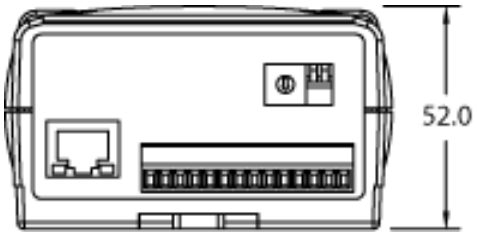
背部視圖



頂端視圖



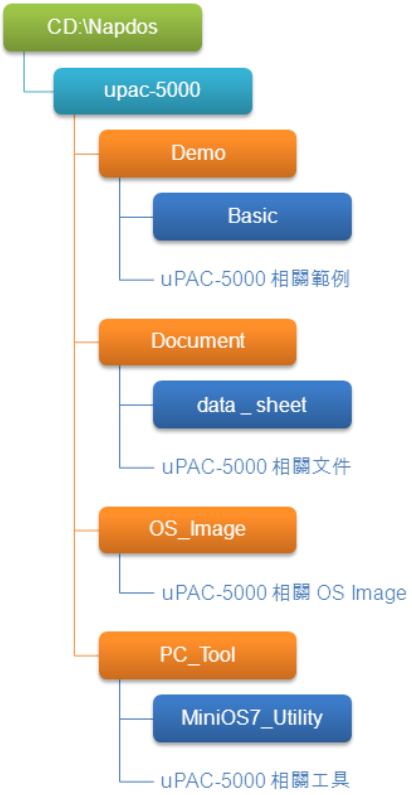
底部視圖



1.5. 隨貨光碟內容

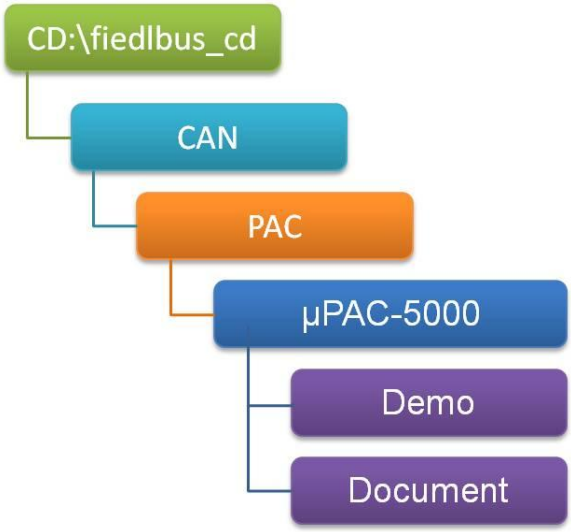
CD:\Napods

隨貨光碟中提供了驅動程式、軟體設定工具、所有相關文件…等，如下圖所示。



CD:\fieldbus_cd

隨貨光碟中提供了驅動程式、軟體設定工具、所有相關文件…等，如下圖所示。



2. 快速上手

若您是初次使用本產品，請由此章節著手。本章節提供了 μPAC-5001D-CAN2 的基本安裝、設定與使用說明導覽。

在開始安裝之前，請先檢查貨品內容。如有任何品項損毀或遺失，請與我們聯繫。

除了『快速安裝指南』，包裝中含有下列項目：

μPAC-5001D-CAN2

軟體工具光碟

RS-232 纜線
CA-0910)

螺絲起子
(1C016)

2.1. 硬體安裝

在安裝硬體之前，您必須先對硬體規格有初步的了解，例如：硬碟容量、電源的可用電壓輸入範圍、通訊介面的類型。請參閱章節 “1.3. 規格” 以取得完整的硬體資訊。

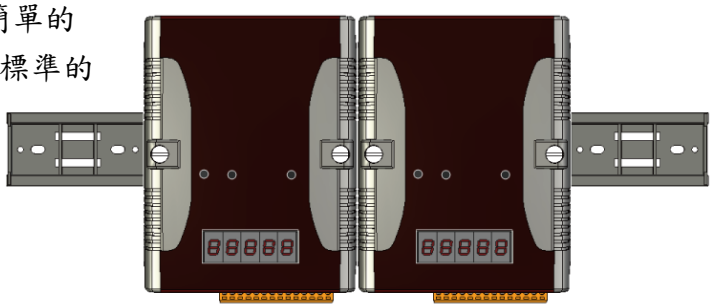
以下將一步步地引導您，部署基本的 μPAC-5001D-CAN2 系統。

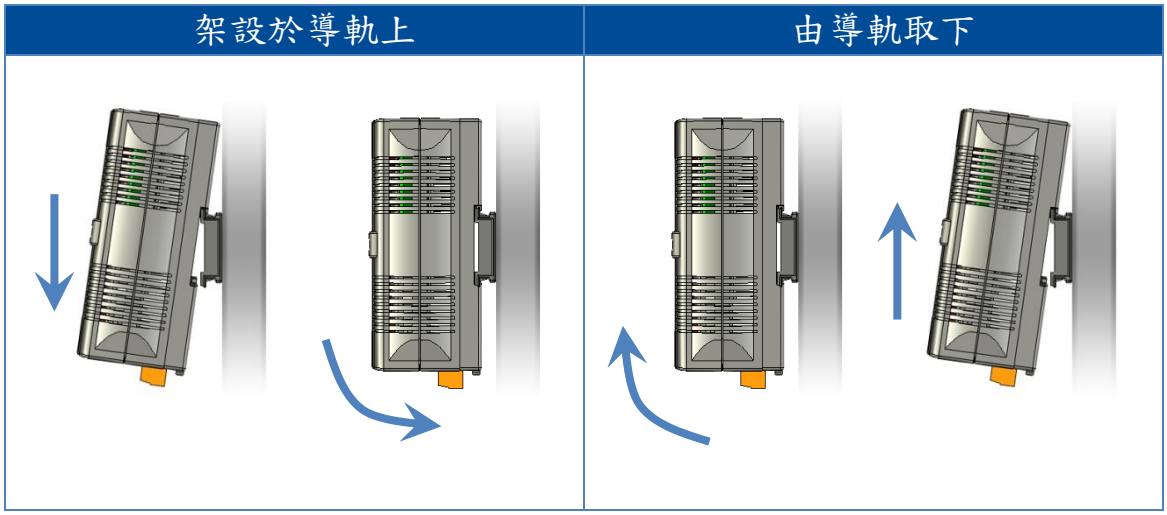
步驟 1: 架置硬體

μPAC-5001D-CAN2 的機殼背部，可採用導軌式或背掛式的架設方式。

➤ 導軌式安裝

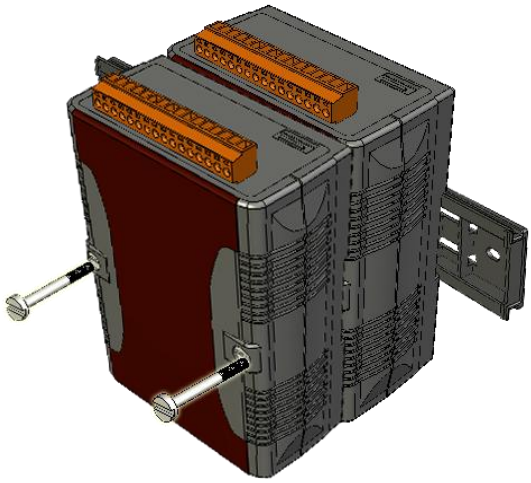
μPAC-5001D-CAN2 背後有一個簡單的導軌夾，用來將其穩固地架設在標準的 35 mm 導軌上。





➢ 背掛式安裝

μPAC-5001D-CAN2 正面的兩端擁有 2 個鎖孔，可供背掛式安裝使用。



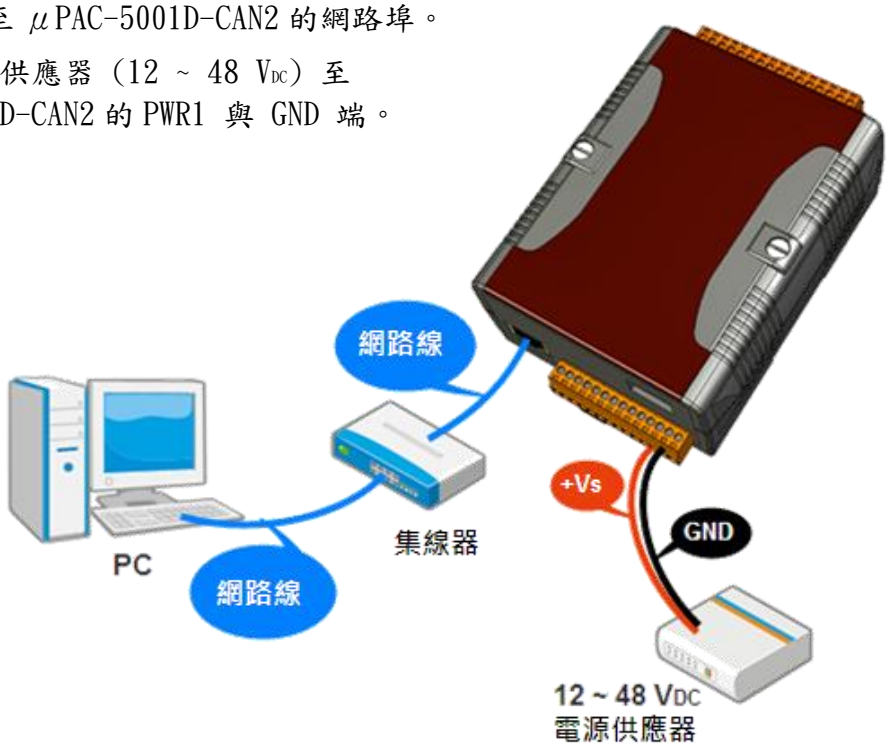
步驟 2: 連接 μPAC-5001D-CAN2 與 PC 並設置電源供應器

μPAC-5001D-CAN2 的 RJ-45 乙太網路埠，可用來連接至網路集線器（Hub）/交換器（Switch）和 PC，並採用標準的 12 VDC 電源供應器 或 網路供電（PoE）交換器來供電。

透過標準的 12 VDC 電源供應器供給外部電源

i. 連接 PC 至 μPAC-5001D-CAN2 的網路埠。

ii. 連接電源供應器（12 ~ 48 VDC）至 μPAC-5001D-CAN2 的 PWR1 與 GND 端。

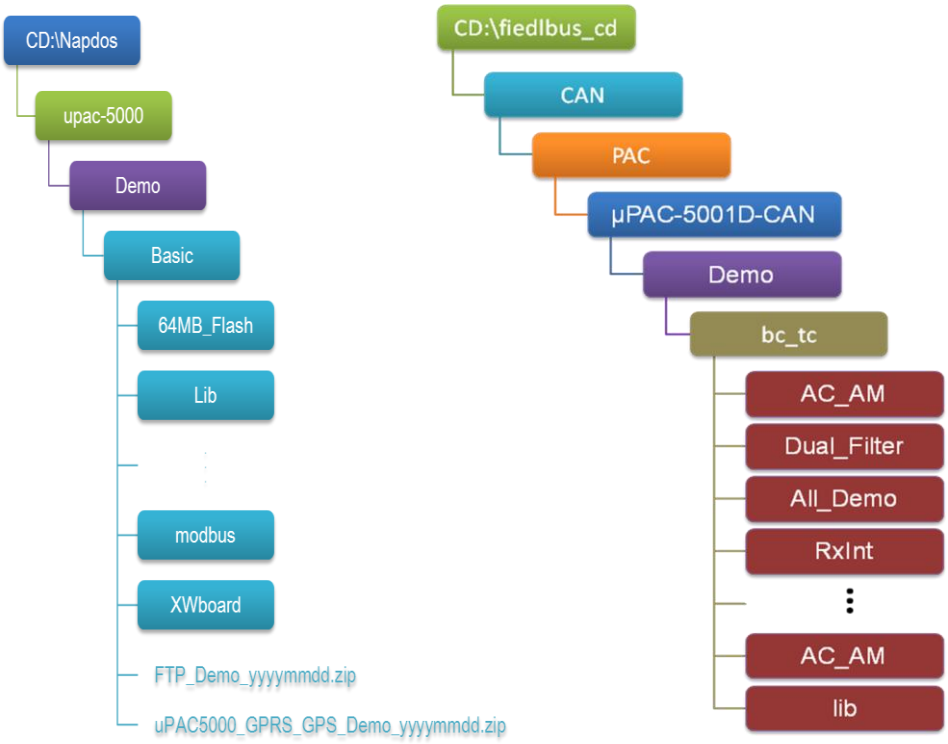


2.2. 軟體安裝

隨貨的光碟中包括了 μPAC-5001D-CAN2 的 API、範例程式與可開發個人應用的發展工具。
以下將一步步地引導您安裝 API、範例程式與工具。

步驟 1: 將隨貨光碟中的 “Demo” 目錄複製到 PC。

“Demo” 目錄中含有客戶開發個人應用所需的重要資源，包括：函式庫，標頭檔，範例程式與更多資訊，如下所示。



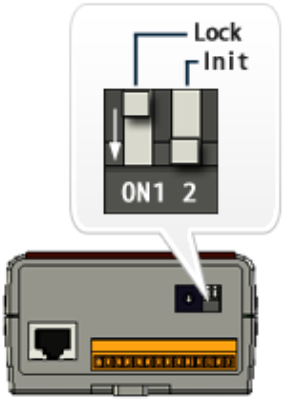
步驟 2: 安裝 MiniOS7 Utility。

MiniOS7 Utility 是一套工具軟體，用於管理 MiniOS7 設備（例如: μPAC-5001D-CAN2）。它區分為四個部份 – 系統偵測、通訊管理、檔案管理與 OS 載入程式。

MiniOS7 Utility 可由隨貨光碟或是 FTP 網站中取得:
CD:\Napdos\minios7\utility\minios7_utility\
ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/minios7/utility/minios7_utility/

2.3. 設定啟動模式

上傳程式至 μPAC-5001D-CAN2 之前，必須先進入初始（Init）模式並關閉防寫功能。請確認 Lock 的位置在 “OFF”，且 Init 的位置在 “ON”。



2.4. 上傳 μPAC-5000 程式

MiniOS7 Utility 是一套工具軟體，用於管理使用 MiniOS7 的設備(例如: μPAC-5001D-CAN2)。此軟體可分為四個部份 – 系統偵測，通訊管理，檔案管理與 OS 載入程式。

使用 MiniOS7 Utility 上傳程式之前，請確認 μPAC-5001D-CAN2 已連接至 PC。

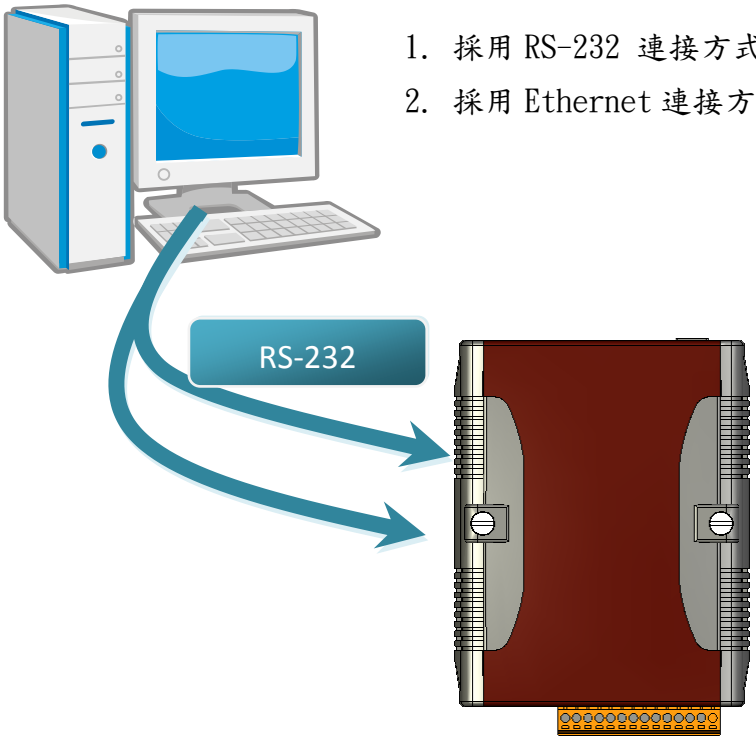
下列為主要的上傳步驟:

1. 建立 PC 與 μPAC-5001D-CAN2 之間的連線。
2. 上傳程式至 μPAC-5001D-CAN2 並執行程式。
3. 設定啟動後，自動執行程式。

後續將說明這些步驟的詳細內容。

2.4.1. 建立 PC 與 μPAC-5001D-CAN2 之間的連線

以下有兩種方式可建立 PC 與 μPAC-5001D-CAN2 之間的連線。



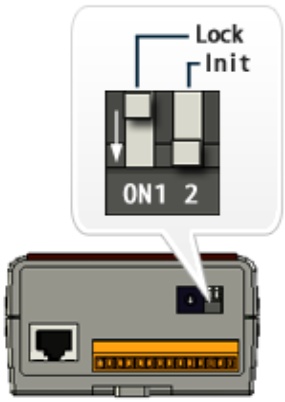
1. 採用 RS-232 連接方式
2. 採用 Ethernet 連接方式

後續將詳細說明此兩種連接方式。

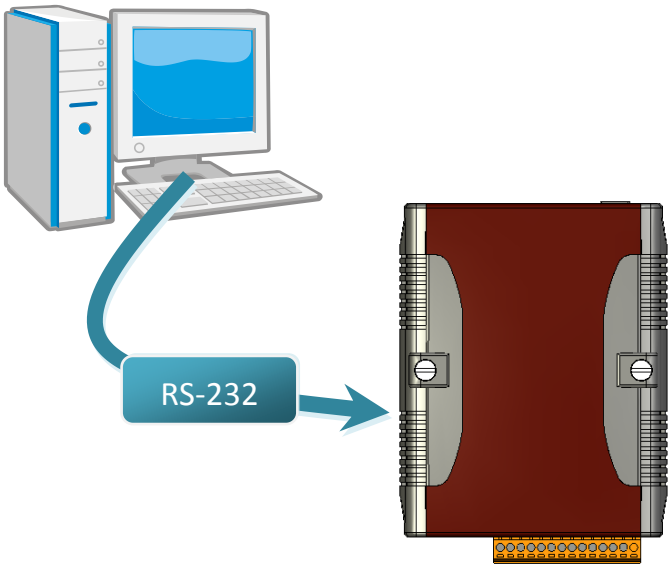
2.4.1.1. 採用 RS-232 連接方式

下列將一步步地引導您，如何使用 RS-232 方式與 PC 相連接。

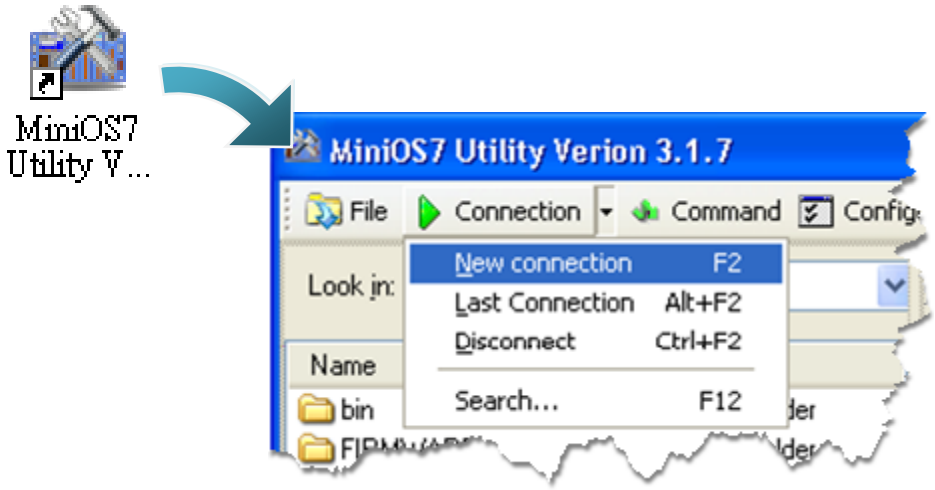
步驟 1: 將開關 Lock 切換至 “OFF”，並將 Init 切換至 “ON”。



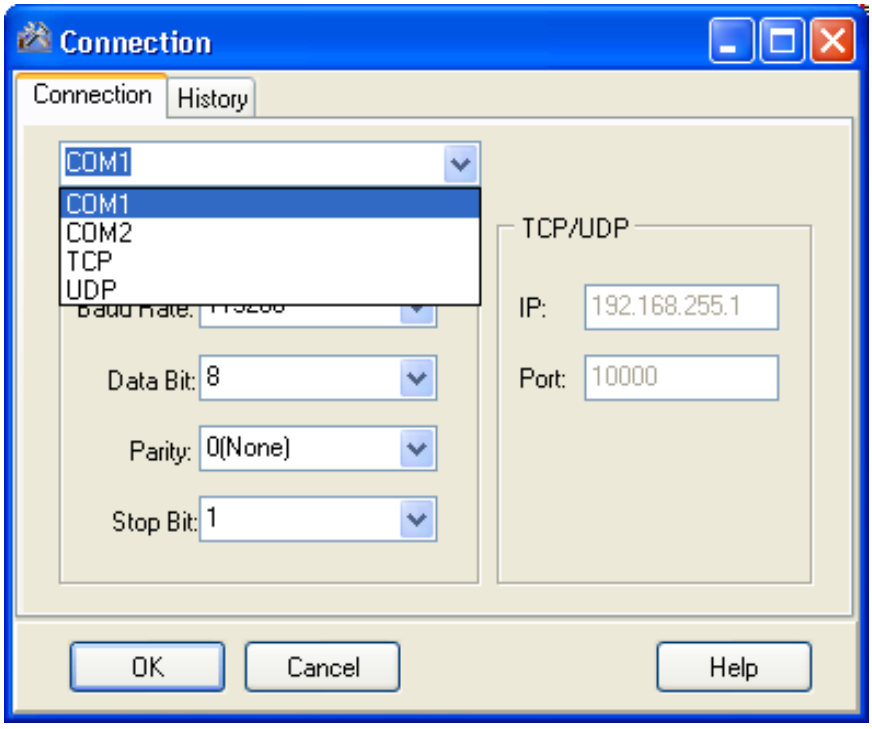
步驟 2: 將 RS-232 纜線 (CA-0910) 連接至 PC。



步驟 3: 執行 MiniOS7 Utility。
步驟 4: 點選功能表 “Connection” > “New connection” 功能。



步驟 5:(“Connection” 視窗 > “Connection” 頁籤 > 下拉選單)
選取 “COM1” 並點選 “OK”。

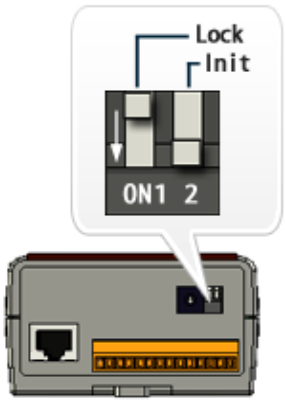


步驟 6: 已建立連線。

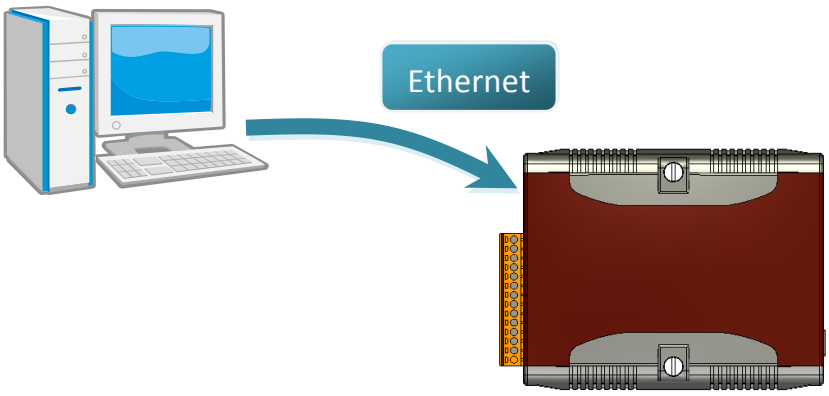
2.4.1.2. 採用 Ethernet 連接方式

下列將一步步地引導您，如何使用 Ethernet 方式與 PC 相連接。

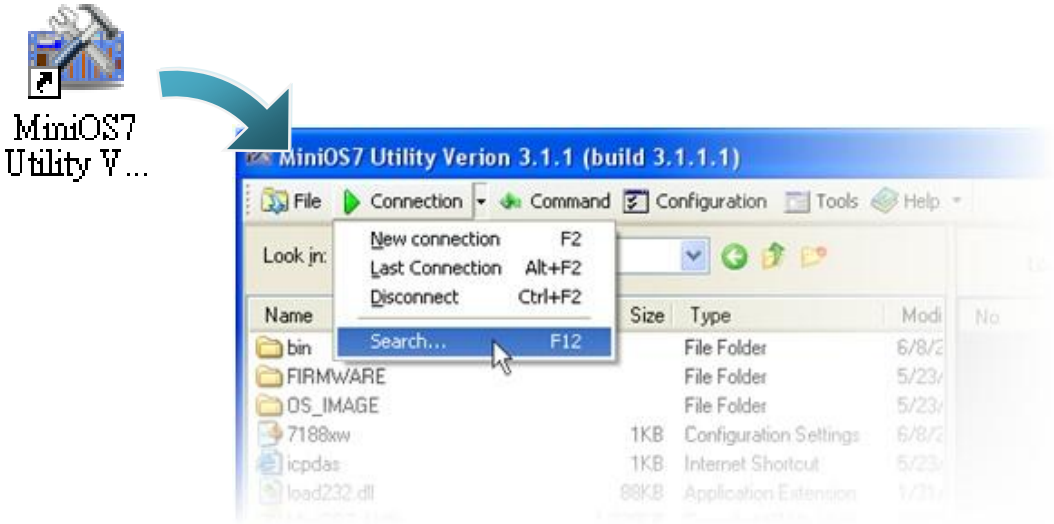
步驟 1: 將開關 Lock 切換至 “OFF”，並將 Init 切換至 “ON”。



步驟 2: 將乙太網路線連接至 PC。

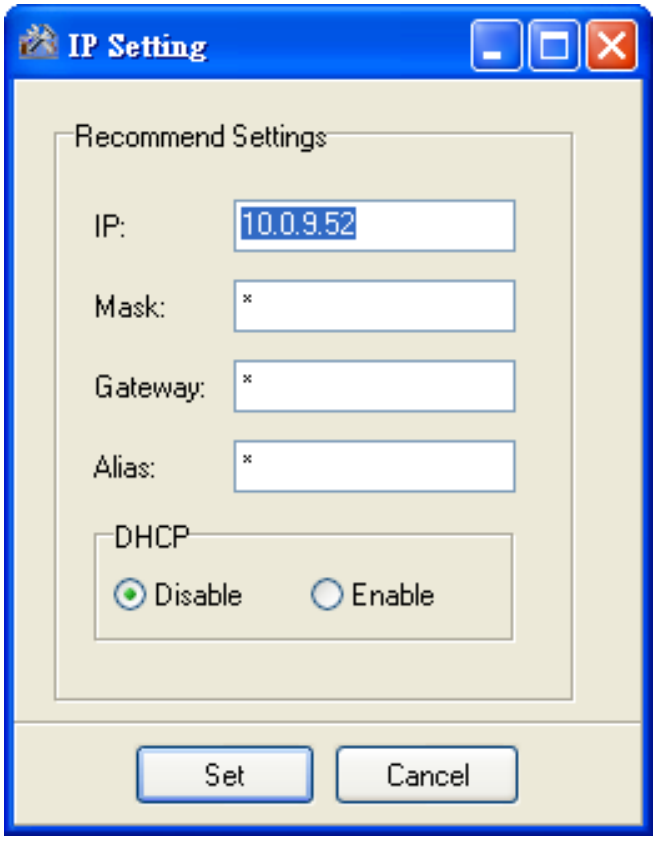


步驟 3: 執行 MiniOS7 Utility。
步驟 4: 點選功能表 “Connection” > “Search” 功能。

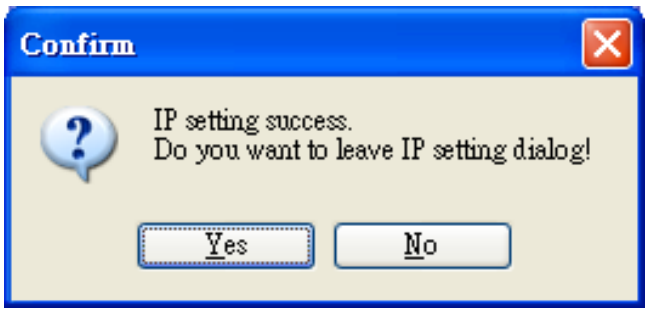


步驟 5: (“MiniOS7 Scan” 視窗)
選取 模組名稱 並點選工具按鈕 “IP setting”。

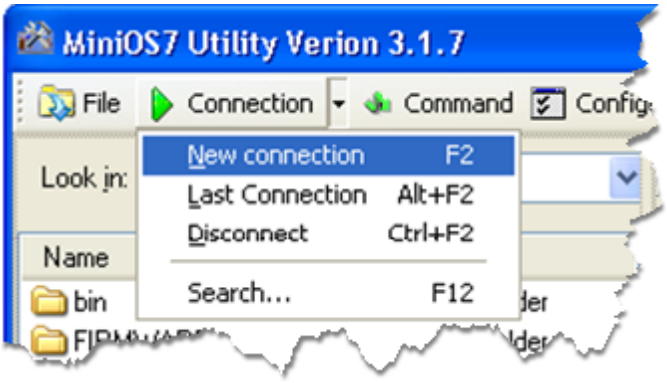
步驟 6: （“IP Setting” 視窗）
設定 “IP” 位址並點選 “Set”。



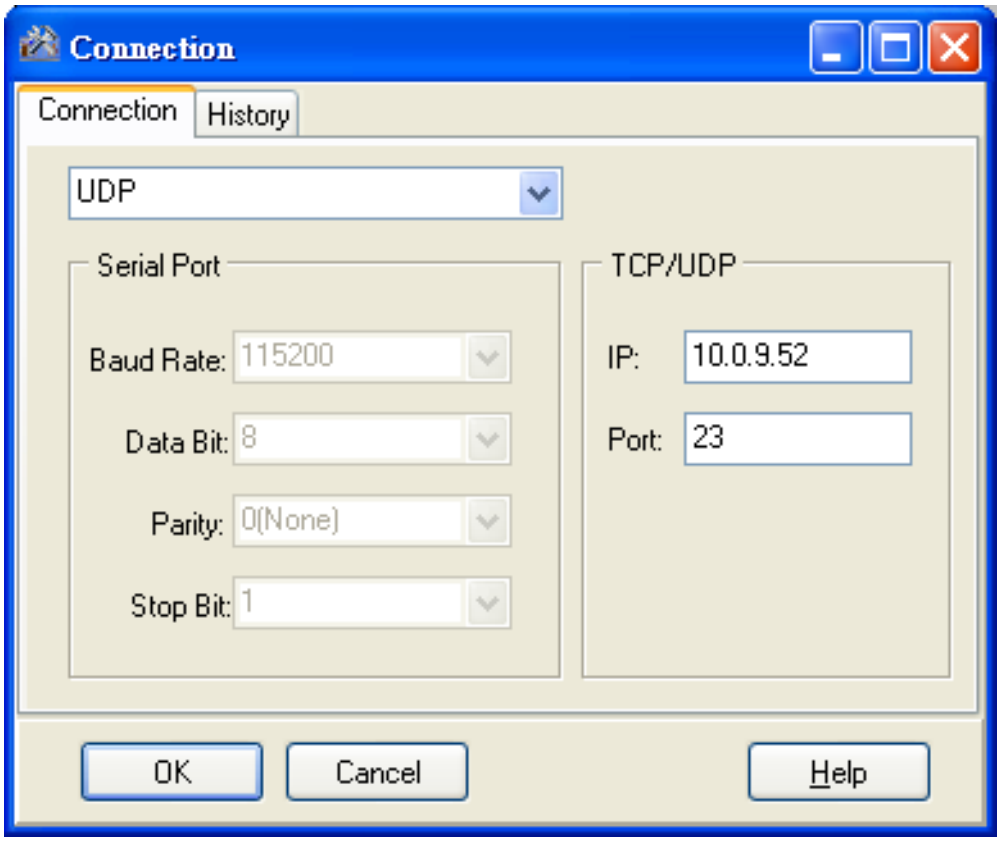
步驟 7: （“Confirm” 視窗）點選 “Yes”。



步驟 8: 點選功能表 “Connection” > “New connection” 。



步驟 9: （“Connection” 視窗 > “Connection” 頁籤 > 下拉選單）
選取 “UDP” 並指定 IP 位址，再點選 “OK” 。

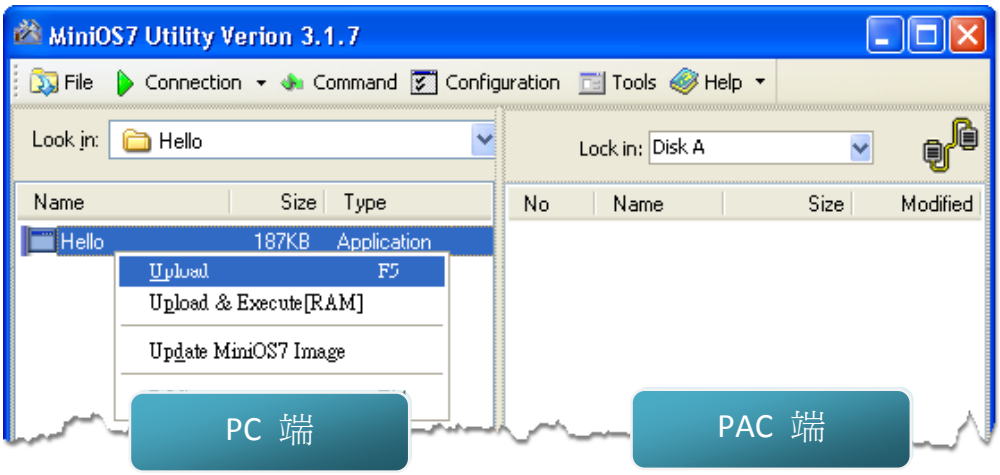


步驟 10: 已建立連線。

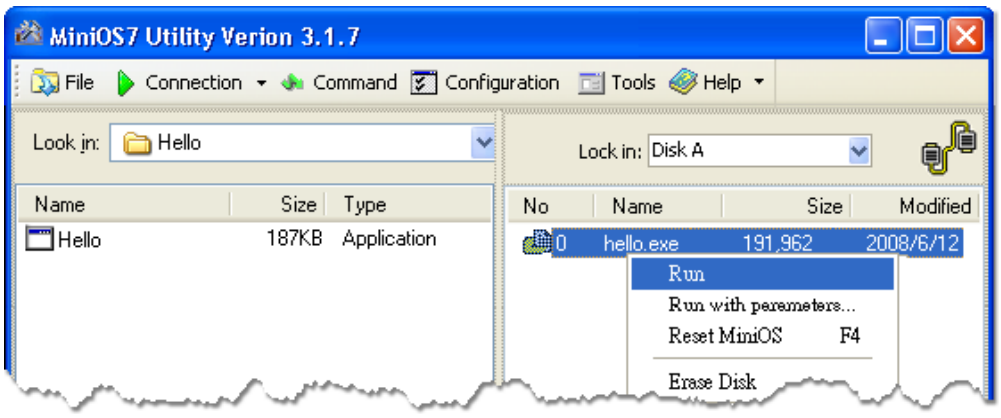
2.4.2. 上傳並執行 µPAC-5001D-CAN2 程式

上傳並執行 µPAC-5001D-CAN2 程式之前，您必須先建立 PC 與 µPAC-5001D-CAN2 之間的連線。請參閱章節 “2.4.1. 建立連線” 以取得詳細內容。

步驟 1: PC 端，於欲上傳的檔案上按滑鼠右鍵並點選 “Upload”。



步驟 2: PAC 端，於欲執行的檔案上按滑鼠右鍵並點選 “Run”。



2.4.3. 設定自動執行程式

上傳檔案至 µPAC-5001D-CAN2 後，若您希望於 µPAC-5001D-CAN2 啟動時能自動地執行程式，有個很簡單的方式，您可建立一個名稱為 autoexec.bat 的批次檔並將其上傳至 µPAC-5001D-CAN2，於下次啟動時程式將自動執行。

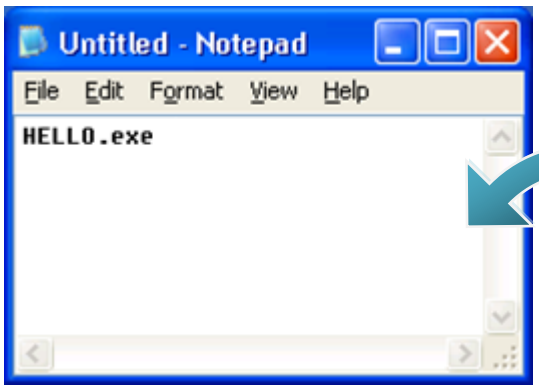
例如，設定在啟動時即執行 “hello” 程式。

步驟 1: 建立一個 autoexec.bat 檔案。

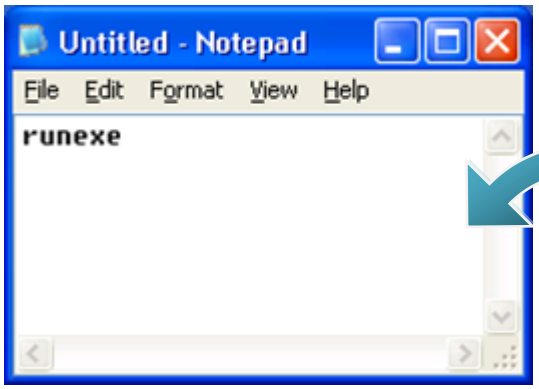
i. 開啟 “Notepad”

ii. 輸入命令
此命令可以是檔案名稱 “hello.exe”（執行特定的檔案）或是 “runexe”（執行上次的執行檔）

iii. 儲存檔案為 autoexec.bat。



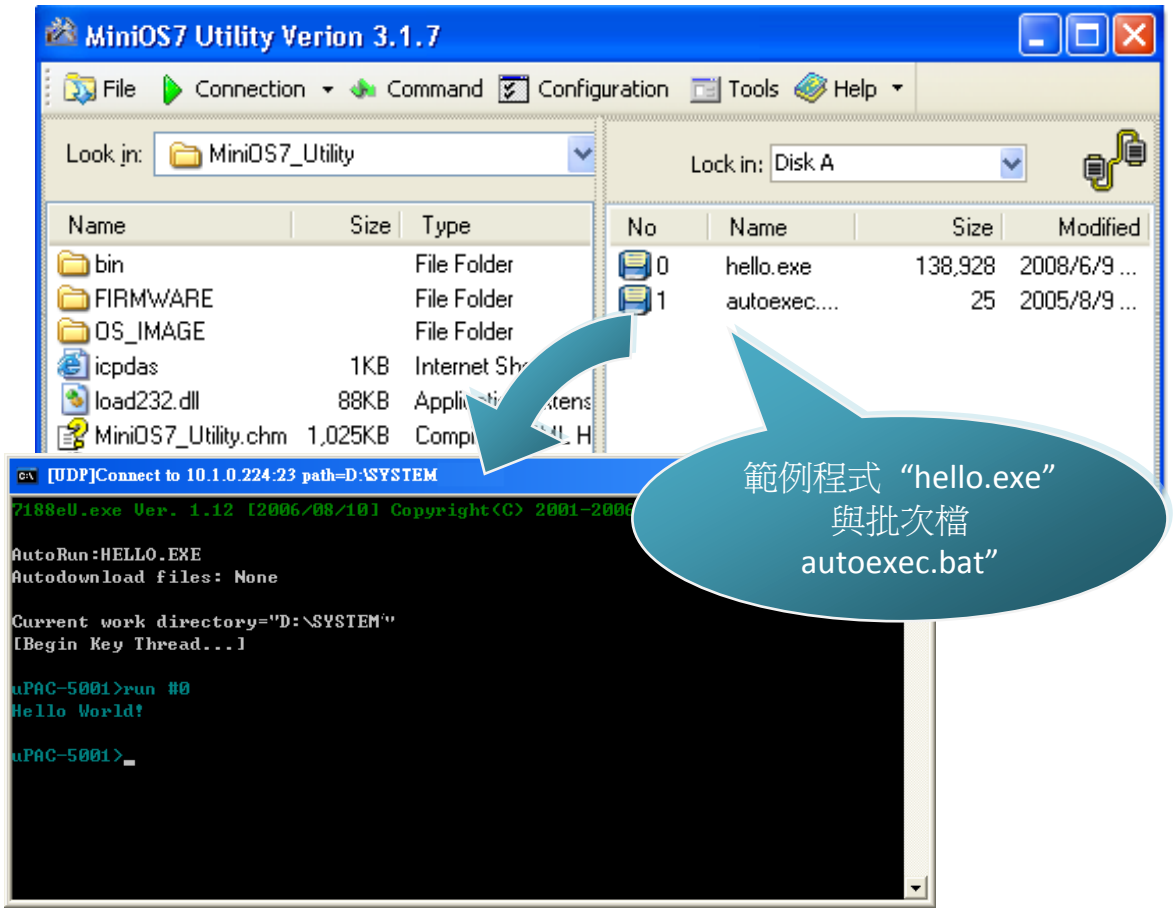
檔案名稱:
執行特定檔案。



Runexe:
執行上次的執行檔。

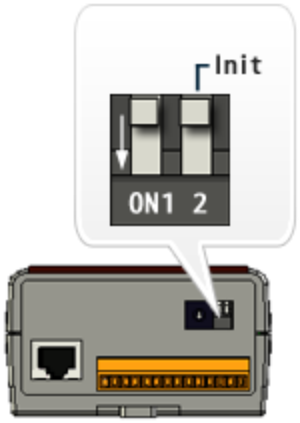
步驟 2: 使用 MiniOS7 Utility 將程式上傳至 µPAC-5001D-CAN2。

請參閱章節 “2.4.1. 建立連線” 以取得詳細資訊。



小技巧 與 安全警告

在重新啟動讓設定生效前，您必須將 Init 開關切換至 “OFF” 。



2.5. 更新 μPAC-5001D-CAN2 的 OS image

泓格科技（ICP DAS）將持續地新增功能於 μPAC-5001D-CAN2 中，建議您定期地參閱我們的網站以取得 μPAC-5001D-CAN2 的最新資訊。

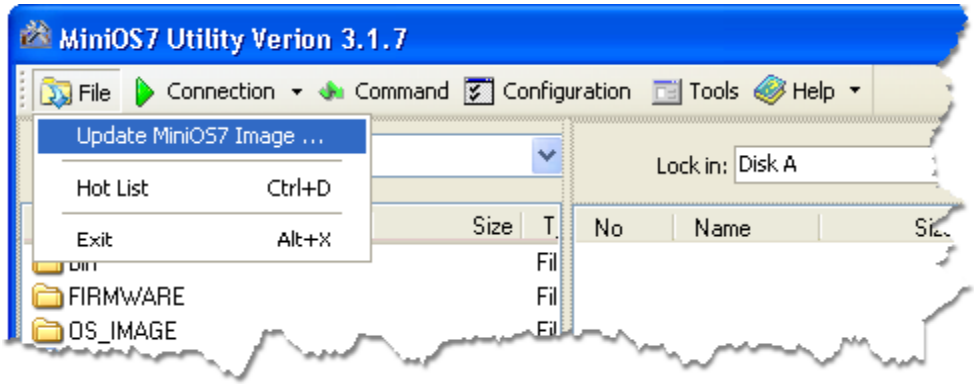
步驟 1: 取得 μPAC-5001D-CAN2 最新版本的 OS image。

可至下列位置，取得 μPAC-5001D-CAN2 最新版本的 OS image:
CD:\NAPDOS\upac-5000\OS_image\
http://ftp.Icpdas.com/pub/cd/8000cd/napdos/upac-5000/os_image/

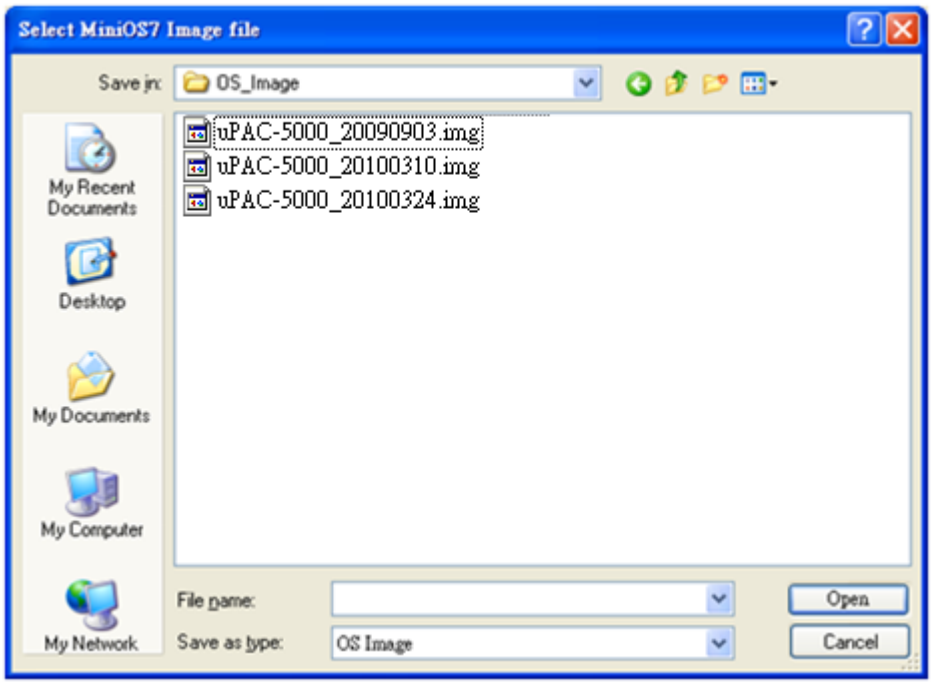
步驟 2: 建立連線。

請參閱章節 “2.4.1. 建立連線” 以取得詳細內容。

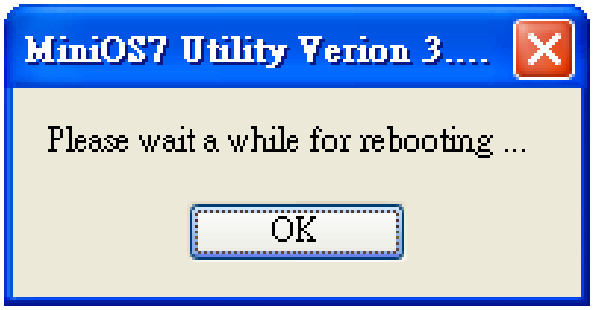
步驟 3: 點選功能表 “File” > “Update MiniOS7 Image …” 。



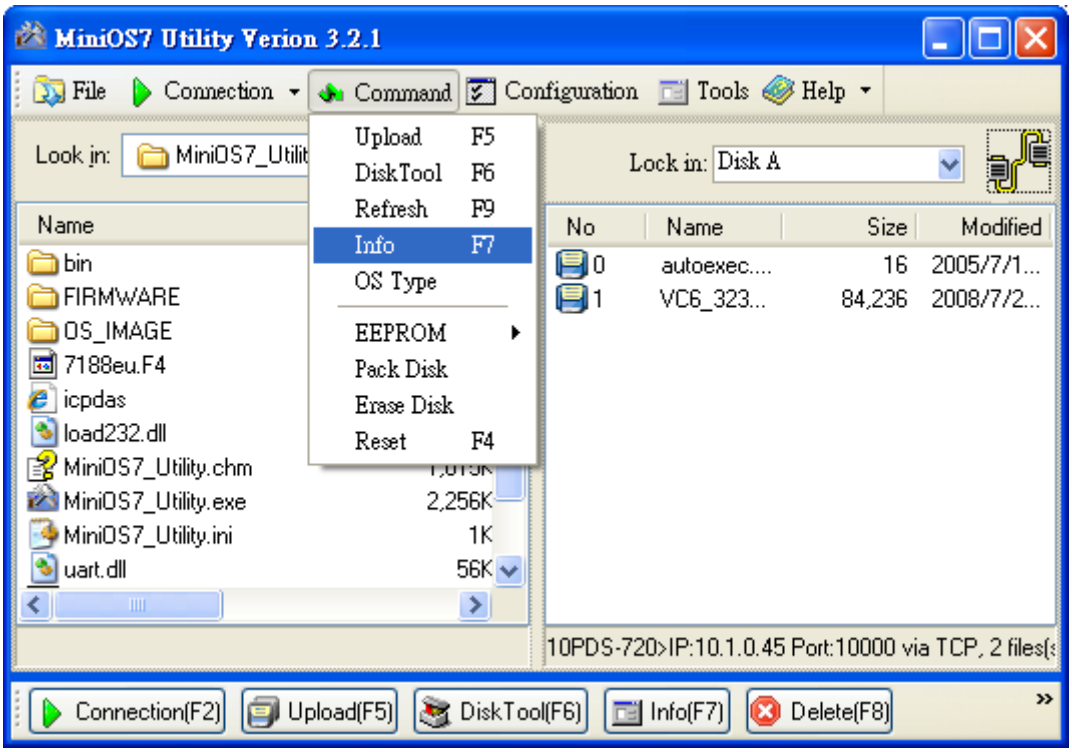
步驟 4: 選取最新版本之 MiniOS7 OS image。



步驟 5: 點選 “OK” 。



步驟 6: 點選功能表 “Command” > “Info” 確認 OS image 版本。



3. 第一個範例程式 - “Hello World”

您可發現在學習每一種電腦編程語言時，首要的範例程式即為 "Hello World"，它簡略地介紹了程式語言的語法與輸出方式。

以下將一步步地引導您，如何編寫第一個 µPAC-5001D-CAN2 程式。

3.1. C 編譯器安裝

C 語言以它的效率聞名，並且是多數人編寫應用時，所使用的編程語言。

在編寫第一個 µPAC-5001D-CAN2 程式前，請確認您的系統中已安裝了必要的 C/C++ 編譯器與相關的函式庫。

下列為應用程式開發服務中常用的 C 編譯器：

- Turbo C++ 版本 1.01
- Turbo C 版本 2.01
- Borland C++ 版本 3.1 - 5.2.x
- MSC
- MSVC ++ 1.52 前版本

建議您使用 Borland C++ 編譯器，如同隨貨光碟中已建立之函式庫的編譯器。

小技巧 與 安全警告

在編譯應用程式之前，請先注意以下事項：

- 建立標準 DOS 可執行程式。
- 設定 CPU 選項為 80188/80186。
- 如需使用浮點數計算，設定浮點數選項為 EMULATION。(請勿選取 8087)
- 取消除錯資訊功能，以減少程式大小。(MiniOS7 支援此功能)

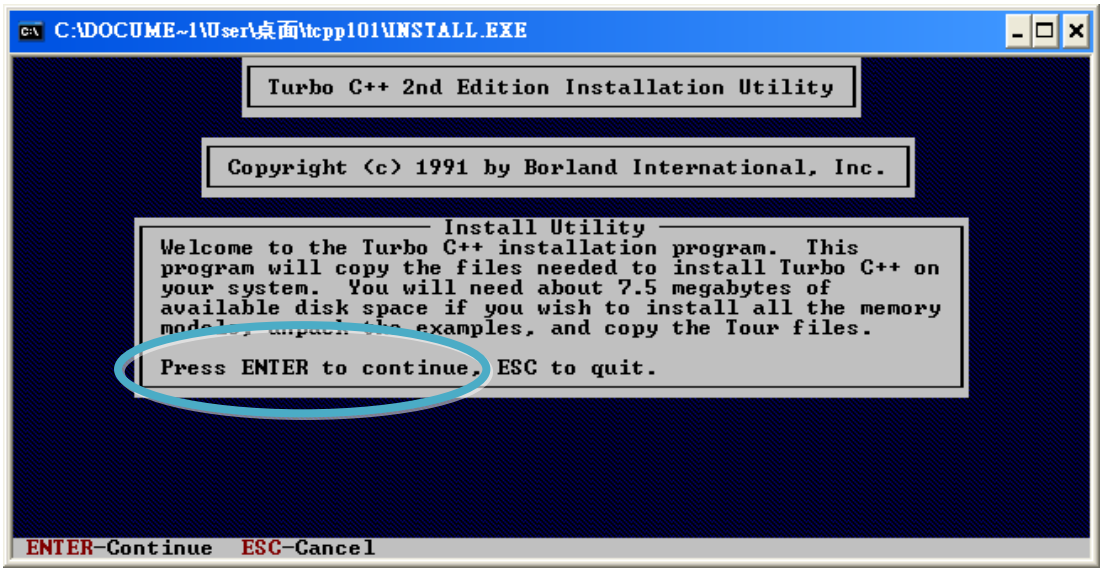
3.1.1. 開始安裝 C 編譯器

若您的系統尚未安裝任何的編譯器，首要步驟即為安裝編譯器。

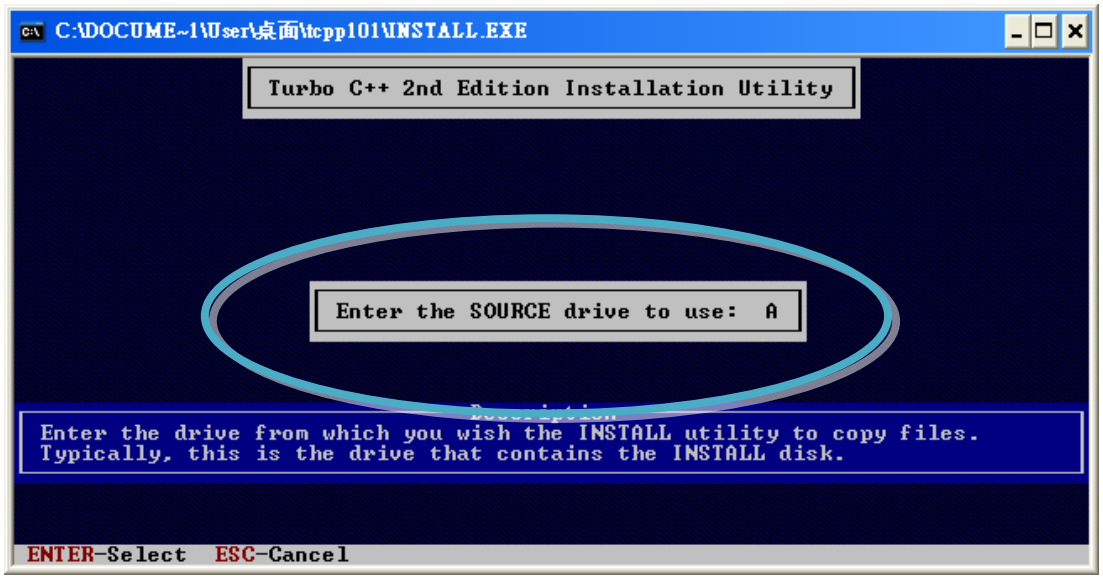
以下將一步步地引導您，於系統中安裝 Turbo C++ 版本 1.01。

步驟 1: 滑鼠雙擊 Turbo C++ 執行檔來啟用安裝精靈。

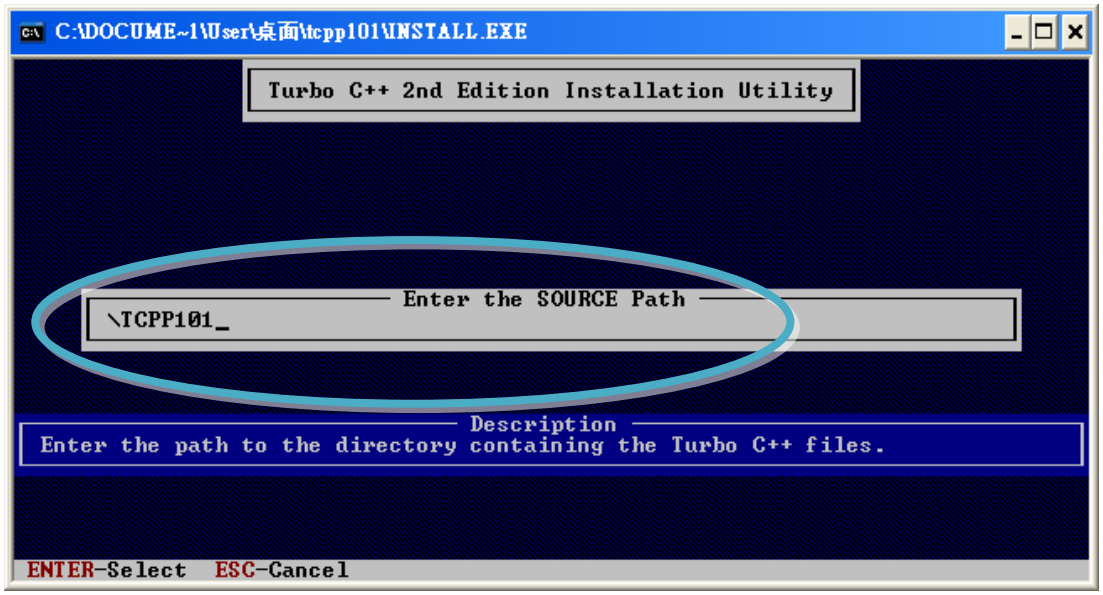
步驟 2: 按 “Enter” 鍵繼續。



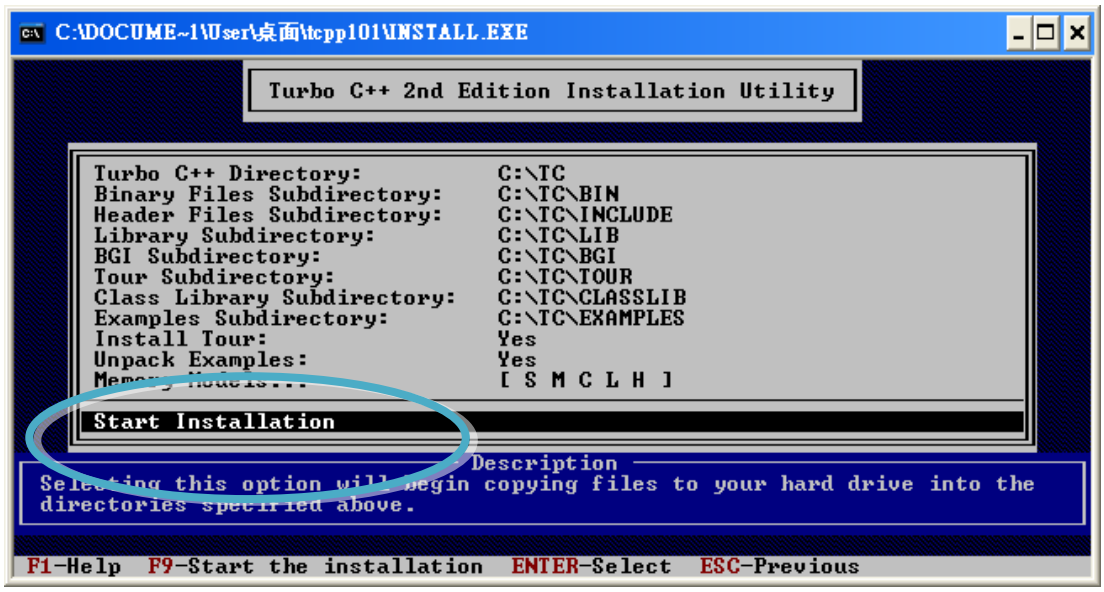
步驟 3: 輸入您欲安裝此軟體的硬碟名稱。(EX. C)



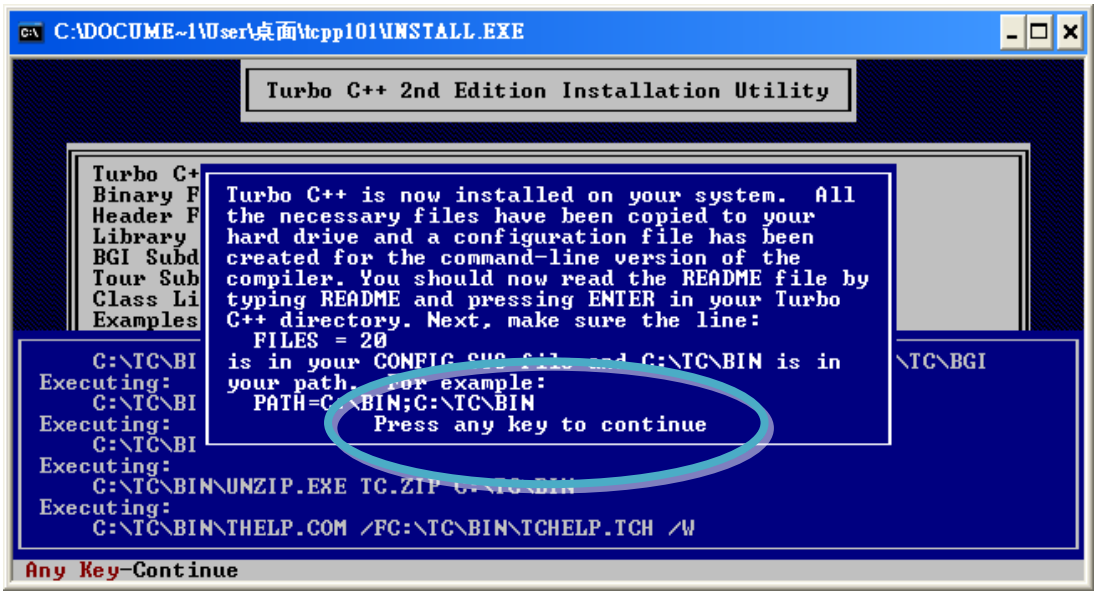
步驟 4: 輸入您欲安裝相關檔案的目錄路徑。(EX. \TC)



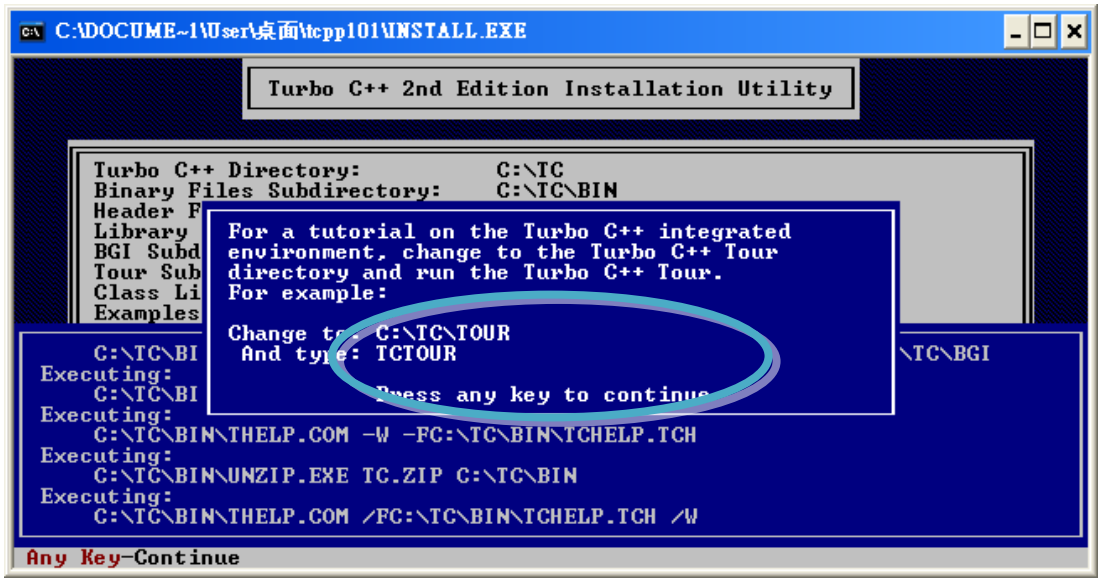
步驟 5: 選取 “Start Installation” 開始安裝程序。



步驟 6: 按任意鍵繼續。



步驟 7: 按任意鍵繼續。

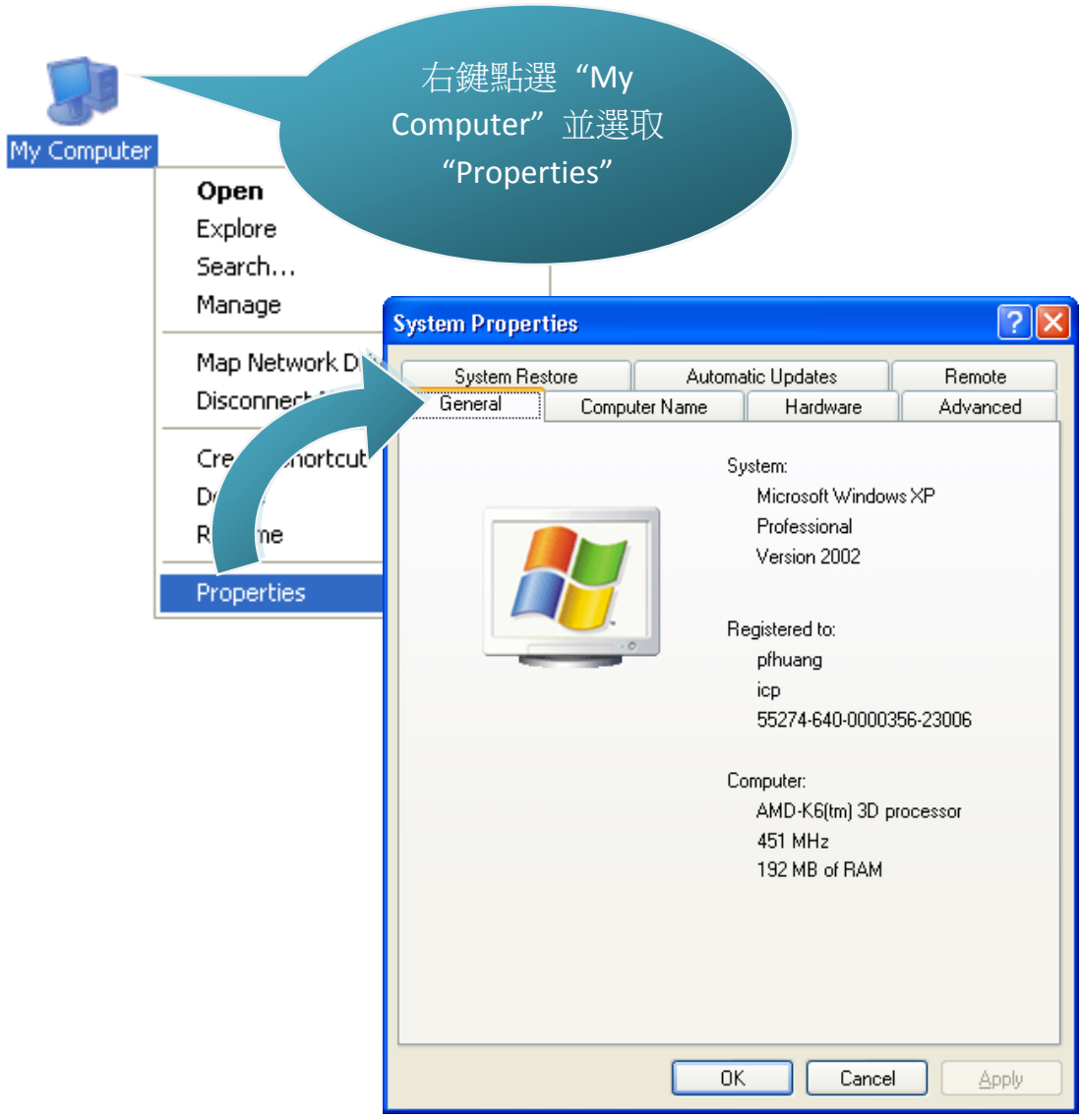


步驟 8: 您已完成安裝。

3.1.2. 設定環境變數

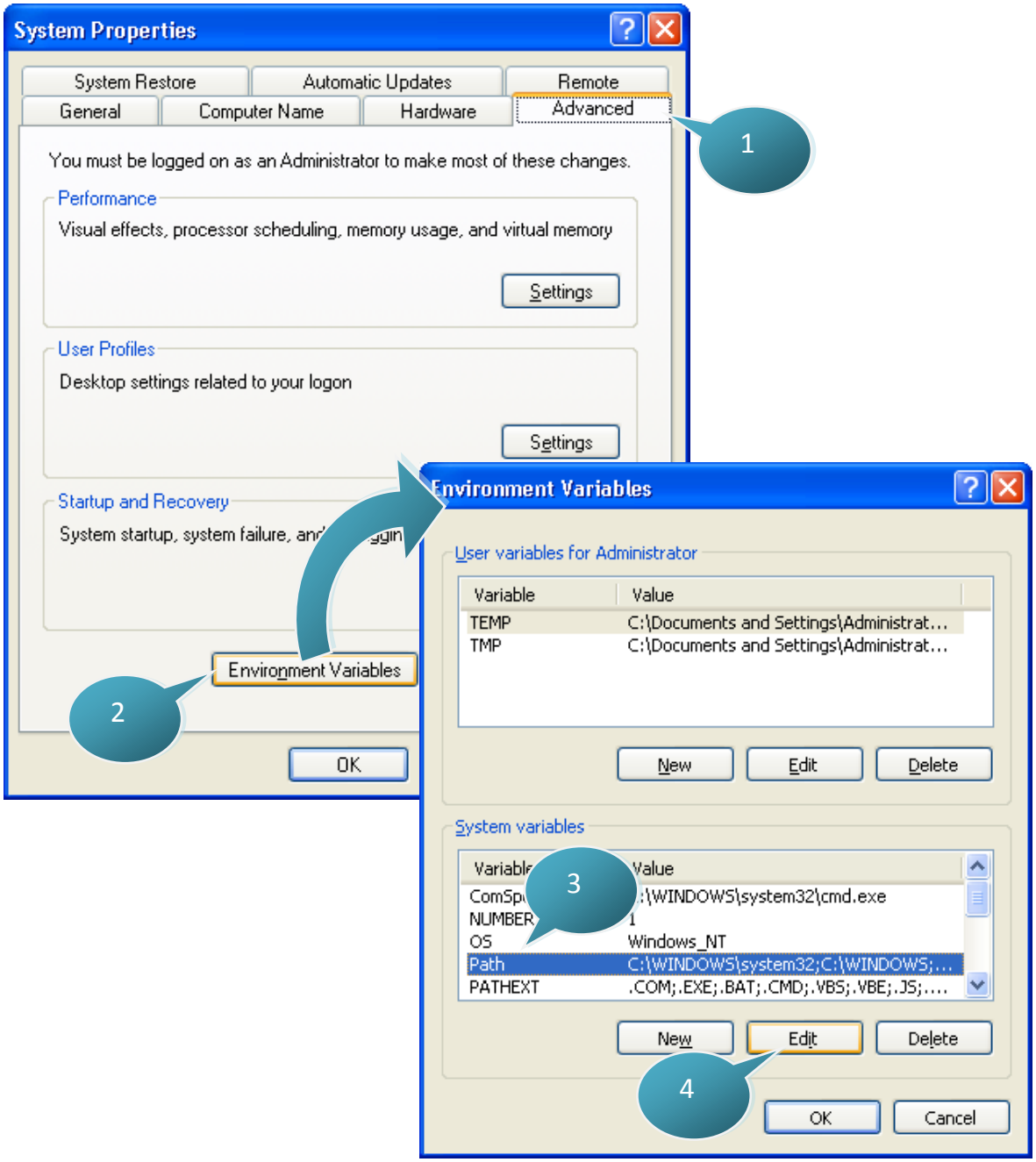
完成編譯器安裝後，在 Windows 命令列中有多種編譯器可用。您可設定環境變數的路徑，以便輸入簡單的名稱即可執行命令列上的編譯器，而不用使用完整的路徑名稱。

步驟 1: 滑鼠右鍵點選桌面圖示 “My Computer” 並選取功能表選項 “Properties” 。



步驟 2: (“System Properties” 視窗)
點選 “Advanced” 頁籤中的 “Environment Variables” 按鈕。

步驟 3: (“Environment Variables” 視窗)
點選 “System variables” 項目中的 “Path” 再點選 “Edit” 按鈕。



步驟 4: 在 “Variable Value” 欄位後面加上目標路徑。

分號（;）用來表示變數值之間的分隔符號。例如，”;c:\TC\BIN\;c:\TC\INCLUDE\”

步驟 5: 重新啟動電腦，讓您的變更生效。

3.2. µPAC-5001D-CAN2 之應用程式介面（API）

以下之 API 包含:標準 API 與 CAN bus API，可提供使用者自訂一些標準功能並可與其它應用程式、設備和服務相整合。

請參閱下列位置的更新紀錄檔案，取得 µPAC-5000 系列 標準 API 之詳細資訊：
CD:\NAPDOS\upac-5000\Demo\basic\Lib\
http://ftp.icpdas.com/pub/cd/8000cd/napdos/upac-5000/demo/basic/lib/

請參閱下列位置，取得 µPAC-5001D-CAN2 CAN bus API 之詳細資訊：
CD:\fieldbus_cd\can\upac-5001D-CAN\demo\bc_tc\lib\
ftp://ftp.icpdas.com/pub/cd/fieldbus_cd/can/pac/upac-5001D-CAN/demo/bc_tc/lib/

建立應用程式之前，請確認您已完成軟體安裝。若您尚未安裝，請參閱章節 “2.2. 軟體安裝” 。

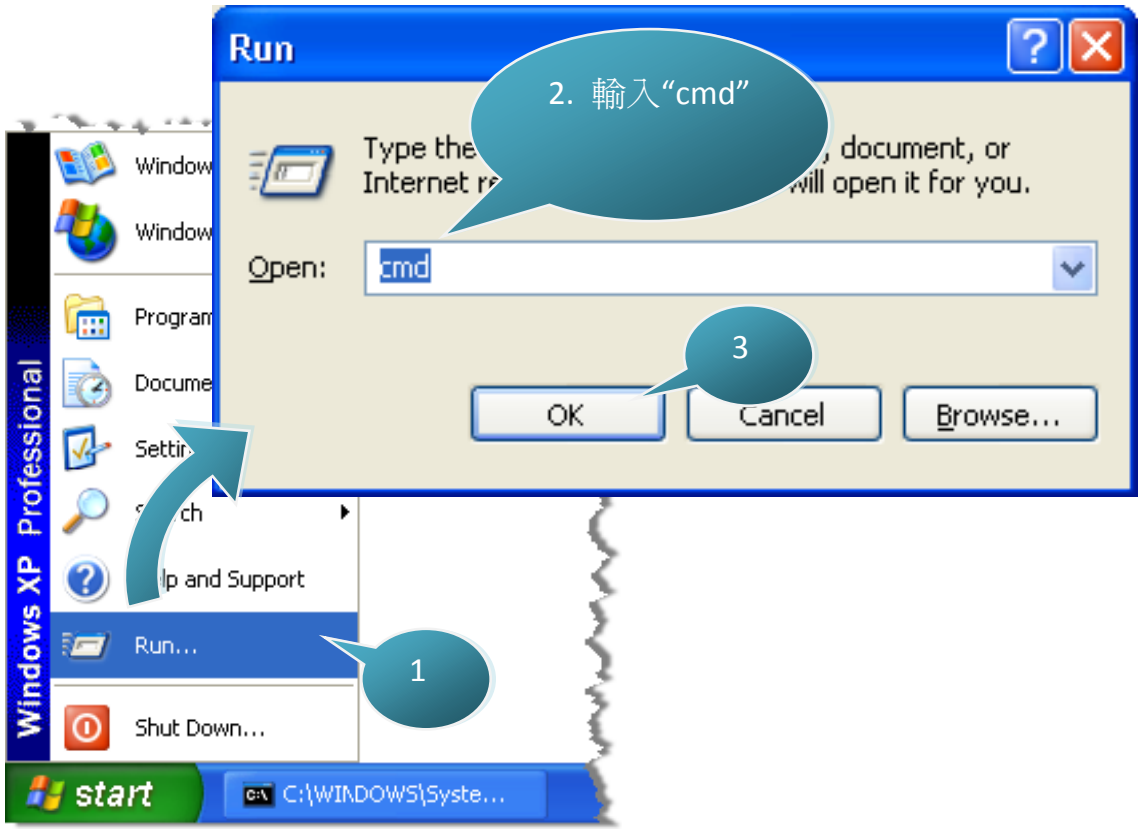
3.3. µPAC-5001D-CAN2 中的第一個程式

此章節，我們假設您已在 C 硬碟區的根目錄下，安裝了 Turbo C++ 1.01（如章節 “3.1. C 編譯器安裝”）與 µPAC-5001-CAN2 的 API（如章節 “2.2. 軟體安裝”）。

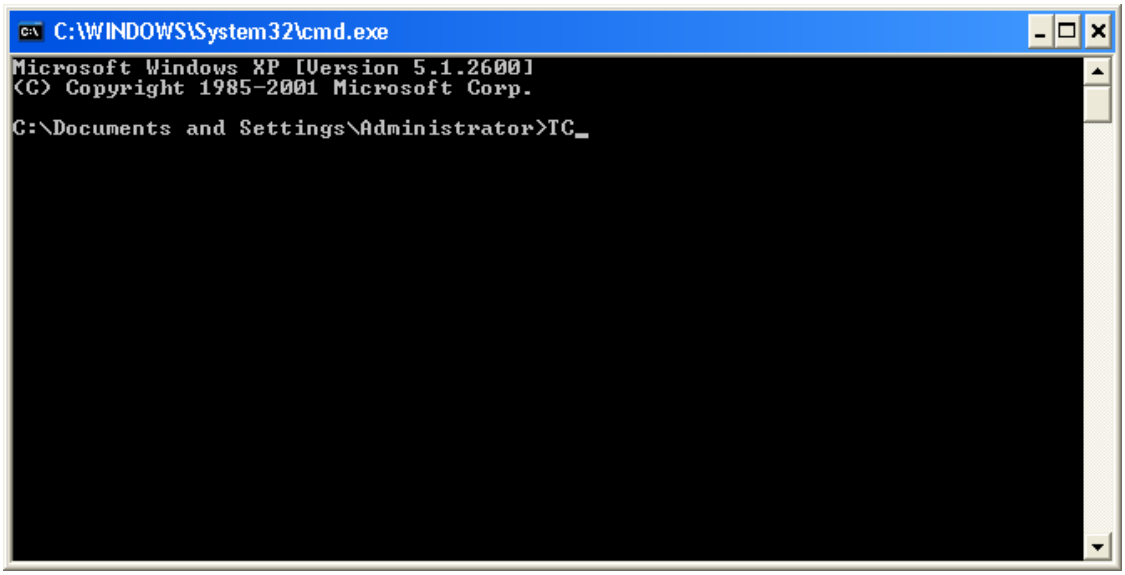
以下將一步步地引導您，編寫第一個程式。

> 步驟 1: 開啟 MS-DOS 的命令提示字元。

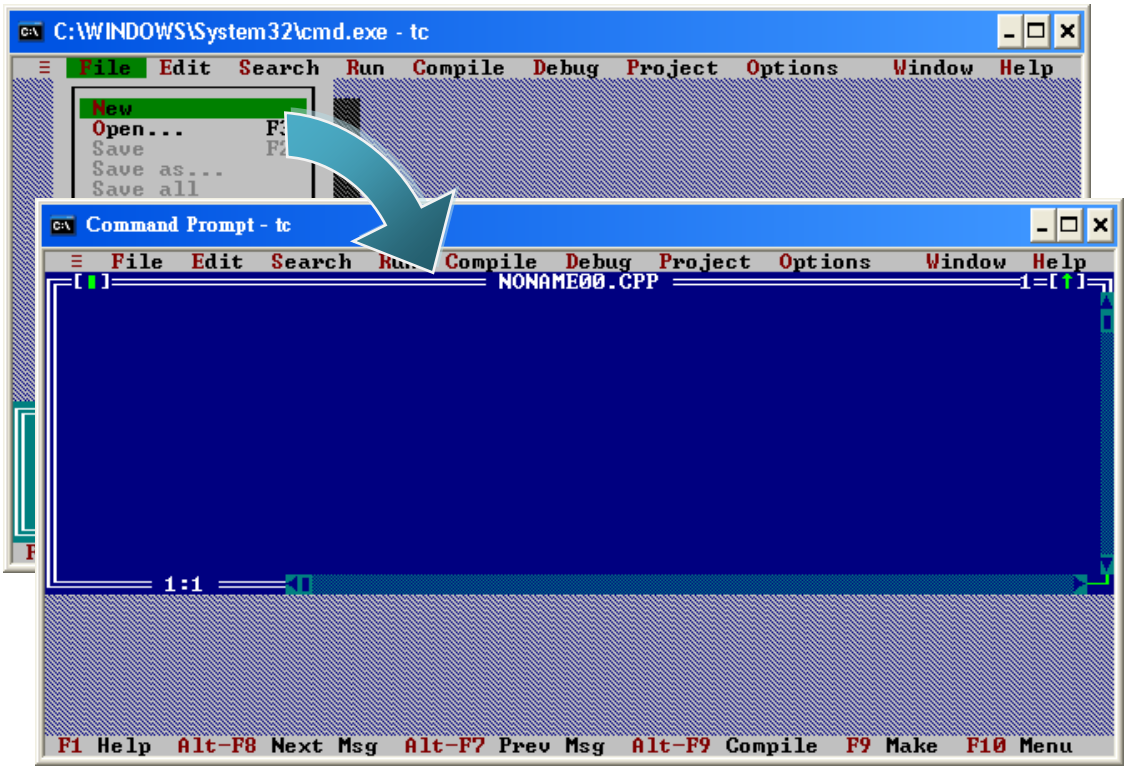
i. “Start” 選單中，點選 “Run”。
ii. “Run” 視窗中，輸入 “cmd”。
iii. 點選 “OK” 按鈕。



步驟 2:（命令提示字元視窗）輸入“TC” 並按 “Enter” 鍵。



步驟 3: 點選功能表 “File” > “New”，新增一個原始檔。

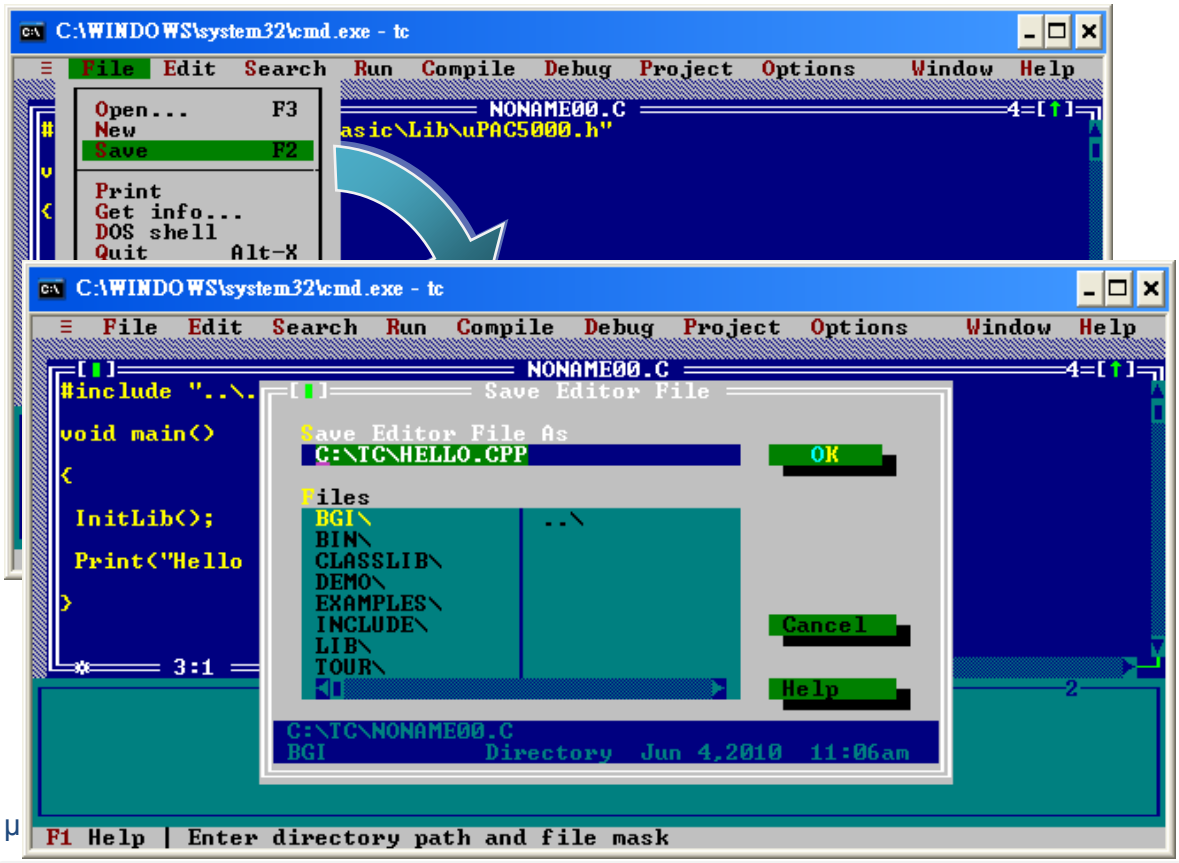


步驟 4: 輸入以下程式碼。（注意：程式碼有區分大小寫）

```
#include  "..\..\Demo\basic\Lib\uPAC5000.h"
/* 引入標題檔以允許使用 uPAC5000.lib 函式 */

void main(void)
{
 InitLib();    /* 初始化 uPAC5000 函式庫 */
 Print( "Hello world!\r\n" );    /* 螢幕上印出訊息 */
}
```

步驟 5: 儲存原始檔。

i. 點選功能表 “File” > “Save” 。
ii. 輸入檔名 “HELLO” 。
iii. 點選 “OK” 。

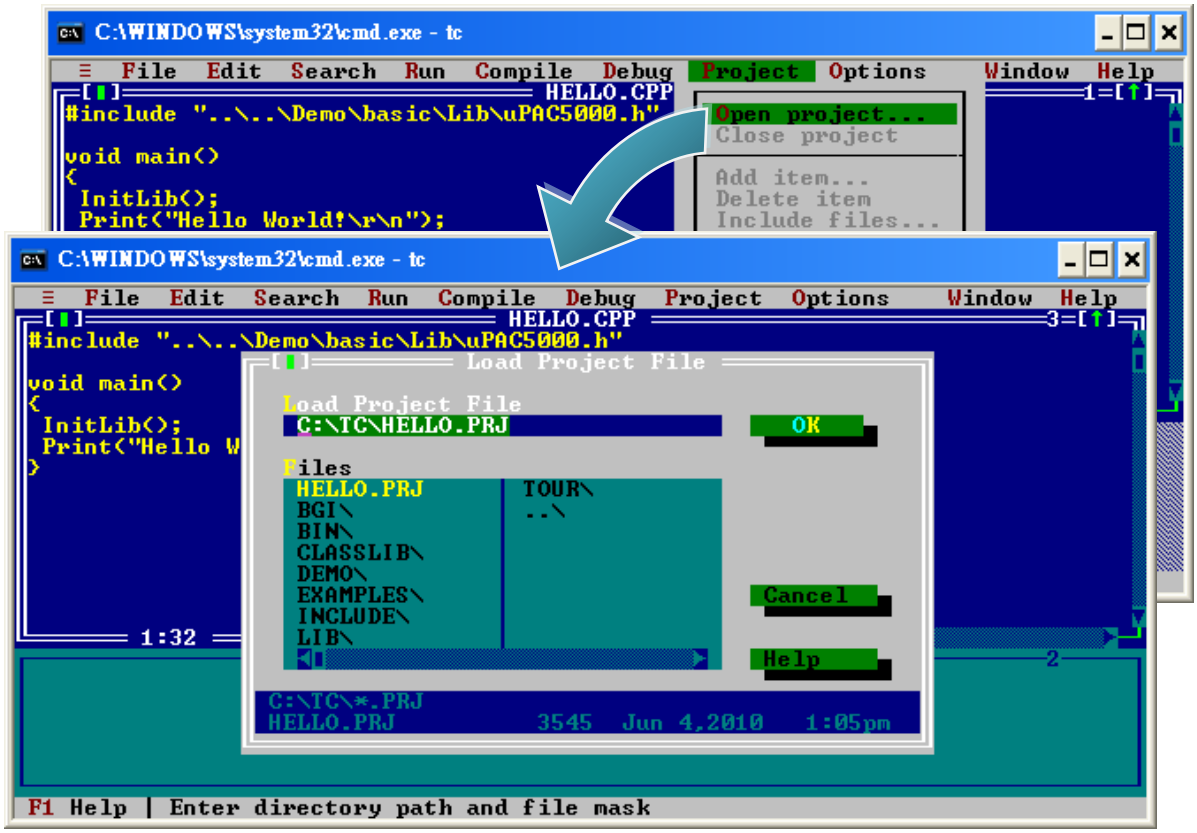


小技巧 與 安全警告

您可使用熟悉的文字編輯器或其他工具來編寫以下程式碼，但您必須將原始檔儲存為 .C 的副檔名。

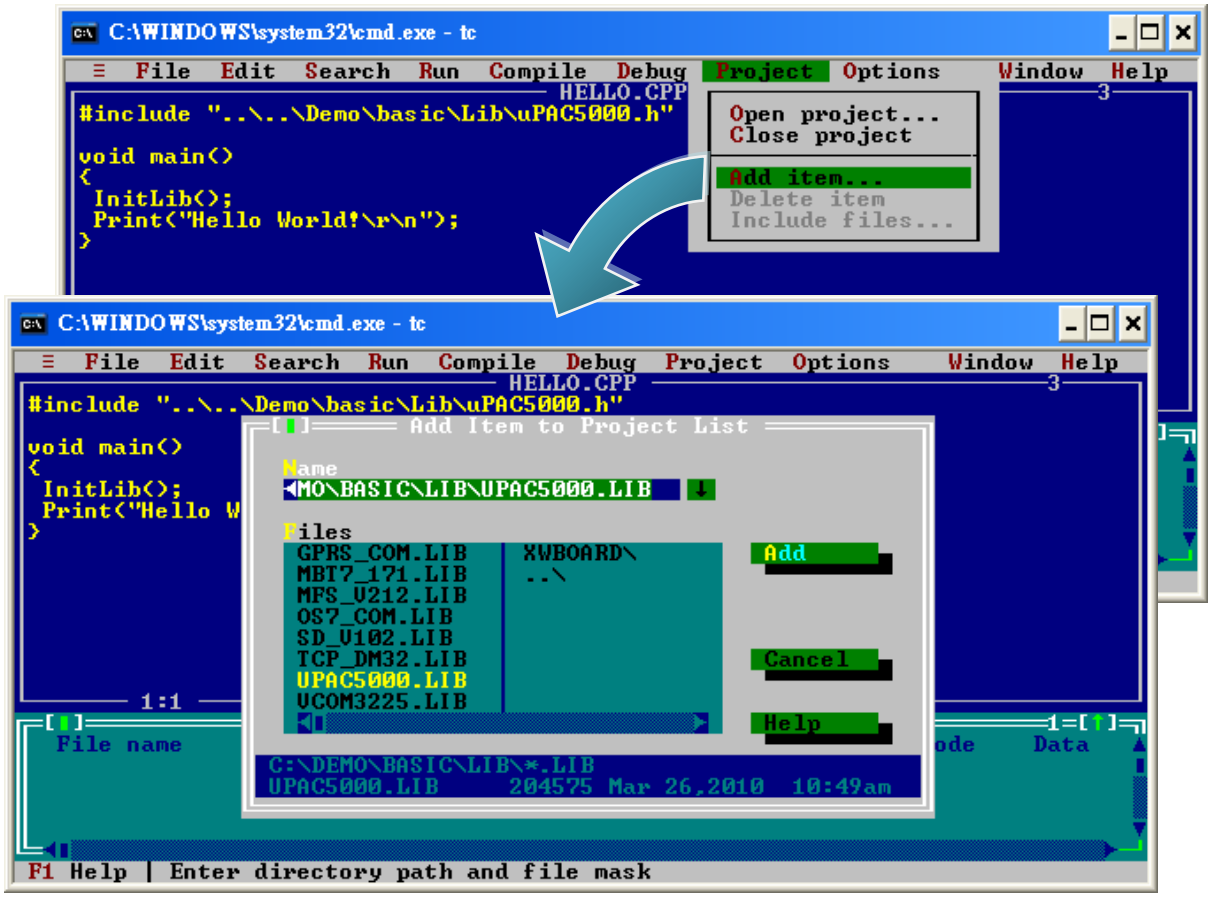
步驟 6: 新建專案 (*.prj)

i. 點選功能表 “Project” > “Open project…” 。
ii. 輸入專案名稱 “HELLO” 。
iii. 點選 “OK” 。



步驟 7: 加入需要的函式庫 (*.lib) 於專案中。

i. 點選功能表 “Project” > “ Add item…” 。
ii. 輸入 “ *.LIB ” 以顯示所有可用的函式庫。
iii. 選取您所需要的函式庫。
iv. 點選 “Add” 加入。
v. 點選 “Done” 完成設定。

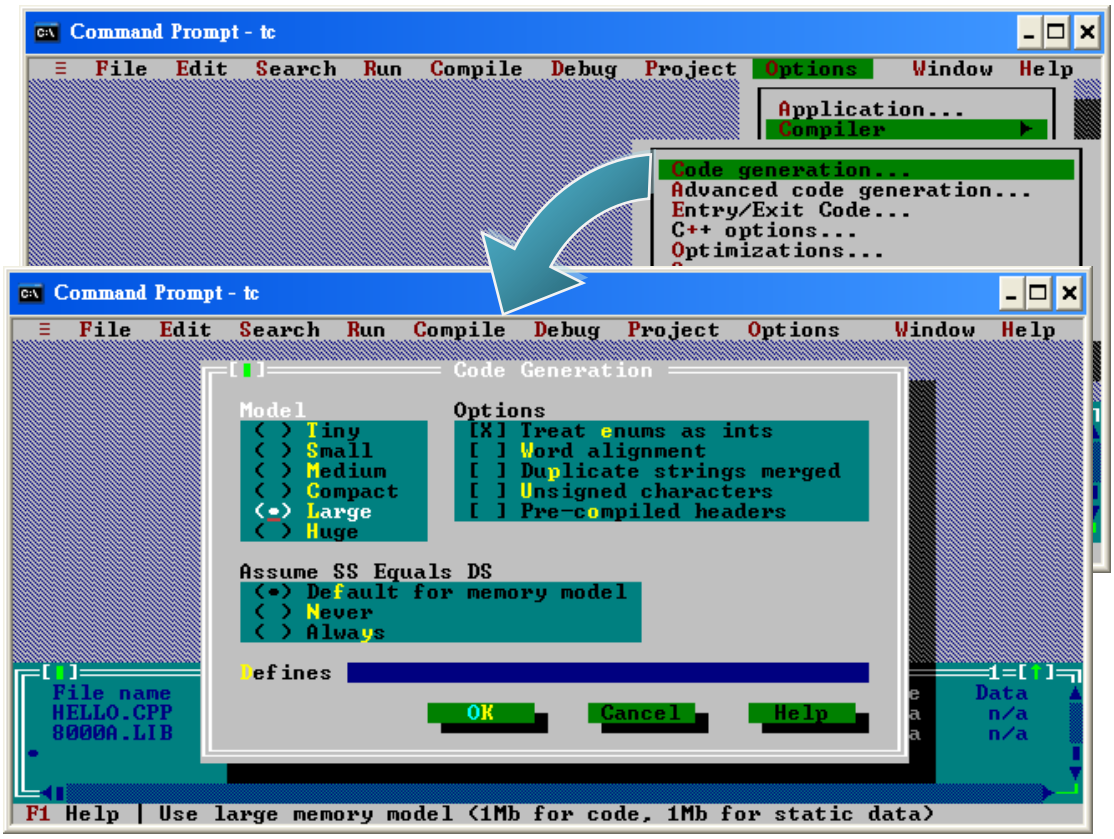


步驟 9:在專案中加入 C 語言的原始碼 (*.c)

i. 點選功能表 “Project” > “ Add item…” 。
ii. 輸入 “ *.C ” 以顯示所有可用的原始碼。
iii. 選取您所需要的原始碼。
iv. 點選 “Add” 加入。
v. 點選 “Done” 完成設定。

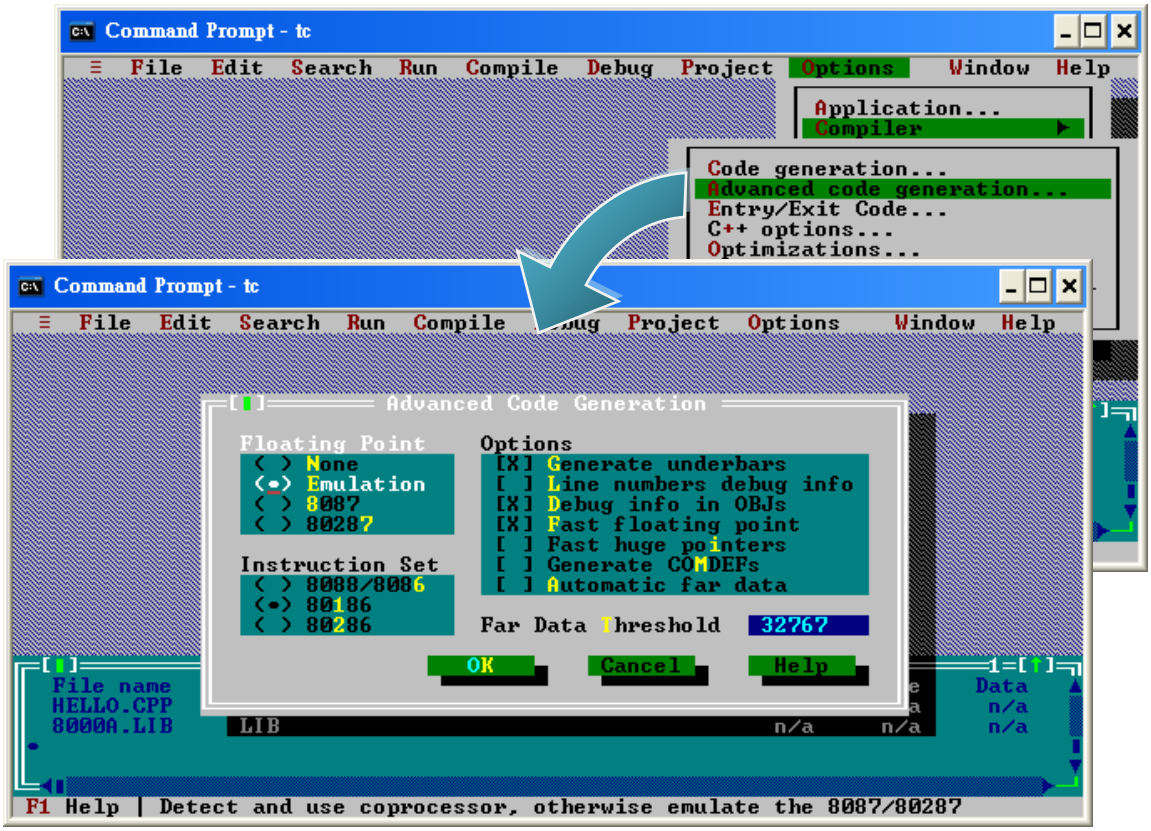
步驟 9: 設定記憶體模式為 “ Large” 。

i. 點選功能表 “Options” > “Compiler” > “Code generation…” 。
ii. 於 “Model” 項目，選取 “Large” 。
iii. 點選 “OK” 。



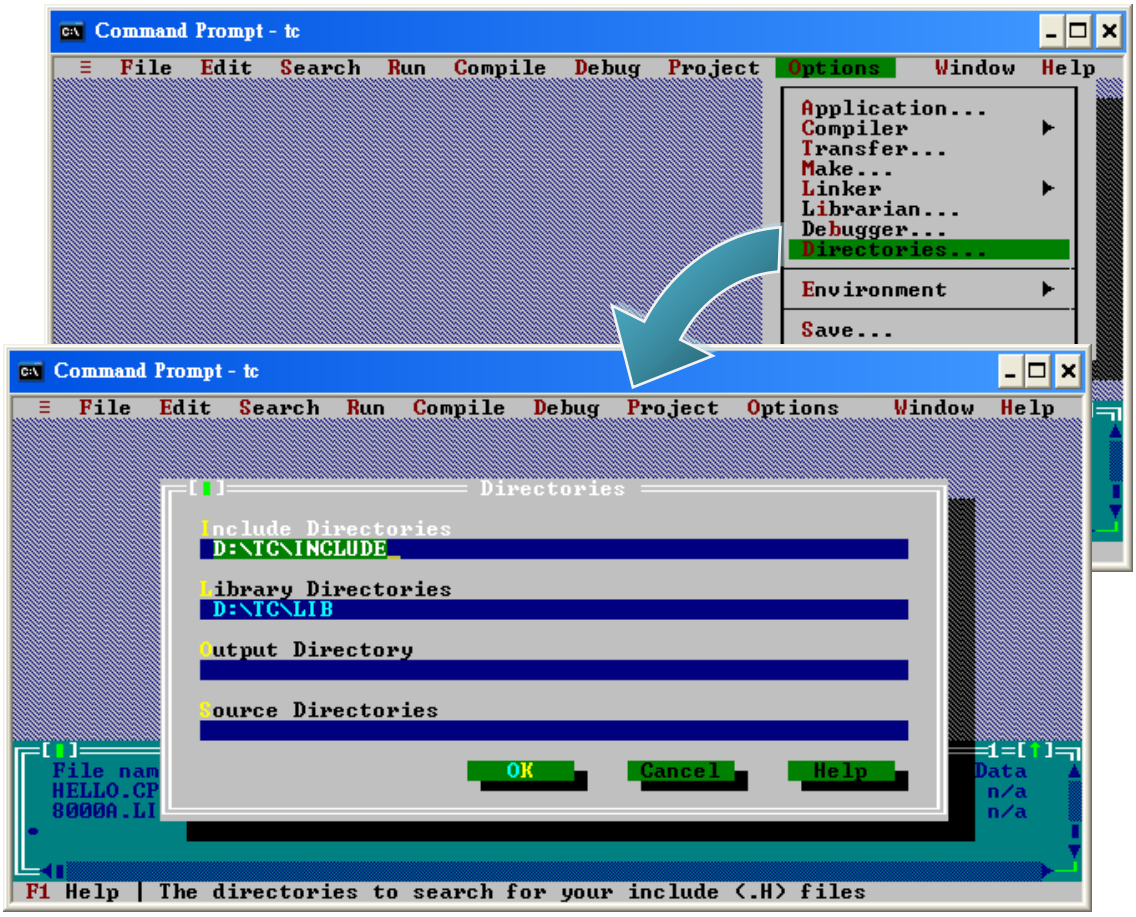
步驟 10: 設定浮點數為 “Emulation”，指令集為 “80186”。

i. 點選功能表 “Options” > “Compiler” > “Advanced code generation…”。
ii. 於 “Floating Point” 項目，選取 “Emulation”。
iii. 於 “Instruction Set” 項目，選取 “80186”。
iv. 點選 “OK”。

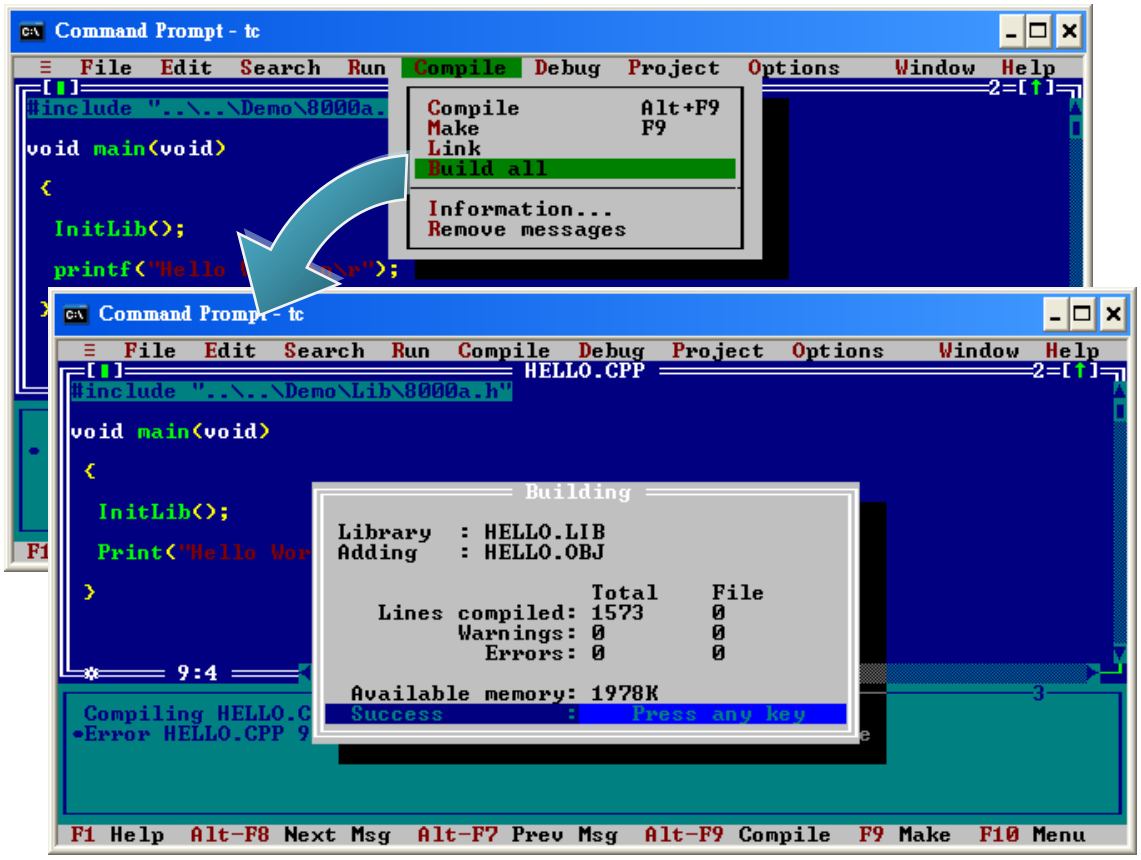


步驟 11: 設定 TC 編譯器的引入檔（Include）與函式庫之目錄。

i. 點選功能表 “Options” > “Directories…” 。
ii. 於 “Include Directories” 項目，設定標頭檔之存放目錄。
iii. 於 “Library Directories” 項目，設定函式庫之存放目錄。
iv. 點選 “OK” 。

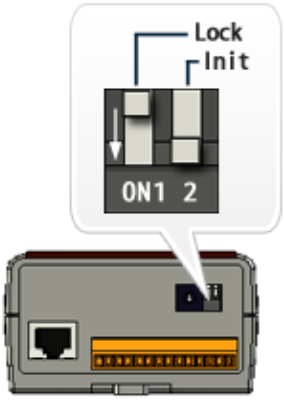


步驟 12: 點選功能表 “Compile” > “Build all” 以建置專案。

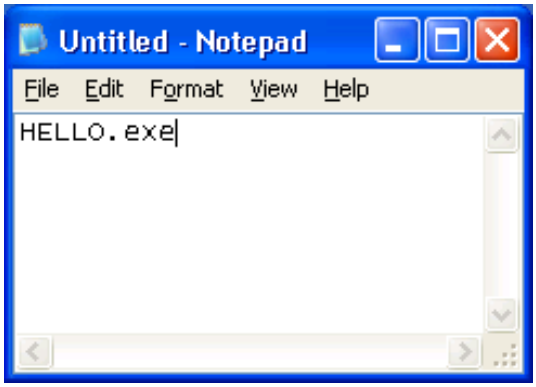


步驟 13: 設定工作模式。

請確認 Lock 開關已經切換至 “OFF” ，且 Init 已切換至 “ON” 。



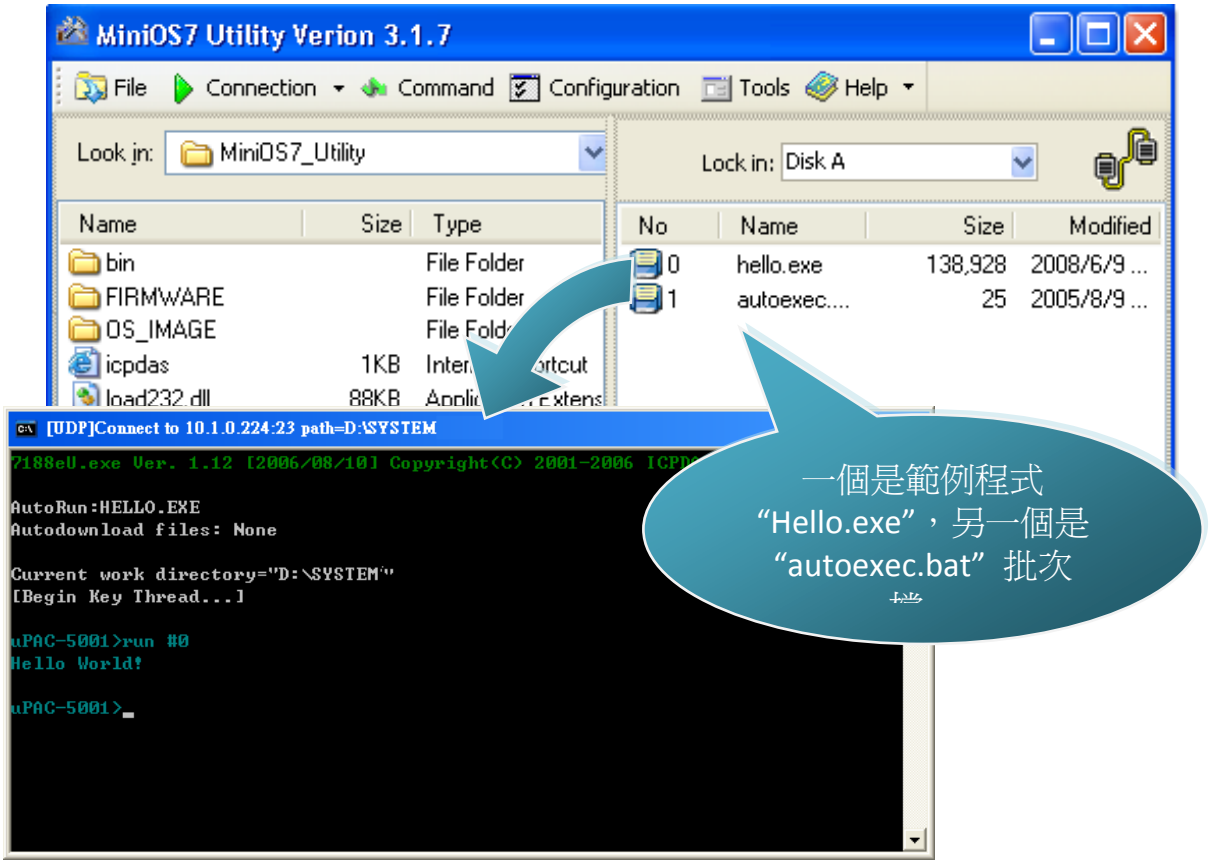
步驟 14: 建立自動執行檔 （autoexec.bat）。



i. 開啟 “Notepad”。
ii. 輸入 “HELLO.exe”。
iii. 將檔案儲存為 “autoexec.bat”。

步驟 15: 使用 MiniOS7 Utility 將程式上傳至 µPAC-5001-CAN2。

請參閱章節 “2.4.1. 建立連線”，取得此設定之詳細資訊。



4. API 與範例程式參考

下列是專為 µPAC-5001D-CAN2 API 所設計的範例程式。 您可檢視這些 API 與範例程式的原始碼，裡面含有許多的函式與註解，可讓您熟悉 MiniOS7 的 API 並可藉由修改範例程式快速地開發自己的應用程式。

下表為依功能分類的 API：

API 說明	標頭檔	函式庫
CPU Driver	uPAC5000.h	uPAC5000.lib
新版本之 COM Port Driver	OS_COM.h	OS_COM.lib
Ethernet Driver	TCPIP32.h	TCP_DM32.Lib
microSD Driver	microSD.h	sd_V102.lib
Xserver	VxComm.h	Vcom3225.Lib
Modbus Driver	MBTCP.h	MBT7_171.lib
Xboard Driver for CAN BUS	XWCAN.h	XWCAN.lib

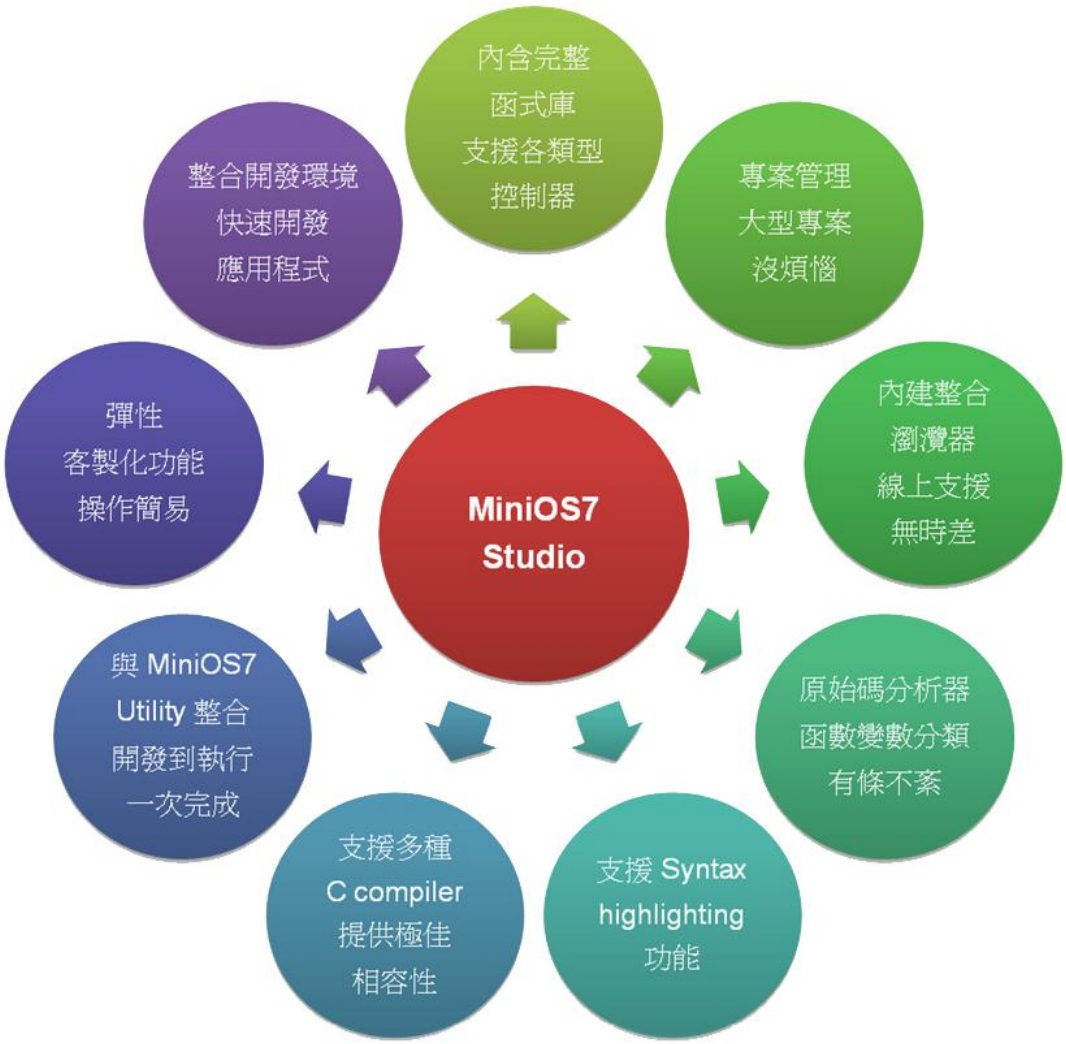
請參閱下列位置，以取得更多關於 µPAC-5000 API 的詳細資訊：
CD:\NAPDOS\upac-5000\Demo\basic\Lib\
http://ftp.icpdas.com/pub/cd/8000cd/napdos/upac-5000/demo/basic/lib/

請參閱下列位置，以取得更多關於 µPAC-5001D-CAN2 API 的詳細資訊：
CD:\fieldbus_cd\can\pac\upac-5001D-CAN\demo\bc_tc\lib\
ftp://ftp.icpdas.com/pub/cd/fieldbus_cd/can/pac/upac-5001D-CAN/demo/bc_tc/lib/

以下將介紹核心的 API – 已整合至 μPAC-5000 API Set 之 MiniOS7 API。

函式庫 — uPAC5000.lib
此檔案包含了 MiniOS7 的應用程式介面（Application Programming Interface）與數百個和 μPAC-5000 相關的預定義函式。

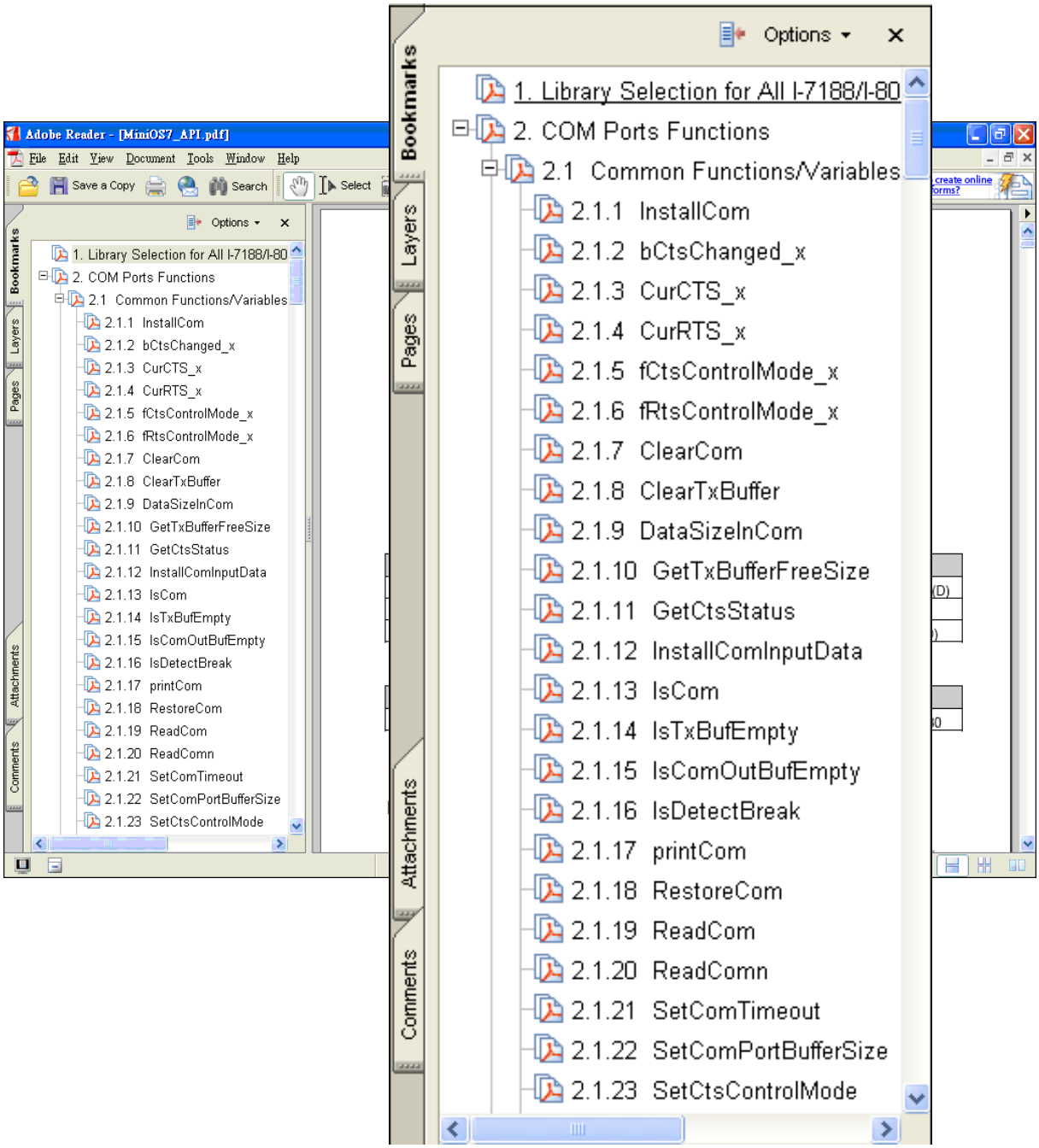
標頭檔 — uPAC5000.h
此檔案包含了用於 MiniOS7 API 之副程式、變數與其它識別符號的前置宣告。



請參閱下列位置的 “MiniOS7 的 API 函式使用手冊” ，以取得關於函式的說明、原型（prototype）與引數的完整使用資訊：

CD:\Napdos\MiniOS7\Document

http://ftp.Icpdas.com/pub/cd/8000cd/napdos/minios7/document/



下表為 µPAC-5001D 系列依功能分類的範例程式：

資料夾	範例程式	說明
LED	LED	示範如何控制 LED 顯示器。
MISC	Seg7LED	示範如何控制五位數七段 LED 顯示器。
	Rotary_Switch	示範如何讀取開關的位置。
Memory	EEPROM	示範如何讀寫 EEPROM。
	Flash	示範如何存取 flash。
	Nvram-r, Nvram-w	示範如何讀寫 nvram。
microSD	sd_qa	示範如何連接並控制 microSD。
	sd_read	
	sd_util	
	sd_write	
Timer	Demo90, demo91, …, demo99	示範如何使用時間函式。
7k87k_Module	7k87k_ai_for_com, 7k87k_demo_for_com, … Ao_24_for_com	示範如何控制 7K 與 87K 模組。

請參閱下列位置，取得更多關於 µPAC-5000 API 之詳細資訊：
CD:\NAPDOS\upac-5000\Demo\basic\
http://ftp.Icpdas.com/pub/cd/8000cd/napdos/upac-5000/demo/basic/

下表為 µPAC-5001D-CAN 系列的範例程式:

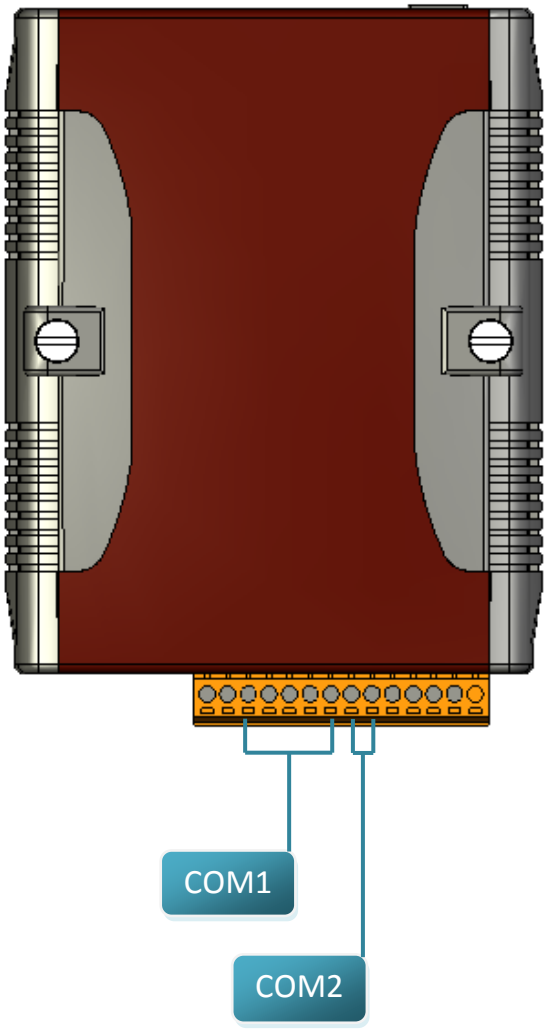
資料夾	範例程式	說明
Demo	AC_AM	使用接受碼 AccCode 與接受遮罩 AccMask。
	Dual_Filter	使用接收碼 AccCode 與接受遮罩 AccMask 進行 Dual filter 功能。
	All_Demo	藉由 XWCAN.lib 提供所有函數的使用範例。
	L1_L2_L3	使用 CL1、 CL2、及 CL3 LED 顯示器。
	RxInt	藉由中斷模式接收 CAN 信息。
	RxPoll	藉由輪循模式接收 CAN 信息。
	TxInt	藉由中斷模式發送 CAN 信息到 CAN 網路。
	TxPoll	藉由輪循模式發送 CAN 信息到 CAN 網路。
	UserInt	利用 “UserCANInt” 函式來實施使用者的 CAN 中斷服務程序。
	SCH_Baud	搜尋 CAN bus 鮑率的範例。

請參閱下列位置，取得更多關於 µPAC-5001D-CAN2 CAN bus API 之詳細資訊：
CD:\fieldbus_cd\can\pac\upac-5001D-CAN\demo\bc_tc\lib\
ftp://ftp.icpdas.com/pub/cd/fieldbus_cd/can/pac/upac-5001D-CAN/demo/bc_tc/lib/

4.1. 用於 COM Port 的 API

μPAC-5001D-CAN2 內建有 2 個通訊埠 – 即 COM1 與 COM2。

- COM1 – 為 RS-232 埠，可用來連接 PC。
- COM2 – 為 RS-485 埠，可用在點對點連接。



4.1.1. COM Port 函式的類型

以下為兩種適用於 COM Port 函式的類型：

1. MiniOS7 的 COM Port 函式
2. (C style) 標準 COM Port 函式

小技巧 與 安全警告

(C style) 標準 COM Port 函式只能在 COM1 使用。若需使用 COM1，您必須在 MiniOS7 COM Port 函式或 (C style) 標準 COM Port 函式中選擇其一。選擇其中一項，另一項則無法使用。

MiniOS7 的 COM Port 函式與 (C style) 標準 COM Port 函式之比較結果：

函式類型	COM Port	Buffer	函式				
MiniOS7 COM Port	1、2,.. 等	1 KB	1 KB	IsCom()	ToCom()	ReadCom()	printCom()
(C style) 標準 COM Port	1	512 Bytes	256 Bytes	Kbhit()	Puts() Putch()	Getch()	Print()

4.1.2. MiniOS7 COM Port 之 API

COM Port 之 API

1. InstallCom()

使用 COM Port 之前，必須先呼叫 InstallCom() 函式來初使化 COM Port。

2. RestoreCom()

若程式呼叫了 InstallCom()，最後需再呼叫 RestoreCom() 函式來釋放 COM Port 的佔用。

檢查 COM Port 輸入暫存器內是否有資料之 API

3. IsCom()

從 COM Port 讀取資料之前，必須呼叫 IsCom() 函式來檢查目前 COM Port 輸入暫存器內是否有資料。

從 COM Port 讀取資料之 API

4. ReadCom()

在 IsCom() 函式確認輸入暫存器內有資料後，必須呼叫 ReadCom() 函式去讀取 COM Port 輸入暫存器內的資料。

傳送資料給 COM Port 之 API

5. ToCom()

在傳送資料 COM Port 之前，必須呼叫 ToCom() 函式來傳送資料至 COM Port。

範例 - 透過 COM1 來讀取與接收資料。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   int quit=0, data;

   InitLib(); /* Initiate the upac5000 library */
   InstallCom(1, 115200L, 8, 0, 1); /* Install the COM1 driver */

   while(!quit)
   {
      if(IsCom(1)) /* Check if there is any data in the COM port input buffer */
      {
         data=ReadCom(1); /* Read data from COM1 port */
         ToCom(1, data);    /* Send data via COM1 port */
         if(data=='q') quit=1; /* If 'q' is received, exit the program */
      }
   }

   RestoreCom(1); /* Release the COM1 */
}
```

6. printCom()

C 函式庫中，像是 printfCom() 函式可允許由 COM Port 輸出資料。

範例 一 由 COM1 Port 顯示資料。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   int i;

   InitLib(); /* Initiate the upac5000 library */

   InstallCom(1, 115200L, 8, 0, 1); /* Install the COM1 driver */
   for(i=0; i<10; i++)
   {
      printCom(1, "Test %d\r\n", i);
   }

   Delay(10); /* Wait for all data are transmitted to COM port */
   RestoreCom(1);
}
```

4.1.3. 標準 COM Port 之 API

標準 COM Port 用來由 PC 上傳程式到 µPAC-5001D-CAN2。

小技巧 與 安全警告

(C style) 標準 COM Port 函式只能用在 COM1，以下為固定的 COM1 配置：

鮑率 = 115200 bps，資料格式 = 8 位元

同位元檢查 = none，起始位元 = 1，停止位元 = 1

檢查輸入暫存器內是否有資料之 API

1. Kbhit()

由標準 I/O Port 讀取資料前，必須呼叫 Kbhit() 函式來確認目前輸入暫存器中是否有資料。

從標準 I/O Port 讀取資料之 API

2. Getch()

在 Kbhit() 函式確認輸入暫存器有資料後，需呼叫 Getch() 函式去讀取輸入暫存器內的資料。

傳送資料至標準 I/O Port 之 API

3. Puts() - 用來傳送字串

傳送資料至標準 I/O Port 之前，需呼叫 Puts() 函式來傳送資料到 COM Port。

4. Putch( ) - 用來傳送一個字元

傳送資料至標準 I/O Port，需呼叫 Putch() 函式來傳送資料到 COM Port。

5. Print()

C 函式庫中，像是 Print() 函式允許由 COM Port 輸出。

範例 - 透過 COM1 來讀取與接收資料。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   int quit=0, data;

   InitLib(); /* Initiate the upac5000 library */

   while(!quit)
   {
      if(Kbhit()) /* Check if any data is in the input buffer */
      {
         data=Getch(); /* Read data from COM1 */
         Putch(data); /* Send data to COM1 */
         if(data=='q') quit=1; /* If ‘q’ is received, exit the program */
      }
   }
}
```

範例 - 透過 COM1 來顯示資料。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   int i;

   InitLib(); /* Initiate the upac5000 library */

   for(i=0; i<10; i++)
   {
      Print("Test %d\r\n", i);
   }
}
```

4.1.4. COM Port 函式之對照

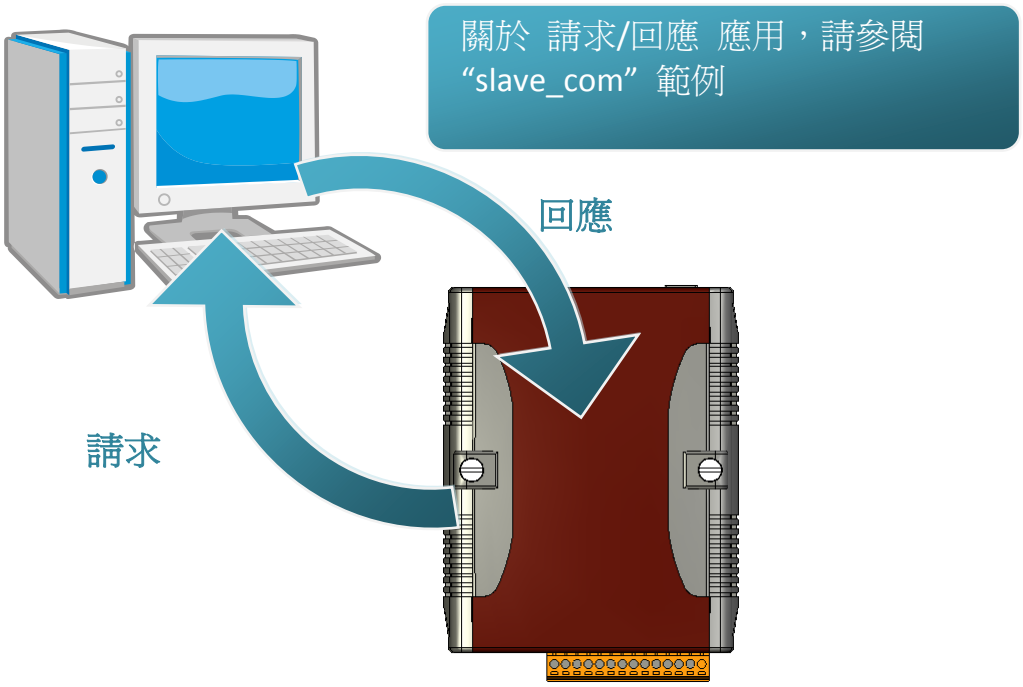
範例 - 學習如何顯示 ASCII 碼。

MiniOS7 的 COM Port 函式	標準的 COM Port 函式

#include <stdio.h>
#include "upac5000.h"
 	
void main(void)
{
  unsigned char item;
 	
  InitLib();
  InstallCom(1,115200,8,0,1);
  printCom(1,"Press any key.\n");
  printCom(1,"Press the ESC to exit!\n");
 	
  for(;;){
  if(IsCom(1)){
  item=ReadCom(1);
  if(item=='q') return;
  else{
  printCom(1,"----------\r\n");
  printCom(1,"char: ");
 	
  ToCom(1,item);
  printCom(1,"\r\nASCII(%c)\r\n",item);
  printCom(1,"Hex(%02X)\r\n",item);
  }
  }
  }
 	
  Delay(10);
  RestoreCom(1);
}
	#include <stdio.h>
	#include "upac5000.h"
	
	void main(void)
	{
	  unsigned char item;
	
	  InitLib();
	
	  Print("Press any key.\n");
	  Print("Press the ESC to exit!\n");
	
	  for(;;){
	  if(Kbhit()){
	  item=Getch();
	  if(item=='q') return;
	  else{
	  Print("----------\r\n");
	  Print("char: ");
	
	  Putch(item);
	  Print("\r\nASCII(%c)\r\n",item);
	  Print("Hex(%02X)\r\n",item);
	  }
	  }
	  }
	
	}

4.1.5. 定義 COM Port 的 請求/回應 通訊協定

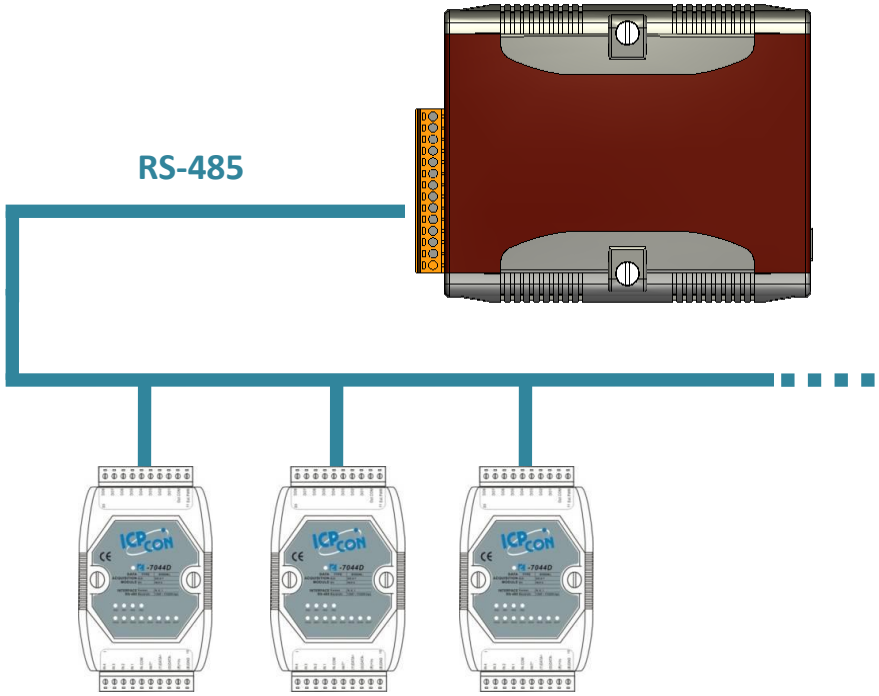
請求/回應 通訊方式是一種很典型的通訊協定架構。若您想設計出如下表中通訊協定的指令集，可參考 “slave_com” 範例。



請求	回應
c1	除錯資訊：指令 1 指令 1
c2	除錯資訊：指令 2 指令 2
Q	除錯資訊：離開程式
其它指令	除錯資訊：未知指令

4.2. 用於 I/O 模組之 API

µPAC-5001D-CAN2 配備有一個 RS-485 的通訊介面－COM2，適用於在廣大範圍的 RS-485 網路應用中存取 I-7K 系列 I/O 模組，如下圖所示。



以下是與 I-7K 系列 I/O 模組通訊的步驟：

步驟 1: 使用 Installcom() 函式來安裝 COM Port 的驅動程式。
步驟 2: 使用 SendCmdTo7000(2,…) 函式來傳送指令。
步驟 3: 使用 ReceiveResponseFrom7000_ms() 函式來取得回應。
步驟 4: 使用 RestoreCom() 函式來回存 COM Port 的驅動程式。

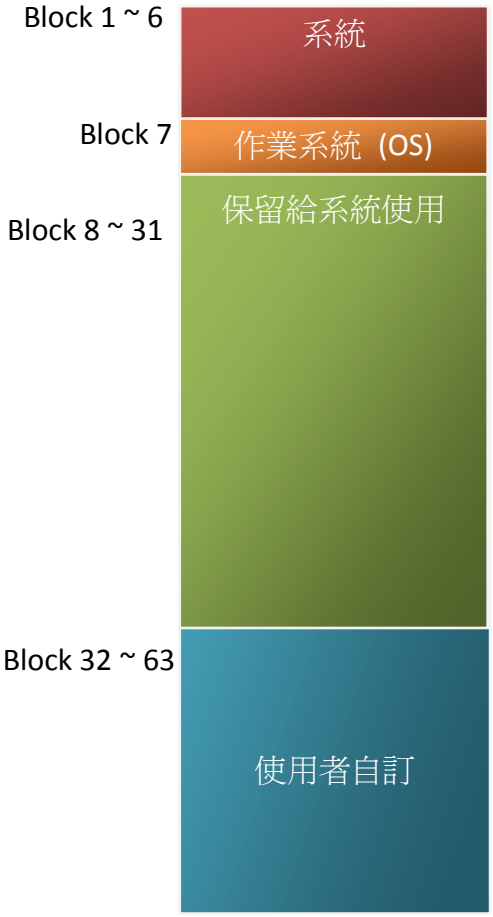
範例－傳送指令 '$01M' 至 I-7K I/O 模組以取得模組名稱。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   unsigned char InBuf0[60];

   InitLib(); /* Initiate the upac5000 library */

   InstallCom(1, 115200L, 8, 0, 1); /* Install the COM1 driver */
   InstallCom(2, 115200L, 8, 0, 1); /* Install the COM2 driver */

   SendCmdTo7000(2, "$01M", 0);      /* Send DCON command via COM2 */

   /* Timeout=50ms, check sum disabled */
   ReceiveResponseFrom7000_ms(2, InBuf0, 50, 0);

   printCom(1, "Module Name = %s", InBuf0);
   Delay(10); /* Wait for all data are transmitted to COM port */

   RestoreCom(1); /* Release the COM1 */
   RestoreCom(2); /* Release the COM2 */
}
```

4.3. 用於 EEPROM 之 API

- EEPROM 含有 64 個區塊 (block 0 ~ 63)，且每個區塊為 256 byte (address 0 ~ 255)，總容量為 16,384 byte (16 KB)。
- EEPROM 的預設模式寫入保護模式。
- 系統程式與作業系統 (OS) 皆儲存在 EEPROM 中，其分配位置如下圖所示。

寫入資料至 EEPROM 之 API



1. EE_WriteEnable()
   寫入資料至 EEPROM 之前，需先呼叫 EE_WriteEnable() 函式來允許寫入。

2. EE_WriteProtect()
   資料寫入至 EEPROM 完成後，需呼叫 EE_WriteProtect() 函式設定為防寫。

3. EE_MultiWrite()
   使用 EE_WriteEnable() 來允許寫入 EEPROM 後，需呼叫 EE_MultiWrite() 來寫入資料。

從 EEPROM 讀取資料之 API

4. EE_MultiRead()

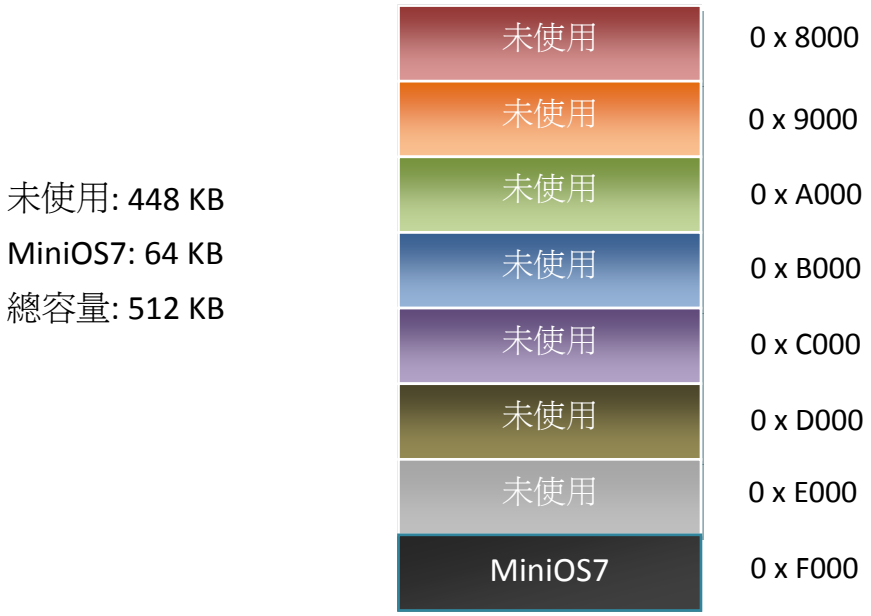
   此函式可從 EEPROM 中讀取多個 byte 的資料。

範例一寫入資料至 EEPROM 的區塊 1、位址 10：

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   int data=0x55, data2;

   InitLib(); /* Initiate the upac5000 library */

   EE_WriteEnable();
   EE_MultiWrite(1, 10, 1, &data);
   EE_WriteProtect();

   EE_MultiRead(1, 10, 1, &data2); /* Now data2=data=0x55 */
}
```

4.4. 用於快閃記憶體之 API

- µPAC-5001D-CAN2 控制器配備有一個 512 KB 的快閃記憶體（Flash Memory）。
- MiniOS7 使用了最後 64 KB；記憶體其餘部分被用來儲存使用者程式或資料。
- 快閃記憶體只能在空的位址被寫入，因此在進行寫入動作之前需先抹除（Erase）記憶體，即表示將所有的資料位元設定為 1，此步驟完成，即可寫入新的資料。



抹除快閃記憶體之 API

1. EraseFlash()

寫入資料前，您必須先使用 EraseFlash() 函式來抹除快閃記憶體中的區塊。(即將所有資料位元皆設為 1)

將資料寫入快閃記憶體之 API

2. FlashWrite()

呼叫 FlashWrite() 函式，將資料寫入至快閃記憶體。

3. FlashRead()

呼叫 FlashRead() 函式，讀取快閃記憶體中的資料。

範例－將整數值寫入至區段 0xD000，且快閃記憶體的偏移位址（offset）為 0x1234。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   int data=0xAA55, data2;
   char *dataptr;
   int *dataptr2;

   InitLib(); /* Initiate the upac5000 library */

   EraseFlash(0xd000); /* Erase a block from the Flash memory */
   dataptr=(char *) &data;
   FlashWrite(0xd000, 0x1234, *dataptr++);
   FlashWrite(0xd000, 0x1235, *dataptr);

   /* Read data from the Flash Memory (method 1) */
   dataprt=(char *) &data2;
   *dataptr=FlashRead(0xd000, 0x1234);
   *(dataptr+1)=FlashRead(0xd000, 0x1235);

   /* Read data from the Flash Memory (method 2) */
   dataptr2=(int far *) _MK_FP(0xd000, 0x1234);
   data=*data;
}
```

4.5. 用於 NVRAM 之 API

- µPAC-5001D-CAN2 配備有一個 RTC (Real Time Clock) 和一個用來儲存資料的 31 bytes NVRAM。
- NVRAM 其實是 SRAM，但它使用了電池來保持資料。因此，斷電時 NVRAM 中的資料不會遺失。
- NVRAM 沒有寫入次數的限制 (Flash 與 EEPROM 皆有寫入限制)。若未發生漏電流 (leakage current) 的狀況，電池可使用 10 年。

將資料寫入 NVRAM 之 API

1. WriteNVRAM()

   需呼叫 WriteNVRAM() 函式，以將資料寫入至 NVRAM。

從 NVRAM 讀取資料之 API

2. ReadNVRAM()

   需呼叫 ReadNVRAM() 函式，以讀取 NVRAM 中的資料。

範例一使用以下程式碼來將資料寫入 NVRAM 的位址 0。

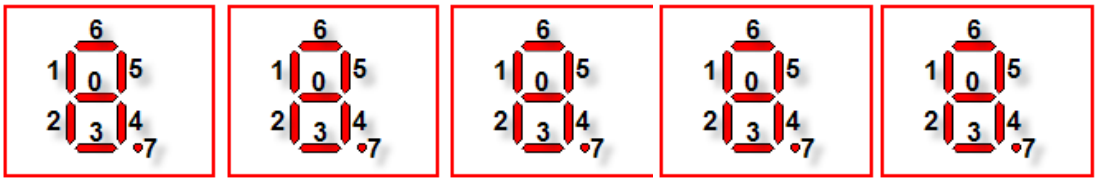
```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   int data=0x55, data2;

   InitLib(); /* Initiate the upac5000 library */

   WriteNVRAM(0, data);
   data2=ReadNVRAM(0); /* Now data2=data=0x55 */
}
```

範例－以下程式碼可用來將整數值（2 byte）寫入至 NVRAM。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   int data=0xAA55, data2;
   char *dataptr=(char *) &data;

   InitLib(); /* Initiate the upac5000 library */

   WriteNVRAM(0, *dataptr); /* Write the low byte */
   WriteNVRAM(1, *dataptr+1); /* Write the high byte */
   dataptr=(char *) &data2;
   *dataptr=ReadNVRAM(0); /* Read the low byte */
   (*dataptr+1)=ReadNVRAM(1); /* Read the high byte */
}
```

4.6. 用於五位數七段 LED 之 API

µPAC-5001D-CAN2 含有一個 五位數七段 LED 顯示器，且每位數的右邊皆有一個小數點。它可用來顯示數字，IP 位址，時間…等等。



開始使用 五位數七段 LED 顯示器 之 API

1. Init5DigitLed()

使用 LED 功能之前，需呼叫 Init5DigitLed() 函式來初始化 五位數七段 LED 顯示器。

於五位數七段 LED 顯示器上顯示訊息之 API

2. Show5DigitLed()

呼叫 Init5DigitLed() 函式初始化後，需呼叫 Show5DigitLed() 函式來將資訊顯示在將五位數七段 LED 顯示器。

範例－使用以下程式碼，於五位數七段 LED 顯示器上顯示 “8000E” 。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   InitLib(); /* Initiate the upac5000 library */

   Init5DigitLed();
   Show5DigitLed(1,8);
   Show5DigitLed(2,0);
   Show5DigitLed(3,0);
   Show5DigitLed(4,0);
   Show5DigitLed(5,14); /* The ASCII code for the letter ‘E’ is 14 */
}
```

4.7. 用於計時器之 API

µPAC-5001D-CAN2 的 OS 提供一個系統的計時信號 (Time tick)，其為 16 位元且精度為 1 ms，可用來支援 8 個碼錶計時器 與 8 個倒數計時器。

開始使用計時器 (Timer) 之 API

1. TimerOpen()

   使用計時器功能之前，需在程式的開頭呼叫 TimerOpen() 函式。

讀取計時器之 API

2. TimerResetValue()

   讀取計時器之前，需呼叫 TimerResetValue() 函式將 Time Tick 重置為 0。

3. TimerReadValue()

   使用 TimerResetValue() 函式將 Time Tick 重置為 0 後，需呼叫 TimerReadValue()函式來讀取 Time Tick。

停止計時器之 API

4. TimerClose()

   程式的結尾，需呼叫 TimerClose() 函式來停止計時器。

範例一以下程式碼可用來從 0 開始讀取 Time Tick。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   Unsigned long time iTime;

   InitLib(); /* Initiate the upac5000 library */

   TimerOpen();
   While(!quit)
   {
   If(Kbhit())
   TimerResetValue(); /* Reset the main time ticks to 0 */

   iTime=TimerReadValue(); /* Read the main time ticks from 0 */
   }

   TimerClose(); /* Stop using the uPAC5000 timer function */
}
```

4.8. 用於看門狗計時器（WDT）之 API

- µPAC-5001D-CAN2 配備有 MiniOS7－小型核心作業系統。MiniOS7 採用計時器 2（CPU 內部計時器）為系統計時器，它是 16-bits 的計時器且每 1 ms 產生一次中斷，因此系統精度為 1 ms。
- 看門狗計時器始終處於啟用狀態，並由系統計時器 ISR（中斷服務常式）刷新它。
- 當系統發生停滯或錯誤時，可使用看門狗計時器將系統重置以恢復正常。其逾時週期為 0.8 秒。

開始使用看門狗功能（WDT）之 API

1. EnableWDT()

   啟動 WatchDog Timer 功能，當系統發生遲緩或停擺時可立即將系統重置，確保系統正常工作。

2. RefreshWDT()

   更新 WatchDog Timer 狀態，使用 WatchDog Timer 功能將系統重置後，需再呼叫 RefreshWDT() 回復系統。

3. DisableWDT()

   關閉 WatchDog Timer 功能。

範例－以下用來刷新看門狗計時器。

```
#include <stdio.h>
#include "upac5000.h"

void main(void)
{
   Unsigned long time iTime;

   InitLib(); /* Initiate the upac5000 library */

   Enable WDT();
   While(!quit) {
      RefreshWDT();
      User_function();
   }
   DisableWDT();
}
```

4.9. 用於 microSD 之 API

所需的函式庫與標頭檔:
SD_V102.LIB 與 microSD.h

µPAC-5001D-CAN2 可支援一片容量為 1 GB 或 2 GB 的 microSD 卡。

- microSD 函式之說明：

函式	說明
pc_init	初始化 microSD socket 的函式庫。
pc_open	1. 開啟舊檔並回傳檔案控制代碼 (File Handle)。 2. 建立新檔。
pc_close	關閉檔案並釋放檔案控制代碼 (File Handle)。
pc_read	讀取指定檔案。
pc_write	寫入指定檔案。
pc_seek	將檔案指標由當前偏移值移動置相對偏移值。
pc_tell	取得檔案指標的目前偏移值。
pc_eof	檢查是否已到檔案結尾。
pc_format	將 microSD 卡 格式化為 FAT (FAT16) 格式。
pc_mkdir	建立一個目錄或子目錄。
pc_rmdir	移除現存的目錄。
pc_move	重新命名現存檔案、目錄或子目錄。
pc_del	刪除指定檔案。
pc_deltree	刪除指定目錄或子目錄。
pc_isdir	檢查該檔是否為一個目錄。
pc_isvol	檢查指定路徑的最終是一個子目錄或檔案。
pc_size	取得指定檔案大小。
pc_set_cwd	設定目前工作目錄。
pc_get_cwd	取得目前工作目錄的路徑名稱。
pc_gfirst	移動指標至第一個元素 (element)。
pc_gnext	移動指標至下一個元素 (element)。
pc_gdone	移動指標至最後一個元素 (element)。
pc_get_freeSize_KB	取得 SD 記憶卡的可使用空間。
pc_get_usedSize_KB	取得 SD 記憶卡的已使用空間。
pc_get_totalSize_KB	取得 SD 記憶卡的總容量。
pc_get_attributes	取得檔案屬性。
pc_set_attributes	設定檔案屬性。
pc_get_errno	取得錯誤編號。

開始使用 microSD 之 API

1. pc_ Init()

使用 miscroSD 功能前，需呼叫 PC_Init() 來初始化 microSD。

啟用/關閉 microSD 之 API

2. pc_open()

在寫入／讀取資料至 microSD 卡之前，需呼叫 PC_open() 來開啟檔案。

3. pc_close()

完成寫入／讀取資料至 microSD 卡後，需呼叫 PC_close() 依檔案控制代碼 (File Handle) 來關閉檔案。

寫入資料至 microSD 之 API

4. pc_write()

此函式可添加一個指定數量之同等大小的資料項目於 microSD 裡的檔案中。

範例，寫入資料至 microSD 卡。

```
#include <string.h>
#include <stdio.h>
#include "upac5000.h"
#include "microSD.h"

void main(void)
{
   int fd, iRet;

   InitLib();

   if(pc_init())
      Print("Init microSD ok\r\n");
   else
   {
      Print("Init microSD failed\r\n");
      iRet=pc_get_errno();
      switch(iRet)
      {
         case PCERR_BAD_FORMAT: //1
```

```
            Print("Error 01: format is not FAT\r\n");
            break;
        case PCERR_NO_CARD: //2
            Print("Error 02: no microSD card\r\n");
            break;
        default:
            Print("Error %02d: unknow error\r\n", iRet);
            break;
      }
   }

   fd=pc_open("test.txt", (word) (PO_WRONLY|PO_CREAT|PO_APPEND),
                                    (word) (PS_IWRITE|PS_IREAD));
   if(fd>=0)
   {
      pc_write(fd, "1234567890", 10); /* write 10 bytes */
      pc_close(fd);
   }

}
```

從 microSD 卡中讀取資料之 API

5. pc_read()

使用 PC_open() 開啟檔案後，需呼叫 PC_read() 來讀取 microSD 中的資料。

範例一讀取 microSD 中的資料：

```
#include <string.h>
#include <stdio.h>
#include "upac5000.h"
#include "microSD.h"

void main(void)
{
   int fd, iRet;
   unsigned char Buffer[80];

   InitLib();

   if(pc_init())
      Print("Init microSD ok\r\n");
   else
   {
      Print("Init microSD failed\r\n");
      iRet=pc_get_errno();
      switch(iRet)
      {
         case PCERR_BAD_FORMAT: //1
```

```
            Print("Error 01: format is not FAT\r\n");
            break;
        case PCERR_NO_CARD: //2
            Print("Error 02: no microSD card\r\n");
            break;
        default:
            Print("Error %02d: unknow error\r\n", iRet);
            break;
    }
  }

  fd=pc_open("test.txt", (word) (PO_RDONLY), (word) (PS_IWRITE|PS_IREAD));
  if(fd>=0)
  {
    iRet=pc_read(fd, Buffer, 10); /* reads 10 bytes */
    Buffer[10]=0; /* adds zero end to the end of the string */
    pc_close(fd);
    Print("%s", Buffer);
  }
}
```

請參閱下列位置，取得關於 microSD 的範例程式：

CD:\NAPDOS\uPAC-5000\Demo\Basic\microSD\

http://ftp.Icpdas.com/pub/cd/8000cd/napdos/upac-5000/demo/basic/microsd/

4.10. CAN bus 之 API

函式定義	說明
CAN_Reset	CAN 控制器硬體重置
XWCANInit	XWCAN 硬體初始化
SetCANBaud	更改 CAN 的鮑率
SetCANMask	更改 CAN 信息的濾波器
CAN_InstallIrq	啟用嵌入式控制器中斷
CAN_RemoveIrq	停用嵌入式控制器中斷
CAN_Restore	釋放資源並停用嵌入式控制器中斷程序
CAN_CreateBuffer	更改接收與傳輸緩衝區大小
SendCANMsg	發送 CAN 信息到 CAN 網路
GetCANMsg	接收 CAN 訊息
GetStatus	取得 CAN 控制器動狀態並傳送接收/傳輸緩衝器狀態
ClearStatus	重置接收與傳輸緩衝區狀態
CL1Off	關閉 LED (CL1)
CL2Off	關閉 LED (CL2)
CL3Off	關閉 LED (CL3)
CL1On	開啟 LED (CL1)
CL2On	開啟 LED (CL2)
CL3On	開啟 LED (CL3)
UserCANInt	設計使用者自定中斷程序
CAN_SearchBaud	搜尋必要的 CAN Bus 鮑率

4.10.1. CAN 初始化 API

CAN 重置 API

1. CAN_Reset(int CANPort)

藉由硬體電路重新啟動 CAN 控制器。在執行此函式後，CAN 控制器設定為初始狀態。更多關於此功能的資訊，請參考 SJA1000 控制器資料表，網址如下。

http://www.semiconductors.philips.com/pip/SJA1000.html#datasheet.

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

CAN 初始化 API

2. XWCANInit(int CANPort, char IntMode, unsigned long CANBaud, char BT0, char BT1, unsigned long AccCode, unsigned long AccMask)

初始化軟體緩衝區及 XW-CAN 200 硬體，其中包含 CAN 控制器、CL1、CL2 及 CL3。

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

IntMode: 設定 CAN 控制器中斷模式。IntMode 參數的每一位元表示不同的功能，如下所示。

中斷種類	IntMode 值
啟用接收中斷	0x01
啟用傳輸中斷	0x02
啟用錯誤警告中斷	0x04
啟用資料溢位中斷	0x08
啟用喚醒中斷	0x10
啟用 Error Passive 中斷	0x20
啟用仲裁遺失中斷	0x40
啟用 Bus 錯誤中斷	0x80

中斷類型	意義
Receive Interrupt	當一個信息正確無誤的被接收時，將觸發接收中斷。
Transmit Interrupt	當一個信息成功傳送或傳輸緩衝器是可再存取的狀態時，將觸發傳輸中斷。
Error Warning Interrupt	如果錯誤或 bus 狀態被設定或清除時，將觸發錯誤中斷。
Data Overrun Interrupt	如果在 FIFO (FIFO 有 64 bytes)中沒有足夠的空間，而造成信息遺失時，將觸發溢位中斷。
Wake-up Interrupt	當 CAN 控制器處於睡眠狀態，且偵測到 bus 上有訊號的時後，將觸發喚醒(Wake-up)中斷。
Error Passive Interrupt	如果 CAN 控制器至少有一個錯誤計數器超過協議定義的 127 計數值，或 CAN 控制器處於 error passive 狀態時，將觸發 Error Passive 中斷。
Arbitration Lost Interrupt	當 CAN 控制器遺失一個仲裁，並且成為接收器時，將觸發仲裁遺失中斷。
Bus Error Interrupt	當 CAN 控制器在 CAN bus 上偵測到錯誤時，將觸發 Bus 錯誤中斷。

IntMode: 利用一個字元的值來實現中斷。例如，如果在 BasicCAN(CAN 2.0A)模式中，是需要接收與溢位中斷時，則設定 IntMode 值為 0x09(意謂著 0x01+0x08)。

CANBaud: 使用一個長整數(long int)來設定此參數。例如，如果使用者欲設定 CAN 的鮑率為 125K bps 時，則設此參數值為 125000UL。

BT0, BT1: 自定鮑率設定。使用者可藉由此參數任意的設定鮑率，但前提是需具有 SJA1000 CAN 控制器與 TJA1042 CAN 收發器，且自行運算 BT0 與 BT1 的值(CAN 控制器的時脈頻率為 16MHz)。

AccCode, AccMask: AccCode 用來決定 CAN 控制器將接受何種 ID 類型。 而 AccMask 用來決定 ID 信息的那些位元需使用 AccCode 來檢查。如果 AccMask 的任一位元被設定為 0 時，這意謂著 ID 信息相同位元值需檢查，並且需要與 AccCode 的相同位元匹配。

關於 11-bit ID 訊息:

暫存器	暫存器位元	濾波器目標
AccCode[0] and AccMask[0]	bit7~bit0	ID 的 bit10 ~ bit3
AccCode[1] and AccMask[1]	bit7~bit5	ID 的 bit2 ~ bit0
AccCode[1] and AccMask[1]	bit4	RTR
AccCode[1] and AccMask[1]	bit3~bit0	未使用
AccCode[2] and AccMask[2]	bit7~bit0	第一個字元資料的 bit7 ~ bit0
AccCode[3] and AccMask[3]	bit7~bit0	第二個字元資料的 bit7 ~ bit0

關於 29-bit ID 訊息:

暫存器	暫存器位元	濾波器目標
AccCode[0] and AccMask[0]	bit7~bit0	ID 的 bit28 ~ bit21
AccCode[1] and AccMask[1]	bit7~bit0	ID 的 bit20 ~ bit13
AccCode[2] and AccMask[2]	bit7~bit0	ID 的 bit12 ~ bit5
AccCode[3] and AccMask[3]	bit7~bit3	ID 的 bit4 ~ bit0
AccCode[3] and AccMask[3]	bit2	RTR
AccCode[3] and AccMask[3]	bit1~bit0	未使用

注意: 1. AccCode[0]指 AccCode 的最低有效位元組而 AccCode[3]指 AccCode 的最高有效位元組。

2. AccMask[0]指 AccMask 的最低有效位元組而 AccMask[3]指 AccMask 的最高有效位元組。

3. Bit10 為最高位元而 Bit0 為最低有效位元。

例如 (在 29 bit ID 訊息):

AccCode : 00h 00h 00h A0h
AccMask : FFh FFh FFh 1Fh
ID Value : ?? ?? ?? Ah and Bh 將被接受。(??:任意位元組的資料)

(注意:“h” 表示該值為十六進制格式)

小技巧 變更 Baud Rate 與 變更 Filter

在呼叫 XWCANInit 函式後，可藉由 SetCANBaud 函式來更改 CAN 的鮑率。

SetCANBaud (int CANPort, unsigned long CANBaud, char BT0, char BT1)

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

CANBaud, BT0, BT1: XWCANInit 函式的參數設明。

在呼叫 XWCANInit 函式後，可藉由 SetCANMask 函式來更改 CAN 信息濾波器的設定。

SetCANFilter(int CANPort, unsigned long AccCode,
unsigned long AccMask)

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

注意:因 µPAC-5001D-CAN1 只有一個 CAN 通訊埠,參數"CANPort"為 0。

AccCode, AccMask:請參考 XWCANInit 函式的參數說明。

釋放 CAN 資源 API

3. CAN_Restore(int CANPort)

停用 CAN 中斷功能，釋放所有軟體緩衝區，並且重置 CAN 晶片。在程式中止前，必須呼叫此函式來釋放系統資源。

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

範例，CAN 初始化設定。

```
#include <stdlib.h>
#include "..\lib\UPAC5000.h"
#include "..\lib\XWCAN.h"

/*If the UserCANInt() function is not used, please don't remove it.*/
void UserCANInt(int CANPort,char CANInt)
{
   ...
}

void main(void)
{
    int ret;
    InitLib();     //Initial uPAC-5000 LIB
    /*********************************************************/
    /*          initialize and configure the CAN controller                */
    /*********************************************************/
    ret=XWCANInit(0,0,125000UL,0,0,0x00000000UL,0xffffffffUL);
    /*
        XCANInit function Parameter descriptions
        0                   : for CAN Port 1
        0                   : for IntMode useless
        125000UL            : for CAN baud
        0                   : for BT0 of user defined baud
        0                   : for BT1 of user defined baud
        0x00000000UL : for AccCode of CAN message filter
        0xffffffffUL : for AccMask of CAN message filter
    */
    switch (ret)
    {
        /*Return Code,Check if configuration is OK*/
        case CAN_ResetError:
            Print("Reset Error!\n\r");
            return;
        case CAN_SetACRError:
            Print("\n\rSet ACR Error!");
```

```
                return;
            case CAN_SetAMRError:
                Print("\n\rSet AMR Error!");
                return;
            case CAN_SetBaudRateError:
                Print("\n\rSet Baud Rate Error!");
                return;
            case CAN_BaudNotSupport:
                Print("\n\rBaud Rate Not Support!");
                return;
            case CAN_ConfigError:
                Print("\n\rConfiguration Failure!");
                return;
              case CAN_SetPortError:
                Print("\n\rSet CAN Port Failure!");
                return;
        }

    CAN_Restore(0); //release resource
    /*CAN_Restore function parameter descirptions:
        0               :for CAN Port 1
    */
}
```

4.10.2. CAN 中斷 API

啟用 CAN 中斷 API

1.CAN_InstallIrq(int CANPort)

啟用 CAN 中斷功能。之後 μPAC-5001D-CAN2 的 CPU 嵌入的控制器就可以接收 CAN 控制器發送的中斷信息。

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

停用 CAN 中斷 API

2.CAN_RemoveIrq(int CANPort)

停用 CAN 中斷功能。之後 μPAC-5001D-CAN2 的 CPU 嵌入的控制器就無法再接收 CAN 控制器發送的中斷信息。

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

改變 CAN 傳送/接收中斷軟體緩衝區 API

3.CAN_CreateBuffer(int CANPort, int BufMode, unsigned int BufferSize)

此函式為改變接收與傳輸訊息的軟體緩衝區大小。

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

BufMode: “0” 表示為改變接收軟體緩衝區大小，其餘則表示為改變傳輸軟體緩衝區大小。

BufferSize: 軟體緩衝區的新空間大小。

注意:在啟用 CAN 中斷後，若使用者未呼叫此功能，API 將會自動建立傳送/接收軟體緩衝區，其大小為 256 筆。

範例，中斷。

```
#include <stdlib.h>
#include "..\lib\UPAC5000.h"
#include "..\lib\XWCAN.h"

/*If the UserCANInt() function is not used, please don't remove it.*/
void UserCANInt(int CANPort,char CANInt)
{
   ...
}
void main(void)
{
      int ret;
      InitLib();     //Initial uPAC-5000 LIB
      /*********************************************************/
      /*          initialize and configure the CAN controller                  */
      /*********************************************************/
      ret=XWCANInit(0,1,125000UL,0,0,0x00000000UL,0xffffffffUL);
      /*
          XCANInit function Parameter descriptions
          0                  : for CAN Port 1
          1                  : for reception interrupt
          125000UL           : for CAN baud
          0                  : for BT0 of user defined baud
          0                  : for BT1 of user defined baud
          0x00000000UL : for AccCode of CAN message filter
```

```
        0xffffffffUL : for AccMask of CAN message filter
    */
    switch (ret)
    {
        /*Return Code,Check if configuration is OK*/
        case CAN_ResetError:
            Print("Reset Error!\n\r");
            return;
        …
        …
      }

    CAN_CreateBuffer(0,0,1000);
    /*CAN_CreateBuffer function parameter descirptions:
        0           :for CAN Port 1
        0           :for CAN reception buffer
        1000        :for 1000 CAN messages
    */
    CAN_InstallIrq(0); //Install Port0 IRQ
    /*CAN_InstallIrq function parameter descirptions:
        0           :for CAN Port 1
    */
    CAN_Restore(0);
}
```

4.11.3. CAN 資料傳送 API

傳送 CAN 資料 API

1. SendCANMsg(int CANPort, unsigned char mod, unsigned long MsgID,
unsigned char RTR, unsigned char DataLen, unsigned cahr *Data)

如果傳輸訊息的緩衝區被停用時，呼叫此函式會先判斷目前 CAN 控制器能否接收訊息，如果可以，則直接寫入一筆信息到 CAN 控制器上，如果不行則會一直檢查 CAN 控制器何時能接收 CAN 訊息而卡在此函式中，直到逾時跳離函式或 CAN 控制器允許寫入訊息後，將一筆訊息寫入控制器為止。之後 CAN 控制器會在 CAN 網路可使用的狀態下將訊息傳出去。然而，若是傳輸訊息的緩衝區被啟用且 XWCANInit 函式的 IntMode 參數設定傳送中斷為致能，呼叫此函式會先判斷目前 CAN 控制器能否接收訊息，如果可以，則寫入一筆訊息到 CAN 控制器中，如果不行，則會將 CAN 訊息儲存在緩衝區內並離開此函式，等 CAN 控制器允許接收訊息時，自動將緩衝區的資料寫入 CAN 控制器中。

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

Mode: 此參數用來表示 CAN ID 的類型。

Mode 值	意義
0	發送 11-bit ID CAN 信息。
others	發送 29-bit ID CAN 信息。

MsgID:CAN 信息的 ID。ID 值可能為 11-bit 或 29-bit。

RTR:遠端傳輸要求位元。

RTR 值	意義
0	此 CAN 信息非遠端傳輸要求信息。
1	此 CAN 信息為遠端傳輸要求信息。

DataLen:CAN 信息的純資料長度，此值的範圍介於 0~8。

*Data:CAN 信息的資料字元，其字元數需與"DataLen"所設定的相同。

範例，傳送 CAN 訊息

```
#include <stdlib.h>
#include "..\lib\UPAC5000.h"
#include "..\lib\XWCAN.h"

/*If the UserCANInt() function is not used, please don't remove it.*/
void UserCANInt(int CANPort,char CANInt)
{
   ...
}
void main(void)
{
      int ret;
      unsigned char MsgMode,MsgRTR,MsgDataLen,MsgData[8];
      unsigned long MsgID,MsgUpperTimeStamp,MsgLowerTimeStamp;

      InitLib();     //Initial uPAC-5000 LIB
      /*********************************************************/
      /*          initialize and configure the CAN controller                */
      /*********************************************************/
      ret=XWCANInit(0,0,125000UL,0,0,0x00000000UL,0xffffffffUL);
      /*
          XCANInit function Parameter descriptions
          0                 : for CAN Port 1
          0                 : for IntMode useless
          125000UL          : for CAN baud
          0                 : for BT0 of user defined baud
          0                 : for BT1 of user defined baud
          0x00000000UL : for AccCode of CAN message filter
          0xffffffffUL : for AccMask of CAN message filter
      */
      switch (ret)
      {
          /*Return Code,Check if configuration is OK*/
          case CAN_ResetError:
                Print("Reset Error!\n\r");
                return;
```

```
    …
    …
}

while(1){
    /*********************************************************/
    /*                  Send CAN messages to CAN bus         */
    /*********************************************************/
    ret=SendCANMsg(0,1,0x12345678UL,0,8,MsgData);
    /*      SendCANMsg function parameter descirptions:
           0                : for CAN Port 1
           1                : for Send message with 29-bit ID
           0x12345678UL : for CAN message ID
           0                : for CAN message RTR
           8                : for CAN message data length
           MsgData          : for CAN message output data
    */
    if (ret)
    {
        switch(ret)
        { /*Return Code,Check if configuration is OK*/
          case CAN_DataLengthError:
                Print("Transmission Data Length Error!\n");
                break;
          case CAN_TransmitIncomplete:
                Print("Transmission is incomplete!\n");
                break;
          case CAN_TransmitBufferLocked:
                Print("CAN controller transmit Buffer is locked!\n");
                break;
          }
          break;
    }
    DelayMs(500);
    ret=SendCANMsg(0,0,0x123UL,0,8,MsgData);
    /*    SendCANMsg function parameter descirptions:
            0                : for CAN Port 1
            0                : for Send message with 11-bit ID
```

```
                0x123UL         : for CAN message ID
                0               : for CAN message RTR
                8               : for CAN message data length
                MsgData         : for CAN message output data
        */
        if (ret)
        {
           switch(ret)
           { /*Return Code,Check if configuration is OK*/
              case CAN_DataLengthError:
                    Print("Transmission Data Length Error!\n");
                    break;
              case CAN_TransmitIncomplete:
                    Print("Transmission is incomplete!\n");
                    break;
              case CAN_TransmitBufferLocked:
                    Print("CAN controller transmit Buffer is locked!\n");
                    break;
           }
           break;
        }
        DelayMs(500);
        if (Kbhit()){ /*if press any key, exit the program*/
              Print("Exit this program!\n");
              break;
        }
    }
    CAN_Restore(0);
}
```

4.11.4. CAN 資料接收 API

接收 CAN 資料 API

1. **GetCANMsg(int CANPort,unsigned char *Mode, unsigned long *MsgID, unsigned char *RTR, unsigned char *DataLen, unsigned char *Data, unsigned long *UpperTime , unsigned long *LowerTime)**

從接收緩衝區或直接從 CAN bus 接收 CAN 信息。如果在 XWCANInit 函式的 IntMode 參數設定接收中斷為致能，則此函式將自動讀回儲存在軟體緩衝區的 CAN 信息。反之，若接收中斷設為停用，此函式使用輪循方法檢查 CAN 控制器內，是否有任何 CAN 信息，若有則將該信息回傳。由於 CAN 控制器內的硬體緩衝區很小，因此如果不使用接收中斷，當 CAN 總線資料流量大時，有可能會因為使用者來不及呼叫此函式導致 CAN 控制器內的原有未讀取訊息被最新收到訊息覆蓋掉的問題。

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

*Mode: 此參數用來取得 CAN 信息的 ID 類型(11-bit ID 或 29-bit ID)。

*MsgID: 取得 CAN 信息的 ID。

*RTR: 取得 CAN 信息的 RTR。

RTR 值	意義
0	此 CAN 信息非遠端傳輸要求信息。
1	此 CAN 信息為遠端傳輸要求信息。

*DataLen: 取得 CAN 信息的資料長度。

*Data: 取得 CAN 信息的資料。資料緩衝區的大小必須是 8 個字元。

*UpperTime: 取得 CAN 信息的時間標記，單位為 us (micro second)。此參數只顯示上部分的時間標記。

即時標記(Real time stamp) = 上半部分(upper part)＊0x1000000UL+下半部分(lower part)

*LowerTime: 取得 CAN 信息的下半部分的時間標記。

小技巧 取得 CAN 狀態

可藉由 GetStatus 讀取 CAN 控制器狀態與軟體緩衝區溢位旗標信息。

GetStatus(int CANPort,unsigned char *CANReg,
unsigned char *OverflowFlag)

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

*CANReg:此指標為取得目前 CAN 控制器狀態。有關於 CANReg 值所代表的意義，請參考如下表格。

Bit NO.	說明
7 (MSB)	Bus 狀態。“1”為 bus off；“0”為 bus on。
6	錯誤狀態。“1”為至少一個錯誤；“0”為 OK。
5	傳送狀態。“1”為傳送中；“0”為閒置。
4	接收狀態。“1”為接收中；“0”為閒置。
3	傳送完成狀態。“1”為完成；“0”為未完成。
2	傳送緩衝區狀態。“1”為釋放；“0”為鎖住。
1	資料溢位狀態。“1”為接收緩衝區溢位；“0”為正常。
0 (LSB)	接收緩衝區狀態。“1”為至少一個信息儲存在接收緩衝區；“0”為空的。

* OverflowFlag: CAN 接收與傳輸溢位旗標資訊。有關於 OverflowFlag 值所代表的意義，請參考如下表格。

Bit NO.	說明
Others	保留
1	“1”為接收軟體緩衝區溢位；“0”為正常的。
0 (LSB)	“1”為傳輸緩衝器溢位；“0”為正常的。

可藉由 ClearStatus 函式來清除 CAN 接收或傳輸軟體緩衝區溢位旗標。

ClearStatus(int CANPort)

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

注意:因 µPAC-5001D-CAN1 只有一個 CAN 通訊埠,參數“CANPort”為 0。
當任一個緩衝區滿了，其對應的溢位旗標被設為 “1” 。
在此情形下，使用者需使用此函式來清除溢位旗標來回應錯誤資訊。

範例，接收。

```
#include <stdlib.h>
#include "..\lib\UPAC5000.h"
#include "..\lib\XWCAN.h"

/*If the UserCANInt() function is not used, please don't remove it.*/
void UserCANInt(int CANPort,char CANInt)
{
   ...
}
void main(void)
{
      int ret;
      unsigned char MsgMode,MsgRTR,MsgDataLen,MsgData[8];
      unsigned long MsgID,MsgUpperTimeStamp,MsgLowerTimeStamp;

      InitLib();     //Initial uPAC-5000 LIB
      /*************************************************************/
      /*          initialize and configure the CAN controller      */
      /*************************************************************/
      ret=XWCANInit(0,0,125000UL,0,0,0x00000000UL,0xffffffffUL);
      /*
          XCANInit function Parameter descriptions
          0                 : for CAN Port 1
          0                 : for IntMode useless
          125000UL          : for CAN baud
          0                 : for BT0 of user defined baud
          0                 : for BT1 of user defined baud
          0x00000000UL : for AccCode of CAN message filter
          0xffffffffUL      : for AccMask of CAN message filter
      */
      switch (ret)
      {
```

```
    /*Return Code,Check if configuration is OK*/
    case CAN_ResetError:
          Print("Reset Error!\n\r");
          return;
    …
    …
 }

while(1)
{
    /*********************************************************/
    /*              Receive CAN messages from CAN bus        */
    /*********************************************************/
    ret=GetCANMsg(0,&MsgMode,&MsgID,&MsgRTR,&MsgDataLen,
                  MsgData,&MsgUpperTimeStamp,&MsgLowerTimeStamp);
    /* GetCANMsg function Parameter descriptions
          0             : CAN Port 1
          &Mode         : Get CAN ID is 2.0A or 2.0B
          &MsgID        : Get CAN Message ID
          &RTR          : Get CAN Message RTR
          &DataLen      : Get CAN Message Data Length
          &Data         : Get CAN Message Data
          &UpperTime : Get the time stamp of a CAN message
          &LowerTime : Get the lower part of time stamp of a CAN message
    */
    if (!ret)
    {
       Print("%lu(%lus-%luus):Mode=%d,ID=%lx,RTR=%d,Len=%d",MsgCnt,MsgUpper
       TimeStamp,MsgLowerTimeStamp,MsgMode,MsgID,MsgRTR,MsgDataLen);
       if (MsgDataLen && !MsgRTR)
       {
            Print(",Data=");
            for (i=0;i<MsgDataLen;i++){
                Print("%x,",MsgData[i]);
            }
            Print("\n\r");
       }
        else
```

```
            {
                switch(ret)
                {
                    case CAN_DataLengthError:
                        Print("Reception Data Length Error!\n");
                        break;
                    case CAN_DataOverrun:
                        Print("Software Reception Data Buffer Overrun!\n");
                        break;
                    case CAN_ReceiveError:
                        Print("Receive data Error!\n");
                        break;
                    case CAN_ReceiveBufferEmpty:
                        /*No message in receive buffer*/
                        break;
                }
            }
        }
    }
    CAN_Restore(0);
}
```

4.10.5. CAN 指示燈 API

CAN LED 指示燈 API

1. CL1Off()、CL2Off()、CL3Off()

   關閉 CL1、CL2、CL3 指示燈。

2. CL1On()、CL2On()、CL3On()

   開啟 CL1、CL2、CL3 指示燈。

範例，CAN LED 指示燈控制。

```
#include <stdlib.h>
#include "..\lib\UPAC5000.h"
#include "..\lib\XWCAN.h"

/*If the UserCANInt() function is not used, please don't remove it.*/
void UserCANInt(int CANPort,char CANInt)
{
   ...
}
void main(void)
{
      int ret,i=0;
      InitLib();
      /*********************************************************/
      /*          initialize and configure the CAN controller                  */
      /*********************************************************/
      ret=XWCANInit(0,0,125000UL,0,0,0x00000000UL,0xffffffffUL);
      switch (ret)
      {
          /*Return Code,Check if configuration is OK*/
          case CAN_ResetError:
                Print("Reset Error!\n\r");
                return;
          …
```

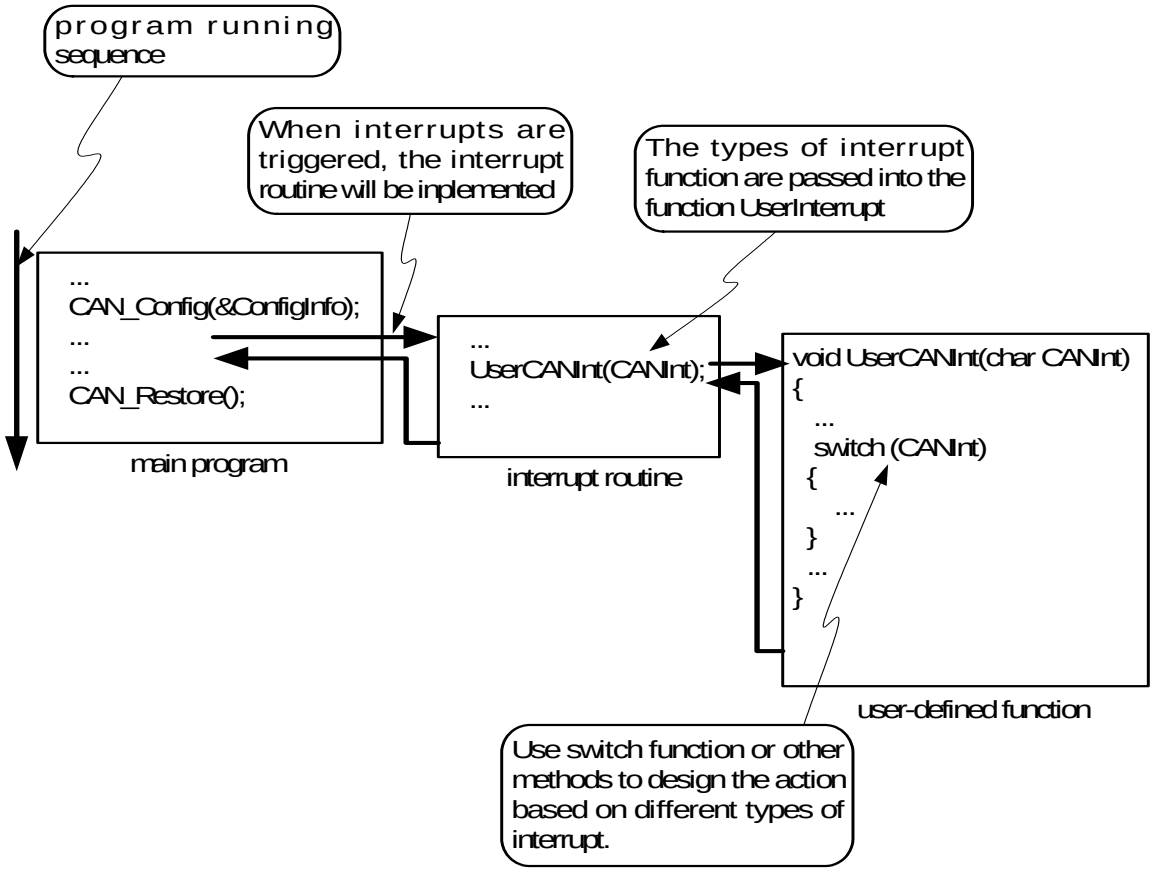
```
    }

    /*************************************************************/
    /*                        Initial all LEDs                   */
    /*************************************************************/
    Print("Press any key to exit the program.\n\r");
    while(1){
        switch(i)
        {
            case 0:
                CL1On();
                CL2Off();
                CL3Off();
                break;
            case 1:
                CL1Off();
                CL2On();
                CL3Off();
                break;
            case 2:
                CL1Off();
                CL2Off();
                CL3On();
        }
        if (++i==3) i=0;
        DelayMs(500);
    }
    CL1Off();
    CL2Off();
    CL3Off();
    CAN_Restore(0);
}
```

4.10.6. CAN 控制器使用者中斷 API

CAN 控制器使用者中斷 API

1. UserCANInt (int CANPort, char CANInt)

此函式由使用者建立，並用於 CAN 中斷服務常式中被呼叫。當 CAN 中斷功能被觸發時，中斷服務常式自動地傳送參數 CANInt 到 UserCANInt 函式中，以便通知使用者那一種 CAN 控制器中斷的種類被觸發了。因此，使用者只需於 UserCANInt 函式中根據不同的中斷種類設計它們的中斷服務功能。如果不使用此函式，請於使用者的.C 檔中保留此函式宣告(函式內容為空的)，以避免編譯錯誤。下圖是 UserCANInt 函式的運行概念。



參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

CANInt: 中斷服務程式將透過 CANInt 參數向使用者說明何種中斷被觸發。有關於 CANInt 參數的意義，請參考下表。

CANIntMode 值 (Hex)	意義
0x01	成功接收信息
0x02	成功傳送信息
0x04	錯誤警告
0x08	資料溢位
0x10	喚醒(wake-up)CAN 控制器
0x20	Bus Passive
0x40	仲裁遺失
0x80	Bus 錯誤

範例，使用者中斷。

```
#include <stdlib.h>
#include "..\lib\UPAC5000.h"
#include "..\lib\XWCAN.h"

unsigned long MsgCount=0;    //Create a global variable to check the    number of RX Interrupt

/*If the UserCANInt() function is not used, please don't remove it.*/
void UserCANInt(int CANPort,char CANInt)
{ /*check CAN Port 1 receive interrupt*/
   if(CANPort ==0x0)
   {
       if (CANInt==0x01)
        {
            MsgCount++;
        }
    }
}

void main(void)
{
      int ret,i=0;
```

```
 unsigned long tmpMsgCount=0;
InitLib();
/*********************************************************/
/*          initialize and configure the CAN controller                  */
/*********************************************************/
ret=XWCANInit(0,1,125000UL,0,0,0x00000000UL,0xffffffffUL);
 /*
     XCANInit function Parameter descriptions
     0                  : for CAN Port 1
     1                  : for reception interrupt
     125000UL           : for CAN baud
     0                  : for BT0 of user defined baud
     0                  : for BT1 of user defined baud
     0x00000000UL : for AccCode of CAN message filter
     0xffffffffUL : for AccMask of CAN message filter
*/
switch (ret)
{
     /*Return Code,Check if configuration is OK*/
     case CAN_ResetError:
          Print("Reset Error!\n\r");
          return;
     …
}

/*********************************************************/
/*                              Enable Irq                                */
/*********************************************************/
CAN_CreateBuffer(0,0,1000);
/* CAN_CreateBuffer function parameter descirptions:
     0          :for CAN Port 1
     0          :for CAN reception buffer
     1000       :for 1000 CAN messages
*/
 CAN_InstallIrq(0); //Install Port0 IRQ
/* CAN_InstallIrq function parameter descirptions:
     0          :for CAN Port 1
*/
```

```
        while(1)
        {
            if(MsgCount != tmpMsgCount)
            {
                Print("%lu CAN message%c %s received.\n\r",
                MsgCount,(MsgCount<2)?' ':'s',(MsgCount<2)?"is":"are");
             }
        }

        CAN_Restore(0);
    }
```

4.10.7. 搜尋 CAN bus 鮑率 API

搜尋 CAN bus 鮑率 API

1. CAN_SearchBaud(**int CANPort,unsigned long CANBaud, char BT0, char BT1, unsigned int Timeout**)

進入“監聽模式(Listen Only Mode)”，並且啟動接收與錯誤中斷來檢測 CAN bus 的正確位元率。當信息成功的接收時，將回傳“CAN_NoError”信息，否則回傳"CAN_AutoBaudTimeout"信息。

參數:

CANPort：選擇 CAN 通訊埠的參數，如下所示:

value	CAN Port
0	For CAN Port 1
1	For CAN Port 2

BT0, BT1:使用者自定鮑率設定。使用者可藉由此參數任意的設定鮑率，但前提是需具有 SJA1000 CAN 控制器與 TJA1042 CAN 收發器，且自行運算 BT0 與 BT1 的值(CAN 控制器的時脈頻率為 16MHz)。

Timeout:設定 timer 是為了搜尋必要的 CAN bus 鮑率。

範例，搜尋 Baud Rate。

```
#include <stdlib.h>
#include "..\lib\UPAC5000.h"
#include "..\lib\XWCAN.h"
/*If the UserCANInt() function is not used, please don't remove it.*/
void UserCANInt(int CANPort,char CANInt)
{
   ...
}
void main(void)
{
      int ret,i;
      unsigned char MsgMode,MsgRTR,MsgDataLen,MsgData[8];
      unsigned long MsgID,MsgUpperTimeStamp,MsgLowerTimeStamp;
```

```
unsigned long CANBaud[]={1000000UL,800000UL,500000UL,250000UL,
                         200000UL,125000UL,100000UL,50000UL,
                         25000UL,20000UL,10000UL,5000UL};
int FoundBaud=0;
InitLib();
/*************************************************************/
/*              Start to detect CAN bus baudrate             */
/*************************************************************/
Print("Start to detect CAN bus baudrate\r\n");
for(i=0;i<=11;i++)
{
   FoundBaud=CAN_SearchBaud(0,CANBaud[i],0,0,500);
   if((FoundBaud == 0) || (FoundBaud != CAN_AutoBaudTimeout))
    break;
}
if(FoundBaud == CAN_NoError)
    Print("Find CAN Baudrate: %lu\r\n",CANBaud[i]);
else if(FoundBaud == CAN_AutoBaudTimeout)
    Print("Timeout, CAN Baudrate Not Find, use default setting.\r\n");
else    Print("CAN_SearchBaud Error, %d\r\n",FoundBaud);
 /*************************************************************/
/*          initialize and configure the CAN controller      */
/*************************************************************/
if(FoundBaud == CAN_NoError)
   ret=XWCANInit(0,3,CANBaud[i],0,0,0x00000000UL,0xffffffffUL);
else
   ret=XWCANInit(0,3,125000UL,0,0,0x00000000UL,0xffffffffUL);
 /* XCANInit function Parameter descriptions
            0                 : for CAN port selection
            3                 : for Receive and transmission interrupt
            125000UL          : for CAN baud
            0                 : for BT0 of user defined baud
            0                 : for BT1 of user defined baud
            0x00000000UL : for AccCode of CAN message filter
            0xffffffffUL : for AccMask of CAN message filter
 */
switch (ret){    /*Check if configuration is OK*/
     case CAN_ResetError:
```

```
                Print("Reset Error!\n\r");
                return;
            case CAN_SetACRError:
                Print("\n\rSet ACR Error!");
                return;
            case CAN_SetAMRError:
                Print("\n\rSet AMR Error!");
                return;
            case CAN_SetBaudRateError:
                Print("\n\rSet Baud Rate Error!");
                return;
            case CAN_BaudNotSupport:
                Print("\n\rBaud Rate Not Support!");
                return;
            case CAN_ConfigError:
                Print("\n\rConfiguration Failure!");
                return;
            }
        }
        while(1)
        {
            /*********************************************************/
            /*              Receive CAN messages from CAN bus        */
            /*********************************************************/
            …
            …
            /*********************************************************/
            /*                Send CAN messages to CAN bus           */
            /*********************************************************/
            …
            DelayMs(100);
            if (Kbhit()){ /*if press any key, exit the program*/
                Print("Exit this program!\n");
                break;
            }
        }
        CAN_Restore(0);
    }
```

4.10.8. 回傳碼

回傳碼	錯誤 ID	註釋
0	CAN_NoError	正常
5	CAN_ResetError	進入重置模式錯誤
8	CAN_ConfigError	CAN 晶片配置錯誤
9	CAN_SetACRError	接受碼暫存器設定錯誤
10	CAN_SetAMRError	接受遮罩暫存器錯誤
11	CAN_SetBaudRateError	鮑率設定錯誤
14	CAN_InstallIrqFailure	中斷功能啟動失敗
15	CAN_RemoveIrqFailure	中斷功能停用失敗
16	CAN_TransmitIncomplete	無法成功地傳送資料
17	CAN_TransmitBufferLocked	資料傳輸尚未完成
18	CAN_ReceiveBufferEmpty	接收緩衝區內無信息
19	CAN_DataOverrun	軟體緩衝區無足夠空間，導致資料遺失
20	CAN_ReceiveError	資料接收未完成
21	CAN_SoftBufferIsFull	軟體傳輸緩衝區已滿
22	CAN_SoftBufferIsEmpty	軟體緩衝區內無信息
23	CAN_BaudNotSupport	不支援此鮑率
24	CAN_DataLengthError	資料長度與總資料字元不一致
25	CAN_NotEnoughMemory	記憶體空間不足，無法建立接收或傳輸軟體緩衝區
50	CAN_AutoBaudTimeout	找不到 CAN bus 鮑率
51	CAN_SendParamError	設定 API SendCANMsg()參數”CANport”錯誤
52	CAN_ReceiveParamError	設定 API GetCANMsg()參數”CAN port”錯誤

附錄 A. 什麼是 MiniOS7?

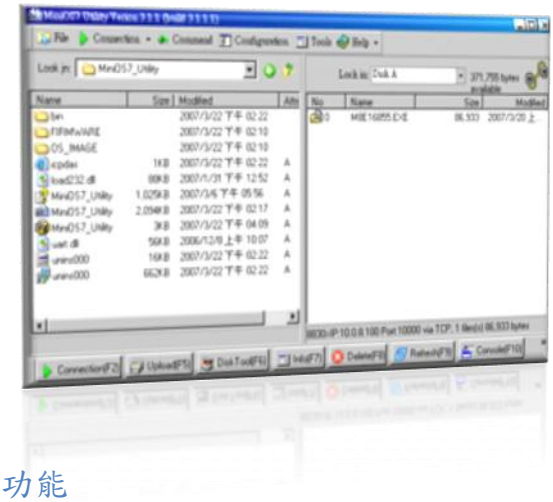
MiniOS7 是泓格科技（ICP DAS）所設計之內嵌式 ROM-DOS 作業系統。功能上與其它 DOS 版本相同，並可運行符合標準 DOS 的可執行檔。

DOS（無論 PC-DOS、MS-DOS 或 ROMDOS）是用來告訴電腦如何處理資訊之指令集或代碼。DOS 執行程式、管理檔案、控制信息處理、指定輸入與輸出，並執行許多其他相關功能。

下列為 MiniOS7 與 ROM-DOS 之特色比較表：

特色	MiniOS7	ROM-DOS
啟動時間	0.1 sec	4 ~ 5 sec
更輕巧的大小	< 64 KB	64 KB
支援 I/O 擴充匯流排	是	否
支援 ASIC 碼	是	否
Flash ROM 記憶體管理	是	否
OS 更新（上傳）	是	否
內建硬體診斷功能	是	否
直接以指令控制 I-7000 系列模組	是	否
客製化（ODM）功能	是	否
免費	是	否

附錄 B. 什麼是 MiniOS7 Utility?



MiniOS7 Utility 是一種軟體工具，用來設定並將檔案上傳至泓格科技（ICP DAS）之所有的內嵌式 MiniOS7 產品。

版本 3.1.1 起，MiniOS7 Utility 可允許使用者透過乙太網路來進行遠端存取控制器（7188E、8000E..等）。

功能

- 支援的連線方式
  1. COM Port 連線（RS-232）
  2. 乙太網路連線（TCP & UDP）
     （支援版本： 3.1.1 起）
- 設定
  1. 日期與時間
  2. IP 位址
  3. COM Port
  4. 磁碟大小（Disk A，Disk B）
- 檔案維護
  1. 上傳檔案
  2. 刪除檔案
  3. 更新 MiniOS7 image
- 檢查產品資訊
  1. CPU 類型
  2. Flash 容量
  3. SRAM 容量
  4. COM Port 數

  …..等等。

包括經常使用的工具

a. 7188XW
b. 7188EU
c. 7188E
---
d. SendTCP
e. Send232
f. VxComm Utility

下載位置:

http://ftp.Icpdas.com/pub/cd/8000cd/napdos/minios7/utility/minios7_utility/

附錄 C. 更多的 C 編譯器設定

此章節描述了下列編譯器之設定方式：

- Turbo C 2.01 編譯器
- BC++ 3.1 IDE（整合開發環境）
- MSC 6.00 編譯器
- MSVC 1.50 編譯器

C.1. Turbo C 2.01

以下有幾個選項，可供您選擇：

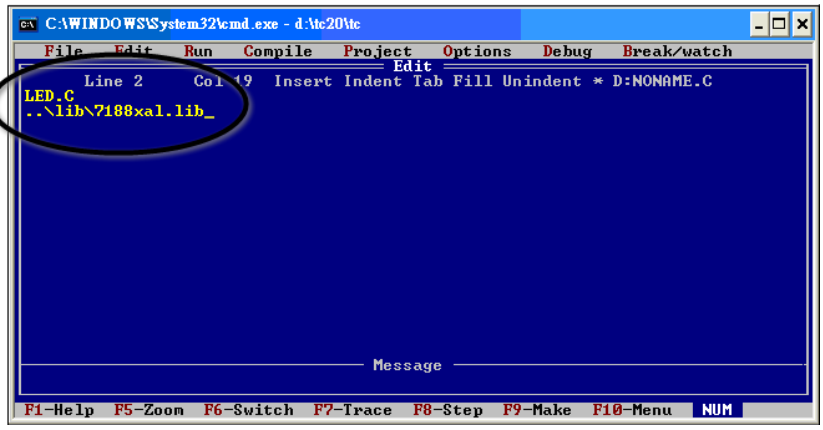
1: 使用指令列。

請參閱以下位置，取得詳細資訊：
CD:\8000\NAPDOS\8000\841x881x\Demo\hello\Hello_C\gotc.bat
tcc -Ic:\tc\include -Lc:\tc\lib
hello1.c ..\..\Demo\basic\Lib\uPAC5000.lib

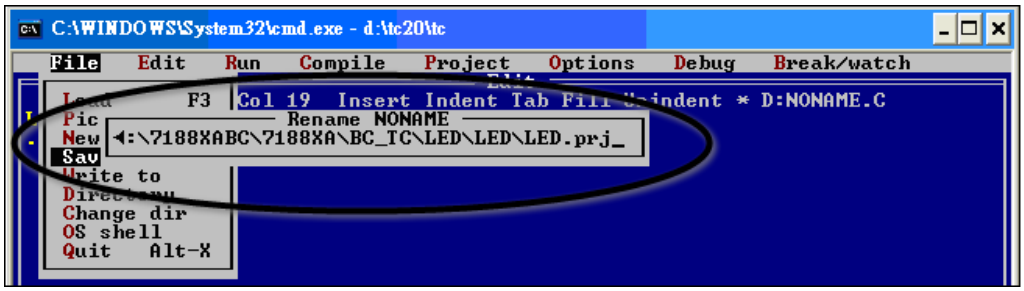
2: 使用 TC 整合環境。

> 步驟 1: 執行 TC 2.01。
>
> 步驟 2: 編輯專案檔。

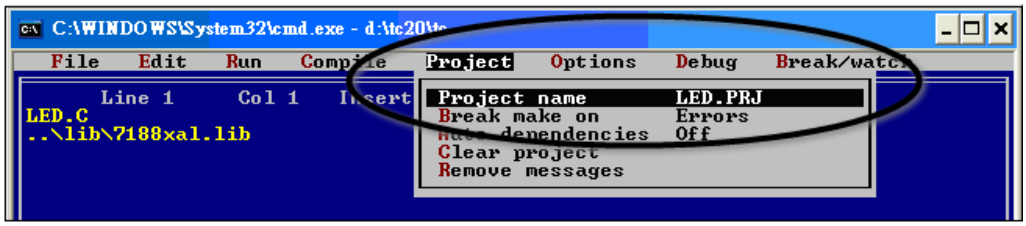
加入所需的函式庫與檔案至專案中。



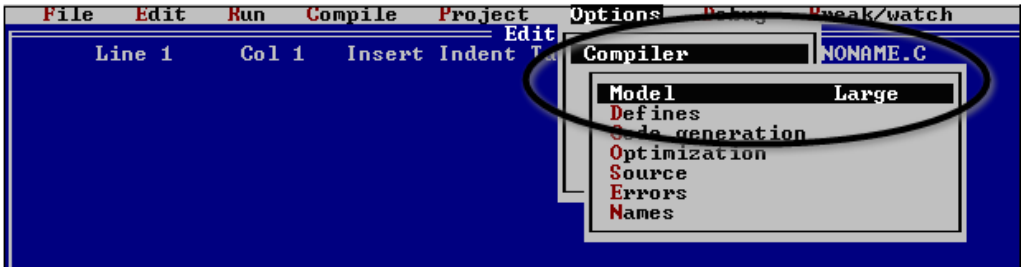
步驟 3: 儲存檔案並輸入檔名，例如 LED.prj。

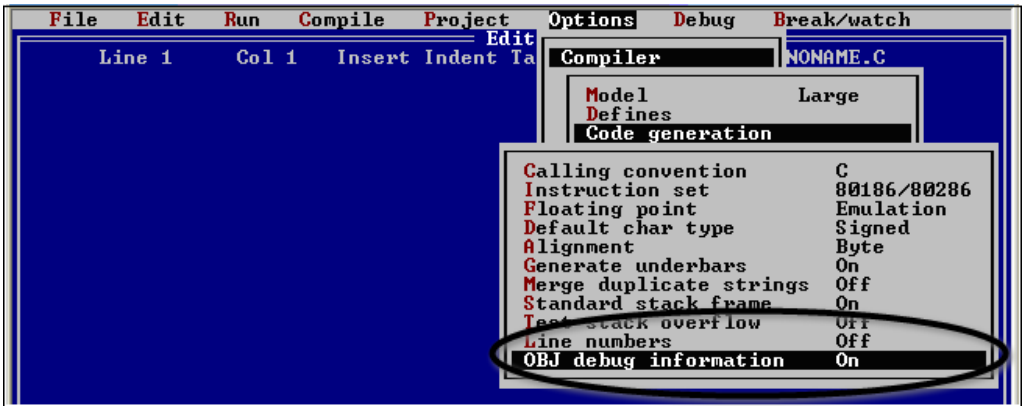


步驟 4: 載入專案。

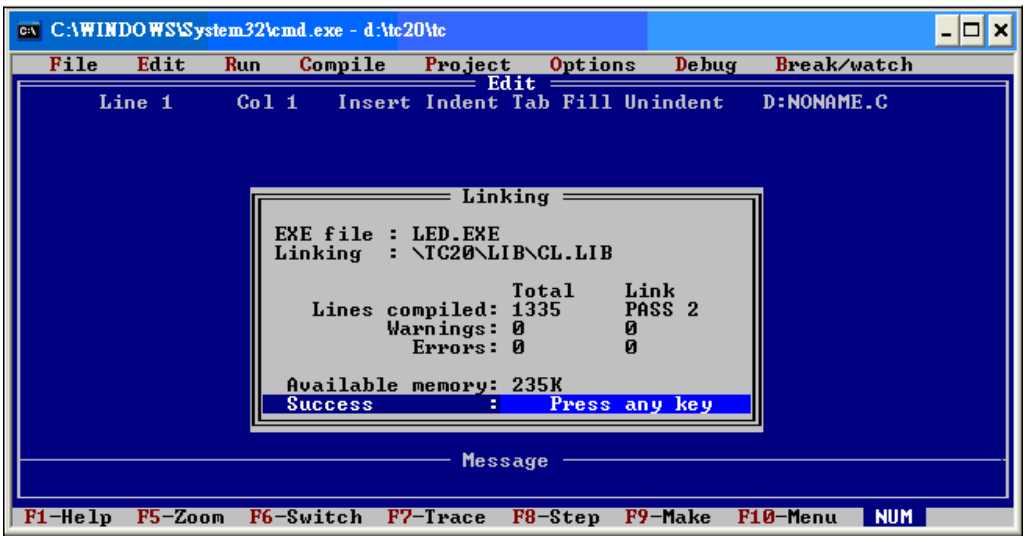


步驟 5: 變更記憶體模式為 “Large”（用於 uPAC5000.lib）並設定 “Code Generation” 為 “80186/80286”。



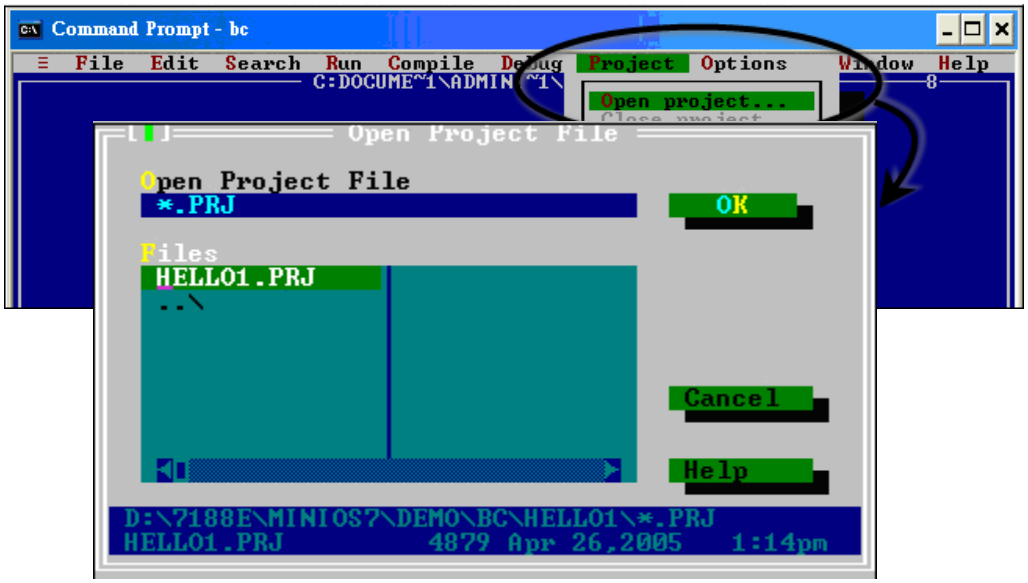


步驟 6: 建立專案。

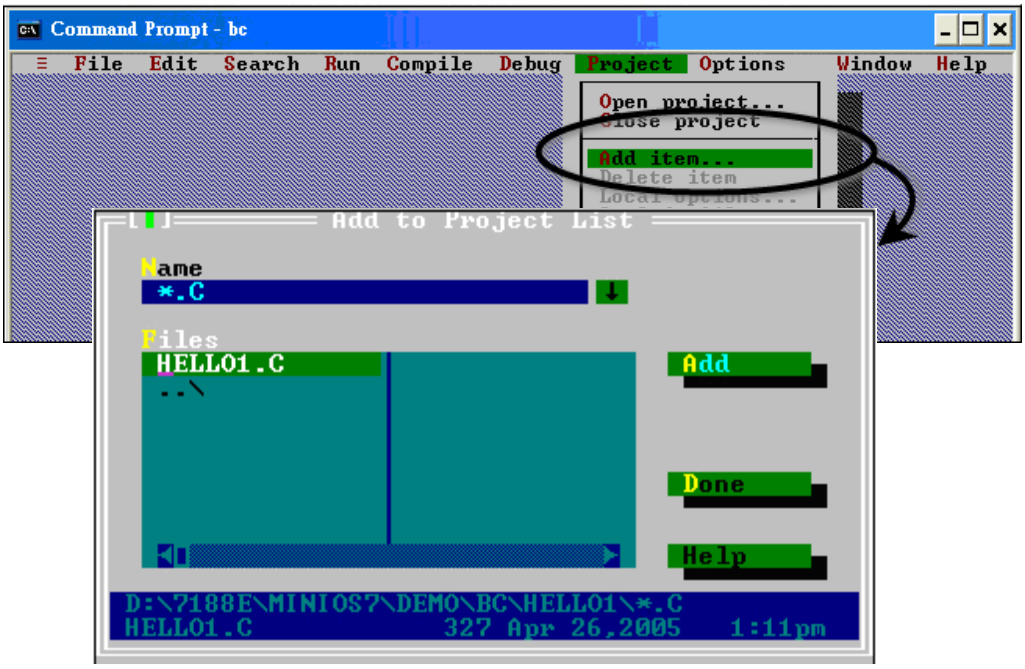


C.2. BC++ 3.1. IDE

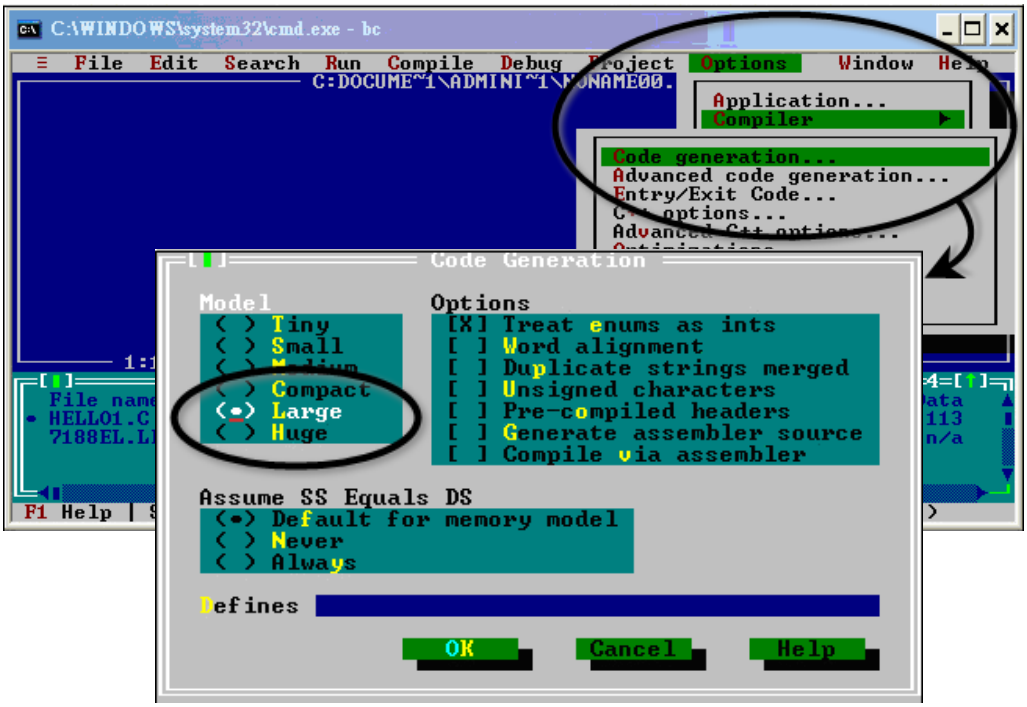
步驟 1: 執行 Borland C++ 3.1。
步驟 2: 新建專案 (*.prj)。



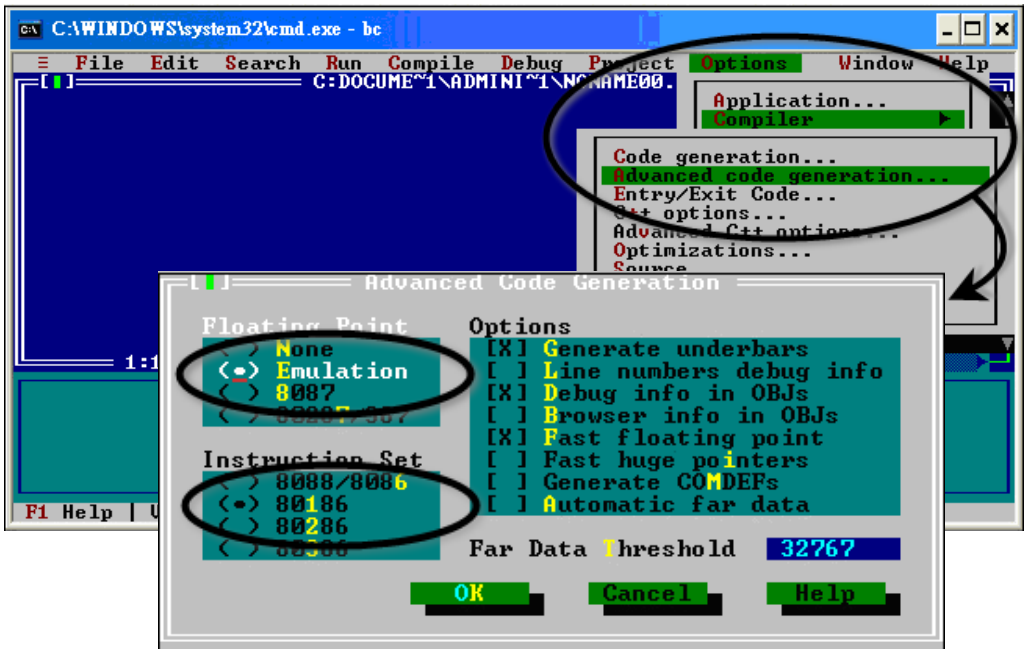
步驟 3: 加入所有需要的檔案至專案中。



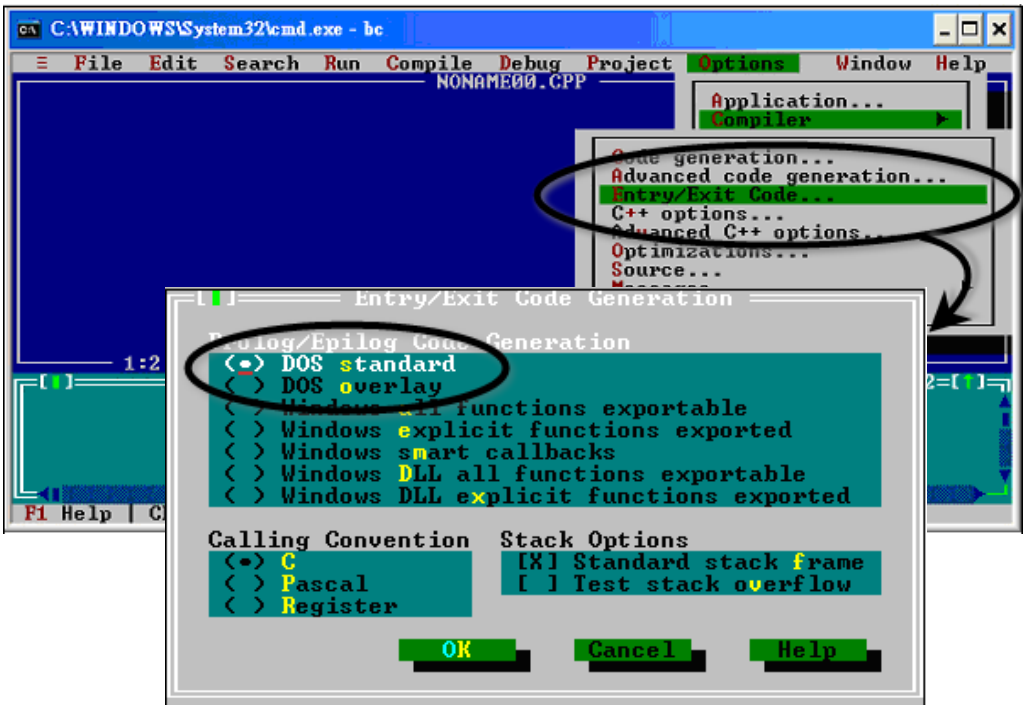
步驟 4: 變更記憶體模式為 “Large” (用於 uPAC5000.lib)。



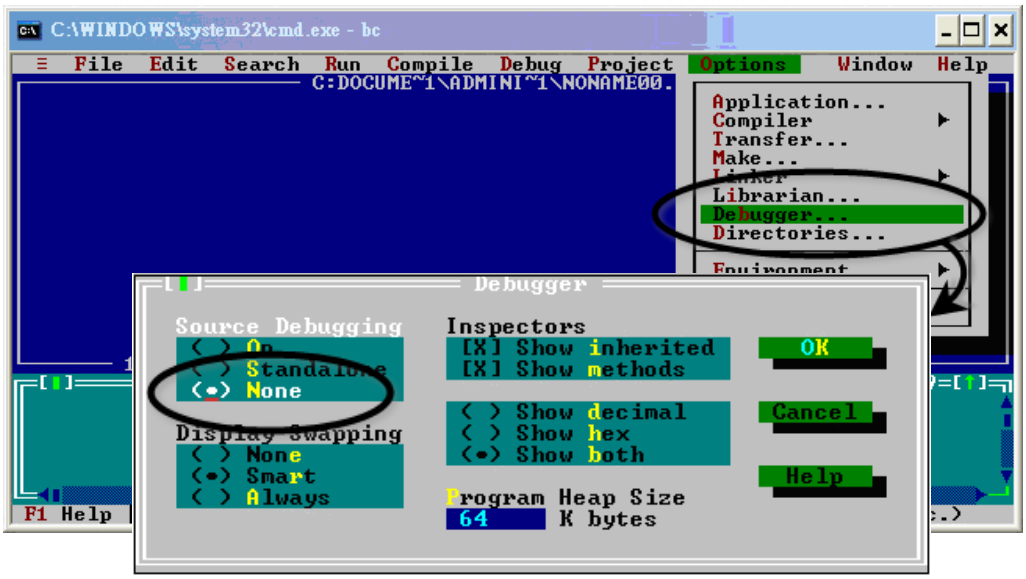
步驟 5: 設定 “Advanced code generation” 選項，將 “Floating Point” 設定為 “Emulation”，並將 “Instruction Set” 設定為 “80186”。



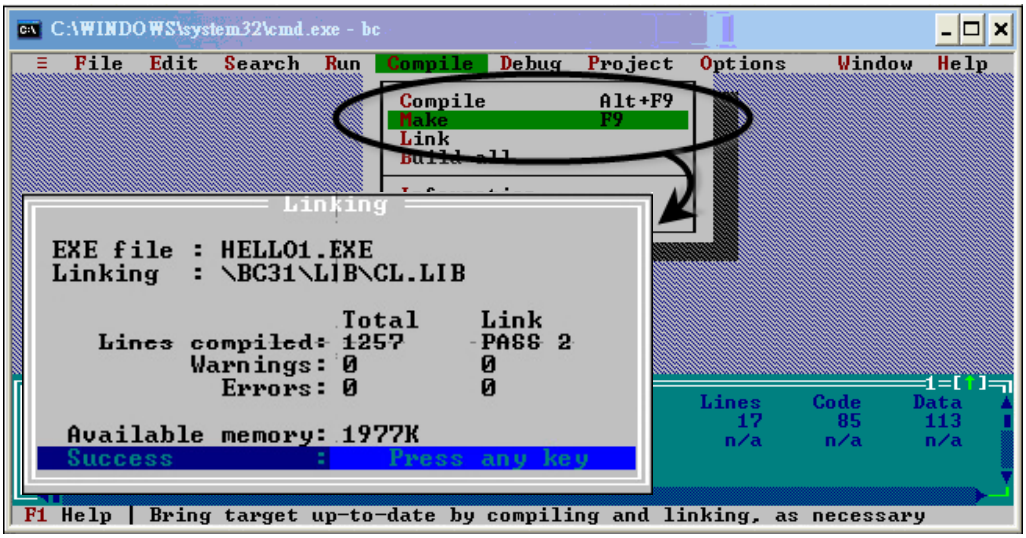
步驟 6: 設定 “Entry/Exit Code Generation” 選項，並選取 “DOS standard”。



步驟 7: 選擇 “Debugger…” 並設定 “Source Debugging” 為 “None” 。

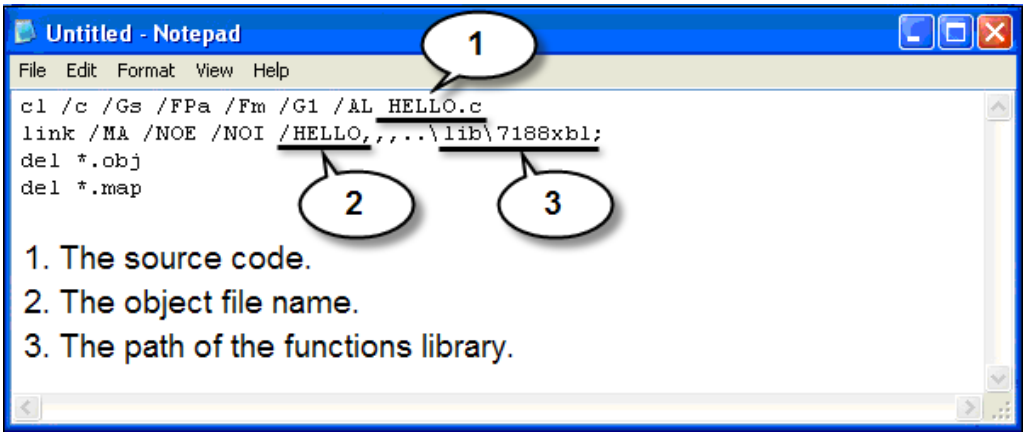


步驟 8: 建立專案。



C.3. MSC 6.00

步驟 1: 於來源檔案資料夾中，使用文字編輯器來建立一個名稱為 Gomsc.bat 的批次檔。



小技巧 與 安全警告

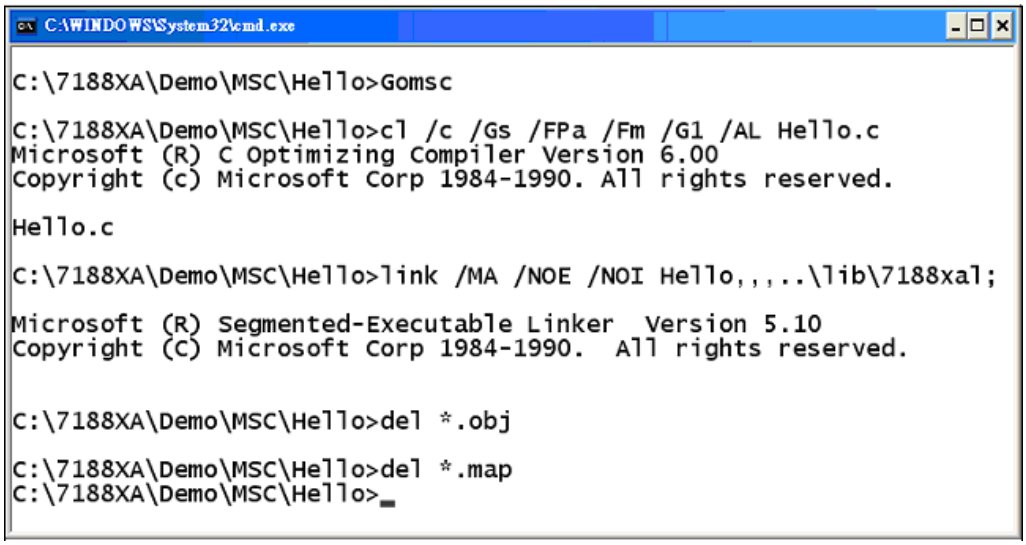
/C 不列出註解 /GS 不檢查堆疊

/Fpa 產生浮點數調用並在目地檔中放入替用函式庫的名稱。

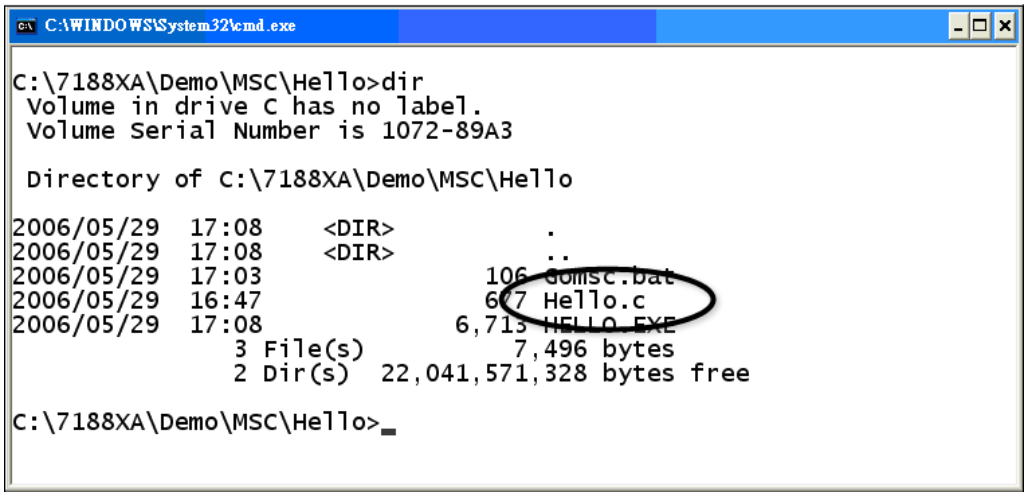
/Fm [map 檔]

/G1 186 指令 /AL Large 模式

步驟 2: 執行 Gomsc.bat 檔案。

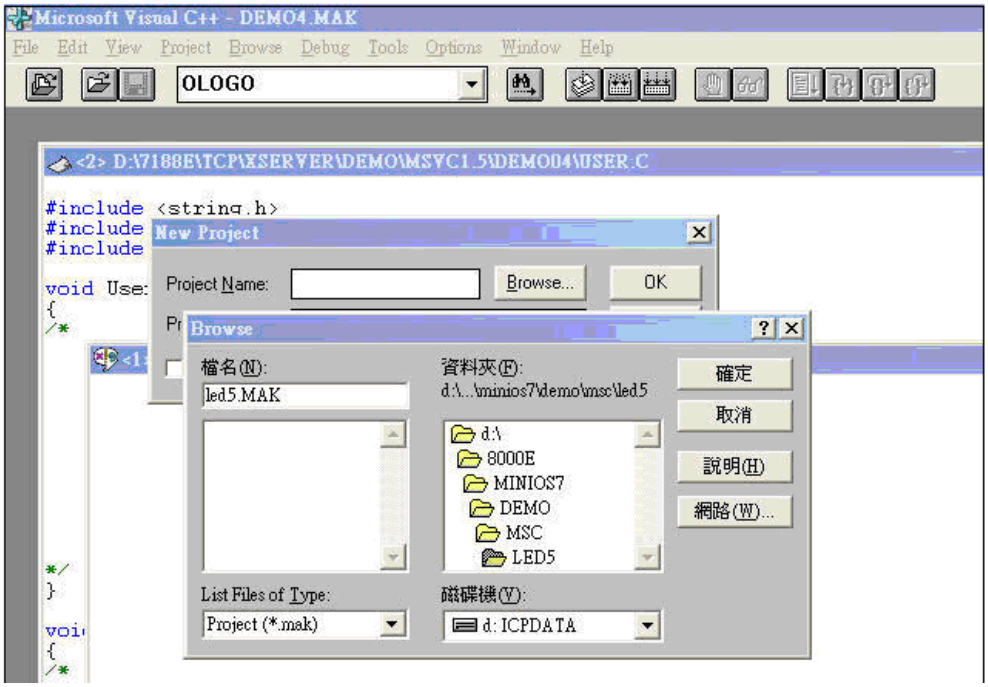


步驟 3: 若編譯成功，將產生一個可執行檔。

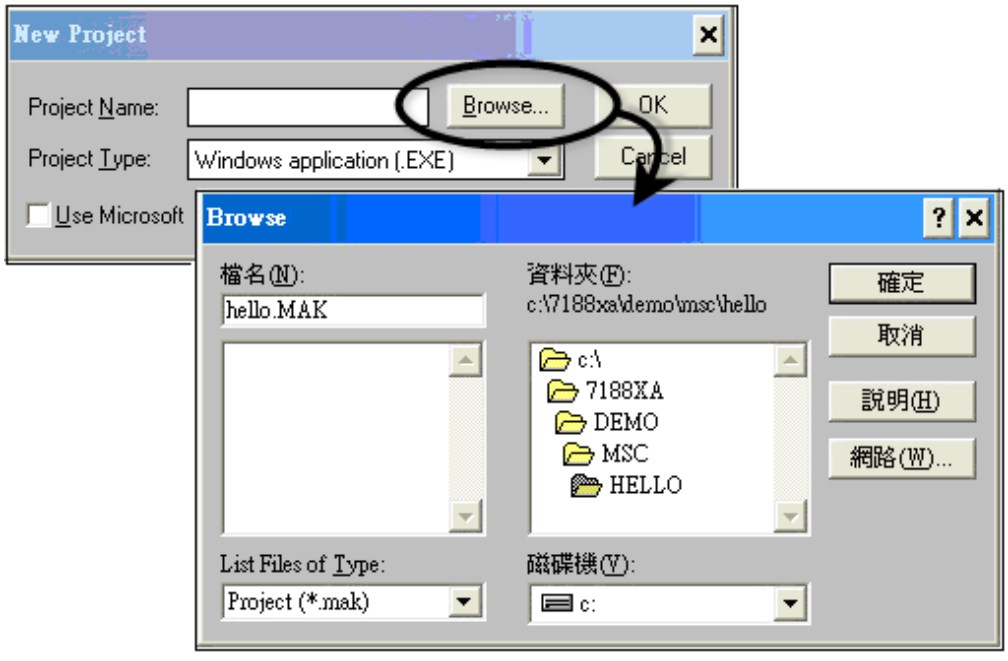


C.4. MSVC 1.50

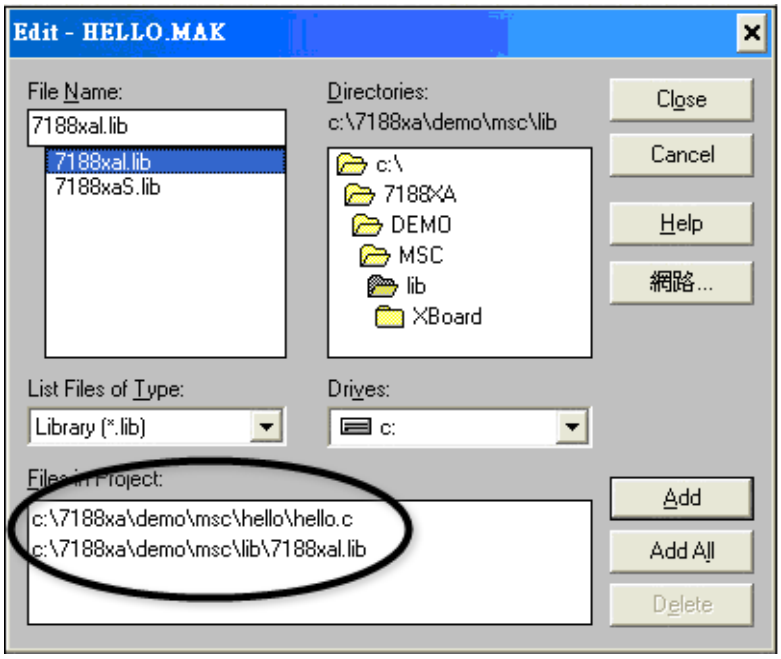
步驟 1: 執行 MSVC.exe。



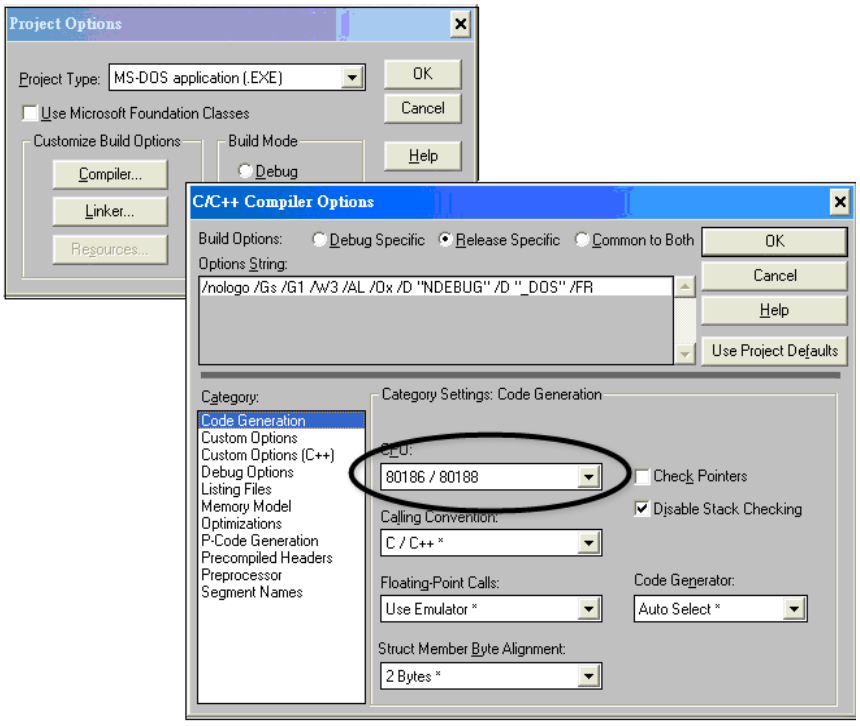
步驟 2: 建立新專案 (*.mak)，於 “Project Name” 欄位輸入專案名稱，並於 “Project Type” 下拉選單中選取 “MS-DOS application (EXE)” 。



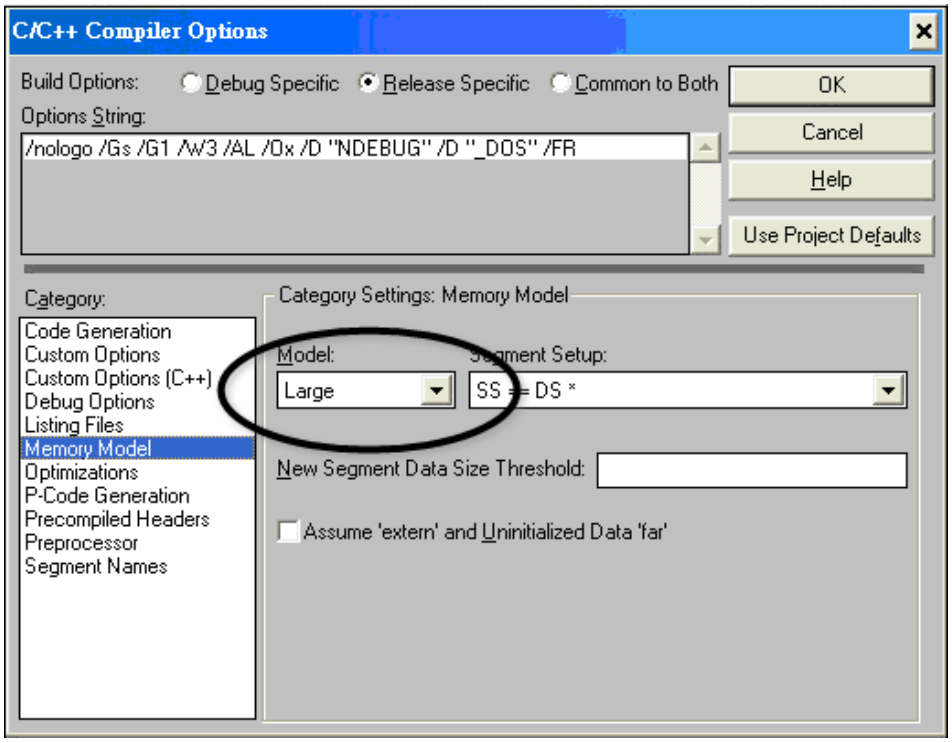
步驟 3: 將使用者程式與所需的函式庫加入到專案中。



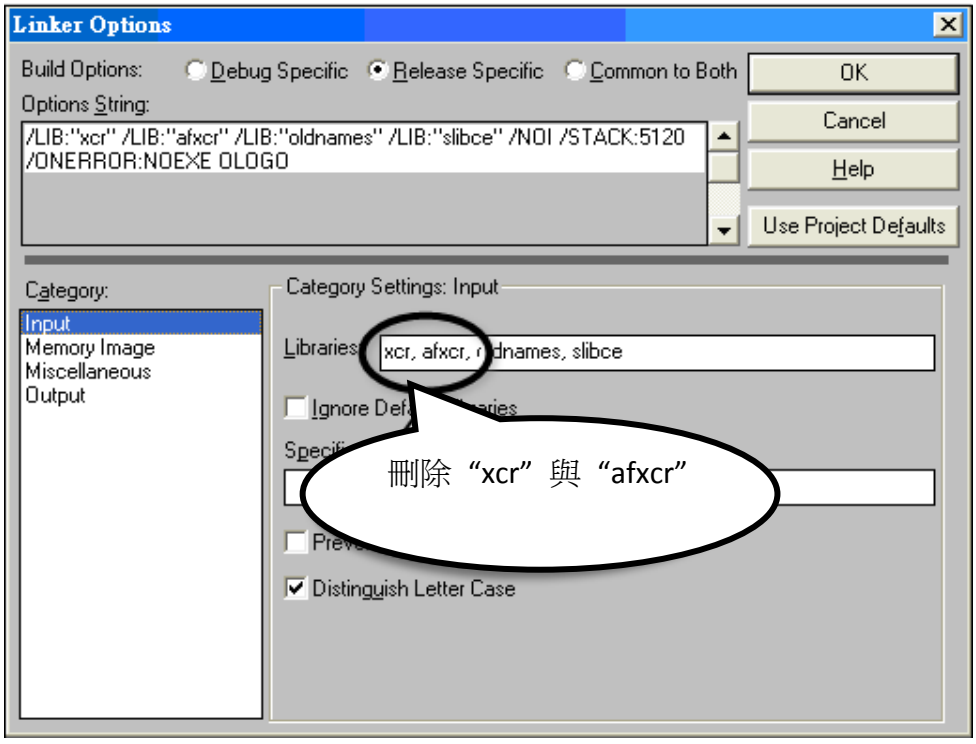
步驟 4: 於編譯器選項中設定 “Code Generation”。



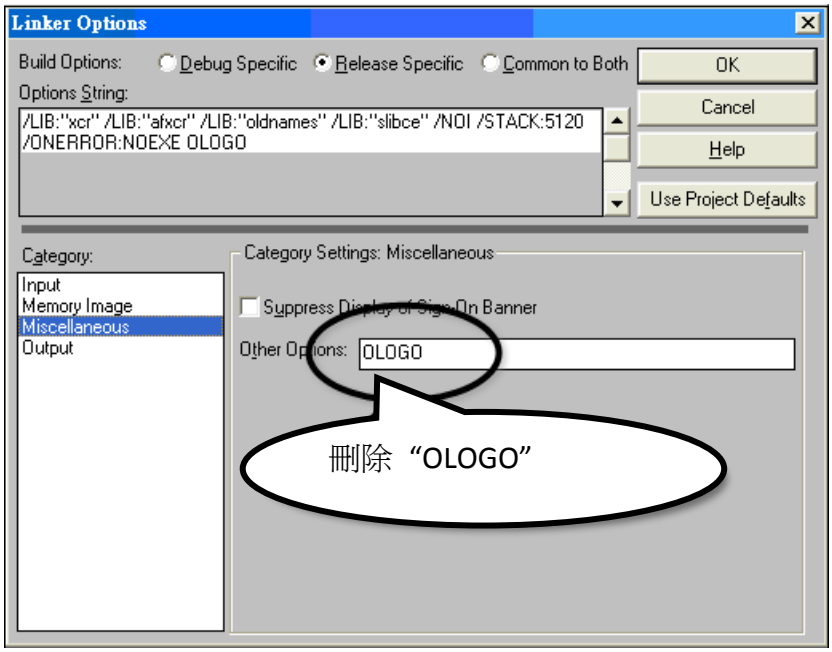
步驟 5: 變更記憶體模式為 “Large” (用於 uPAC5000.lib)。



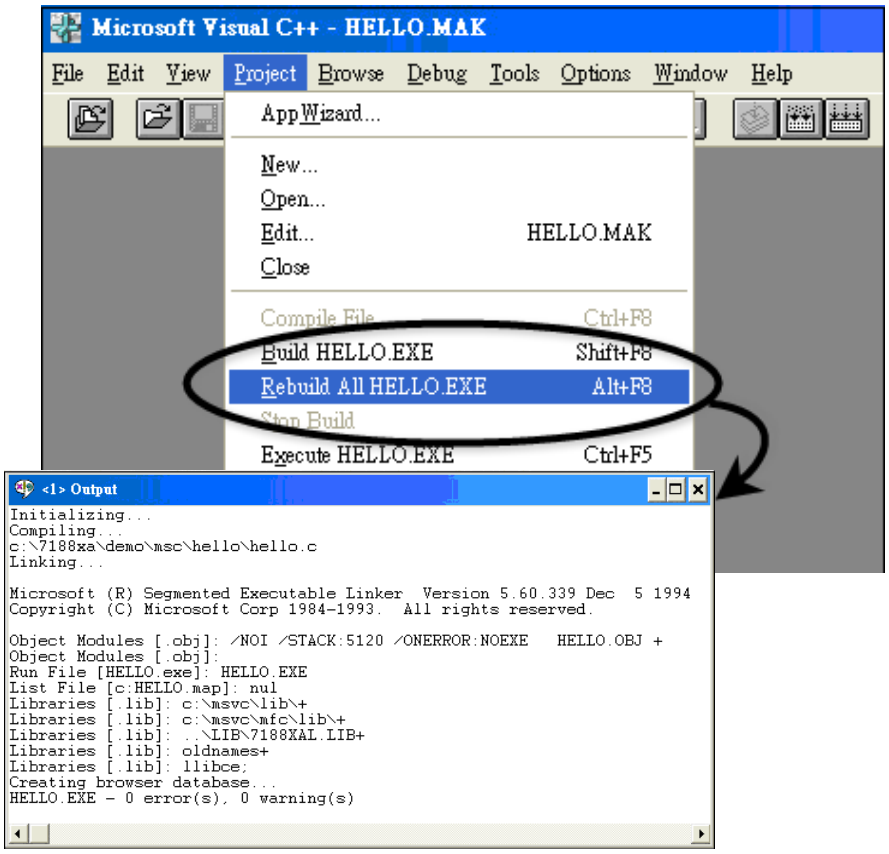
步驟 6: 於 Input 類別，刪除 “xcr，afxcr” 。



步驟 7: 於 miscellancous 類別，刪除 “OLOGO” 選項。



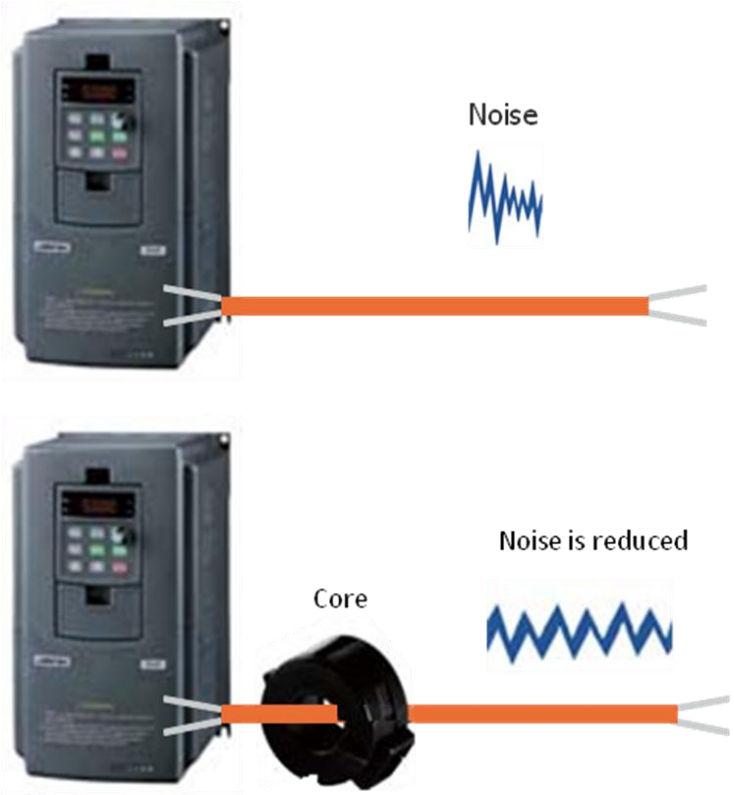
步驟 8: 重建（Rebuild）專案。



附錄 D. 磁環應用與接線方式

磁環(鐵氧體)能有效地用來抑制電磁干擾(EMI)及抗雜訊，主要用於通訊介面的纜線，如 RS-232、RS-422、RS-485、CAN Bus、FRNET、PROFIBUS、Ethernet…等，更可用於 AC/DC 電源端的纜線。

下圖將介紹磁環如何抑制雜訊:



下圖是在 CAN Bus 端使用磁環的接線方式:

注意:在正常的通訊下，若過度的使用磁環則會造成通訊異常的情況發生。