

Penbex OS PDA 程式設計與我

初版
上冊

王銘德 著

目錄

< 上冊 >.....	7
--- 基礎篇 ---.....	8
一、 怎麼在 PC 上開發 PDA 程式?.....	9
1. 前言.....	9
2. 使用 Penbex OS 的裝置有哪些?.....	10
3. PDA 硬體特性.....	11
4. Penbex OS 基本架構.....	13
5. Penbex SDK 包含哪些東西?.....	15
6. 如何安裝 Penbex SDK 軟體開發工具?.....	18
7. 如何使用 PDA 模擬器?.....	20
8. 如何使用 GNU Add-in 自動生成 pbx 執行檔?.....	26
9. 移植除錯小秘訣.....	30
10. 練習題.....	31
二、 我的第一支 PDA 程式.....	32
1. 前言.....	32
2. 又是拿 Hello World 程式為例.....	35
3. 程式基本結構.....	38
4. 什麼是 Container?.....	39
5. 事件迴路.....	41
6. 有哪些 header 檔能用?.....	42
7. 有哪些事件訊息種類?.....	43
8. Penbex 對全區域變數的規定及特性.....	44
9. 如何在 LCD 螢幕上顯示字串?.....	45
10. 練習題.....	46
三、 Penbex 提供哪些 GUI 物件?.....	47
1. 前言.....	47
2. 按鈕物件.....	48
3. 列表物件.....	50
4. 文字輸出入物件.....	52
5. 表格物件.....	55
6. 對話框物件.....	58
7. 樹狀物件.....	60
8. 白板物件.....	62

9.	資料庫欄位物件.....	64
10.	練習題.....	67
四、	進階程式技巧.....	68
1.	前言.....	68
2.	記憶體配置不再造成當機!.....	70
3.	記憶體 Handler 又是什麼?.....	71
4.	多視窗環境怎麼實現?.....	74
5.	什麼時候用 Popup Container?.....	75
6.	PDA 能播放音樂嗎?.....	77
7.	如何製作圖案物件?.....	81
8.	我能修改桌面程式嗎?.....	83
9.	什麼是執行狀態?.....	85
10.	如何做自己的專屬的 ROM image?.....	89
11.	模擬器使用 C 硬碟檔案的小技巧.....	91
12.	Cut 與 Paste	92
13.	練習題.....	95
五、	我能用 ANSI C 標準副程式庫嗎?.....	96
1.	前言.....	96
2.	Penbex 不需再外掛 Math 副程式庫	97
3.	整數運算相關函式.....	98
4.	收尋與排序.....	99
5.	如何產生亂數?.....	100
6.	有字串及記憶體相關函式真方便.....	101
7.	資料間的轉換.....	103
8.	字的分類檢測及轉換.....	104
9.	如何使用 ANSI C 記憶體管理函式?.....	106
10.	如何使用檔案?.....	107
11.	ANSI C 與時間有關的函式.....	109
12.	Assertion 是甚麼?.....	110
13.	其他 Penbex 支援的 ANSI C 函式.....	112
14.	練習題.....	113
	--- 遊戲篇 ---.....	114
六、	遊戲程式設計範例.....	115
1.	前言.....	115
2.	遊戲程式設計基本原理.....	116
3.	怎麼抓到筆觸 LCD 螢幕的位置?.....	117

4. Timer 的應用?	119
5. 影像轉換工具	120
6. 繪圖相關 APIs	122
7. 電子寵物貓範例	123
8. 如何抓到按鍵訊號?	126
9. 如何畫會動的球?	127
10. 如何控制球拍?	128
11. 如何產生聲音?	130
12. 如何利用紅外線對打?	133
13. 練習題	135
--- 參考資料 ---	136
--- 下冊預告 ---	138

圖表索引

表 1.3.1	DRAGONBALL CPU 規格對照表.....	11
圖 1.3.2	PDA 硬體架構.....	12
圖 1.4.1	PENBEX OS 軟體架構.....	13
圖 1.4.2	PENBEX OS 軟體模組.....	14
圖 1.5.1	PENBEX 的 SDK 開發環境.....	17
圖 1.6.1	解壓縮後的 SDK 目錄.....	18
圖 1.6.2	安裝 SDK Disk1 目錄.....	19
圖 1.6.3	PENBEX SDK GNU 環境.....	19
圖 1.7.1	PENBEX PDA 模擬器.....	20
圖 1.7.2	模擬器 F7 測試畫面.....	22
圖 1.7.3	修改模擬器外觀圖檔.....	22
圖 1.7.4	修改模擬器資源檔.....	23
圖 1.7.5	IDS_BMP_INFO.....	24
圖 1.7.6	IDS_BTN_INFO.....	24
圖 1.7.7	IDS_BTN_TIPS.....	25
圖 1.8.1	GNU ADD-IN 功能.....	26
圖 1.8.2	HELLO 程式原本目錄.....	27
圖 1.8.3	HELLO 轉譯後目錄.....	28
圖 2.1.1	HELLO WORLD 程式畫面.....	35
圖 2.3.1	加入新程式 HELLO.C.....	38
圖 2.4.1	SETTABCONTENTRY 結果.....	39
圖 2.4.2	SETTITLETABCONTENTRY 結果.....	40
圖 2.6.1	事件迴路示意圖.....	41
圖 2.9.1	顯示字串.....	45
圖 3.2.1	按鈕物件.....	48
圖 3.3.1	列表物件.....	50
圖 3.4.1	編輯器物件.....	52
圖 3.5.1	表格物件.....	55
圖 3.6.1	對話框物件.....	58
圖 3.6.2	FLOATBOX 的畫面.....	59
圖 3.7.1	樹狀物件.....	60
圖 3.8.1	白板物件.....	62
圖 3.9.1	資料庫欄位物件.....	64

圖 4.4.1	TABCONTAINER 範例.....	74
圖 4.8.1	內建桌面操作畫面.....	83
圖 4.8.2	XB.EXE ROM 燒寫器	84
圖 4.10.1	PU.EXE 執行畫面.....	90
圖 5.8.1	字的分類示意圖	104
圖 5.12.1	ASSERTION 訊息視窗.....	111
圖 6.5.1	程式貼圖範例.....	120
圖 6.5.2	BMP2ARRAYPRO.EXE	120
圖 6.7.1	電子寵物.....	123
圖 6.11.1	簡易電子琴.....	132

< 上冊 >

--- 基礎篇 ---

一、怎麼在 PC 上開發 PDA 程式？

1. 前言

PDA (Personal Digital Assistant)個人數位助理原本是一種內嵌式裝置，就是所謂 embedded system 的一種。在還沒人做到可以在 PC 上開發 PDA 程式之前，類似 PDA 的產品軟體都是事先開發好燒錄在 ROM 裡，所以功能是固定的。類似股票機、電子字典、電子雞、甚至於電子計算機等，都屬於不能再開發程式的內嵌式裝置。當然內嵌式裝置還有很多包括冷氣機的控制器、飛彈的控制中心、甚至於飛機的起落架本身都屬於一個內嵌式系統，但我們就不在本書中介紹。我只介紹採用互慧科技 Penbex 掌天作業系統所開發出來的 PDA，它能讓程式開發者在 PC 上使用微軟 Visual C6.0 開發 PDA 上的應用程式。這是一種方法，這種方法改變了 PDA 的命運，它不再只是一台 PDA，我們甚至可以稱它為一台”行動個人電腦”。

在介紹如何撰寫 Penbex OS PDA 程式之前，本章節主要介紹”如何開始”，包括第二節介紹如何準備一台 PDA，一台裝有 Penbex OS 的 PDA。未來還有哪些設備會使用 Penbex OS，而 Penbex OS 的下一步又會如何走？第三節介紹 PDA 的硬體的特性，以及一些基本操作之說明。其次，從軟體的角度來介紹 PDA 作業系統的架構和原理，了解 OS 的特性才有可能將軟體發揮的淋漓盡致。

第五節開始，我將介紹 Penbex SDK 軟體開發工具(Software Development Kit)，以及如何在 PC 上安裝 Penbex SDK。SDK 又包含哪些部分，如何利用 VC (Visual C) 環境編輯程式且直接在 PDA 模擬器裡執行，反覆進行除錯編譯等直到功能運作無誤為止。最後階段則是透過 GNU 68K 交叉編譯器，將程式轉譯成適合 PDA DragonBall CPU 指令集的執行檔，在 Penbex 環境以”pbx”為下載執行檔的副檔名，它成了能在 PC 上開發 PDA 應用程式的關鍵所在。

由於在 PC 上使用的是 Intel x86 CPU，而在 PDA 上目前最常用的是 DragonBall CPU (它是以 68000 (68K)為核心的 CPU，執行與 68K 相同的指令集)，因此在 x86 開發的程式要直接轉譯成 68K 程式而不發生錯誤，除了整個開發環境必須設計的非常巧妙以外，程式開發者對其中的差異性也必須了解，第九節將介紹其中的不同，成為移植到 PDA 時除錯的小秘訣。

2. 使用 Penbex OS 的裝置有哪些？

目前 Penbex OS 主推解析度 160x160 LCD 螢幕且只給 DragonBall CPU 的 PDA 使用。主要原因是一個作業系統，不只要考慮技術問題，更需要考慮在它上面開發的應用軟體多寡的問題。如果同時支援太多種規格，開發環境無法統一，跟隨而來的就會是大災難；例如，(1)使用者下載程式時還要區分該程式是否可以在他的 PDA 執行，這種判斷對 PDA 的使用者來說太需要專業技術了。(2)對程式開發者來說，太多種情況需要事先考慮，程式就變得相當複雜。(3)以上兩種狀況，產生的就是一連串的安裝不當，程式不對，造成當機事件層出不窮。所以所有使用 Penbex OS 的 PDA 都採用類似的規格，包括相容同系列的 CPU，相同的螢幕解析度，相同的 RS-232 與紅外線介面，相同提供上、下、左、右、A、B 及電源等七個以上的按鈕(英文稱為”Hard Key”)，以及 LCD 下方有類似的銘版，至少有”Home”、“選單”、“軟鍵盤”、“返回”、“幫助”以及”全域搜尋”等按鈕(英文稱為”Soft Key”)，以圖標表示亦可。

目前已有中環、互億、廣傑、陸仕、精英、慧達(博達的子公司)等公司的 PDA 採用互慧科技 Penbex OS 為其作業系統，除內銷國內市場以外，還銷售到大陸、亞洲、美國、中南美、及歐洲地區國家。用戶或軟體開發者已能經由這些製造商的行銷管道從市面上買的到 Penbex OS 的 PDA。只要是相同版本之作業系統，我們要求所有廠商做到執行程式相容，且周邊設備相容，保障所有軟體開發商的研發權益及減少重複開發或移植所需的成本。當然廠商在相容的情況下，亦能增加其功能或修改畫面做市場區隔。例如有些機種內建 MP3 播放器，有些機種內建傳呼機功能，有些機種將 CF 卡擴充槽作為基本配備，增加了 PDA 的周邊能力及擴充性等等。

未來 Penbex OS 也會朝向另一個產品線邁進，那就是所謂的”Smart Phone”，一種結合 PDA 與無線通訊的手持設備。由於時機尚未成熟，不在此多做介紹，但既然是同一套作業系統，將提供相同的開發環境以保障軟體開發者的投資，以及用戶程式使用習慣和軟體及資料得延續性。

3. PDA 硬體特性

以下是目前一台標準的 Penbex OS PDA 所採用的硬體基本架構：

- CPU 採用 Motorola DragonBall(MC68EV328) 16.6Mhz 或 (MC68VZ328) 33Mhz
- 最大 4MB 唯讀記憶體(ROM)
- 最大 8MB 隨機存取記憶體(RAM)
- 160x160 像素的 LCD 觸控螢幕, 4 色灰階
- 銘版區提供英文手寫辨識以及 6 個軟按鍵
- 至少 7 個硬按鍵, 包括上、下、左、右、A、B、及電源鍵
- 1 個 RS232 埠
- 1 個紅外線埠
- 1 支觸控筆
- 超過 50 小時電源供應器(AAA 電池, 或充電電池)
- 1 個蜂鳴器或喇叭(選項)
- 1 個麥克風(選項)
- Compact Flash 擴充槽(選項)
- MP3 播放器(選項)

微處理器	CPU 速度 MHz	匯流排介面 Bits	運算效能 MIPS	運作電壓 V	IC 包裝方式
MC68EZ328	20	24 addr/16 data	3.4 @ 20 MHz	2.7 - 3.3	100 TQFP
MC68VZ328	33	24 addr/16 data	5.4 @ 33 MHz	3.0 - 3.6	144 TQFP or PBGA

表 1.3.1 DRAGONBALL CPU 規格對照表

如圖 1.3.2 所示, 一台 PDA 基本上提供按鍵和觸控螢幕以及銘版區來讓使用者對 PDA 進行輸入或控制。而系統聲音(包括鬧鈴聲及筆觸聲)、螢幕顯示當成 PDA 的主要輸出。而對周邊的介面則以有線的 RS232 串接埠(又稱為 UART)、無限的紅外線埠(又稱為 IrDA)為主、和可能有的 Compact Flash 卡擴充槽。只要是 PCRS232 能接的東西大部分 PDA 都能接, 例如條碼掃描器、衛星定位系統、數據機、讀卡機等等。甚至於把 PDA RS232 接到 UNIX 電腦或路由器等 COM port 當成終端機使用

也未嘗不可(但 PDA 上需安裝 vt100 模擬軟體), 而紅外線可能是 PDA 最重要的標準通訊介面, 可與大多數的筆記型電腦、手機、其他 PDA、或所有紅外線裝置對傳資料。

有幾個 PDA 的特性與 PC(筆記型電腦)或其他手持裝置最大不同的地方, 值得在本節一提: 一、PDA 一次充電使用時間可長達兩週至兩個月之久, 這是同樣功能的手提電腦(2 至 3 個小時電池的運作時間)所望塵莫及。二、PDA 的螢幕很小, CPU 運算能力也很弱, 記憶體只有 1 至 8MB, I/O 的緩衝區也小的可憐, 所以撰寫軟體或規劃資料量時都需要小心謹慎。三、他沒有真正的硬碟, 只靠 RAM disk, NAND gate, 或 Compact Flash 記憶卡當成它的儲存區。由於 Penbex OS 支援檔案系統, 所以相對就有 A:\、B:\、C:\ 目錄能使用, 當系統沒有 NAND gate 時, B:\ 就由 Compact Flash 記憶卡所取代, 如果也沒有 CF 記憶卡時, 該 PDA 就只剩下 A:\ 目錄。但它的 OS 基本上就有上網或與 PC 對傳資料或同步的能力, 所以考慮基本資料要常常或自動備份到伺服器或個人電腦 PC 上顯得非常重要。四、由於全屏手寫辨識已經是 Penbex OS 的基本中英文輸入, 非不得已沒有叫出軟體鍵盤或外接鍵盤做輸入得必要。五、紅外線自動接收外來的資料或檔案且直接呼叫相對應程式回應之, 成為非常好用的機制, 不管是名片交換或是檔案下載, 利用紅外線無線傳輸成為 PDA 最重要的輸出入介面。由於 PDA 是一種行動裝置(mobile device), 紅外線無線傳輸相對的就比在 PC 上更顯其重要且常被使用。

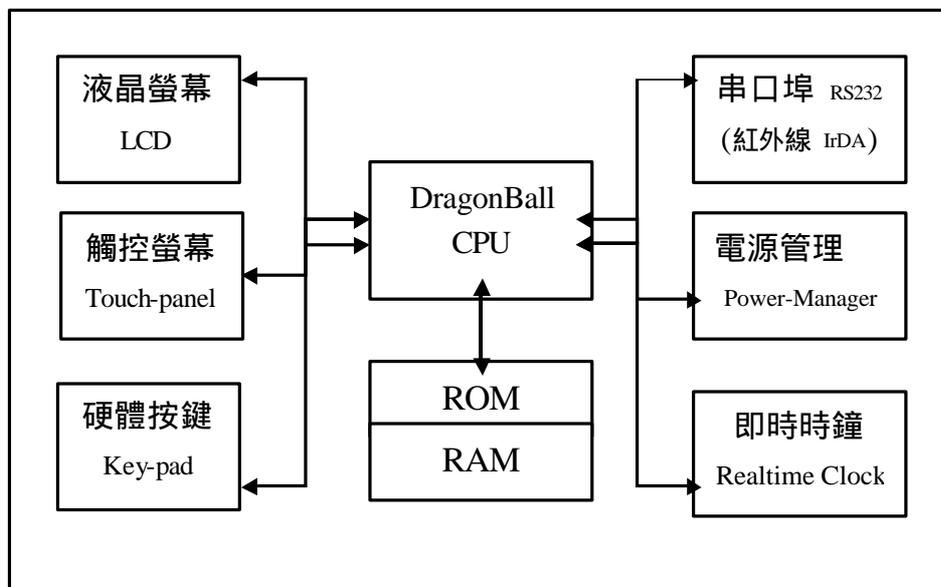


圖 1.3.2 PDA 硬體架構

4. Penbex OS 基本架構

Penbex OS 是專為行動手持裝置, 像 PDA 或 smart phone 等 mobile IA 而設計。在 PDA 還不流行使用 OS 的觀念時, 所有的 PDA 都是直接採用內嵌式系統的方式針對單一功能撰寫程式, 且不提供開發環境給第三方軟體商能在該機器上撰寫軟體。也就是該 PDA 除非送回原廠升級, 否則功能永遠是固定的; 若想增加新功能, 換機可能是唯一的選擇。但在未來的 PDA 和手機市場, 下載軟體、資料交換或無線上網成為基本功能時, 沒有使用 OS 的 PDA 或手機就幾乎無法生存, 因此將 OS 與應用程式分開, 並提供軟體開發工具(所謂的 SDK)在 PC 上開發軟體, 當然其中包括內建於 ROM 的常用軟體(對純 PDA 用戶而言就是計算機、行事曆、電話簿、記事本、世界鐘等 ..), 還有可以下載到 RAM disk 的其他第三方(3rd party)軟體公司所開發的應用軟體。這個道理跟個人電腦使用微軟的視窗作業系統, 每台 PC 安裝視窗作業系統才能開機使用, 且附屬應用軟體已內建有小畫家、小算盤、記事本、及 IE 瀏覽器是一樣, 而其他軟體公司能利用微軟的 Visual C 或 Visual Basic 等軟體開發工具在 PC 上開發應用軟體, 讓使用者自行安裝後使用。

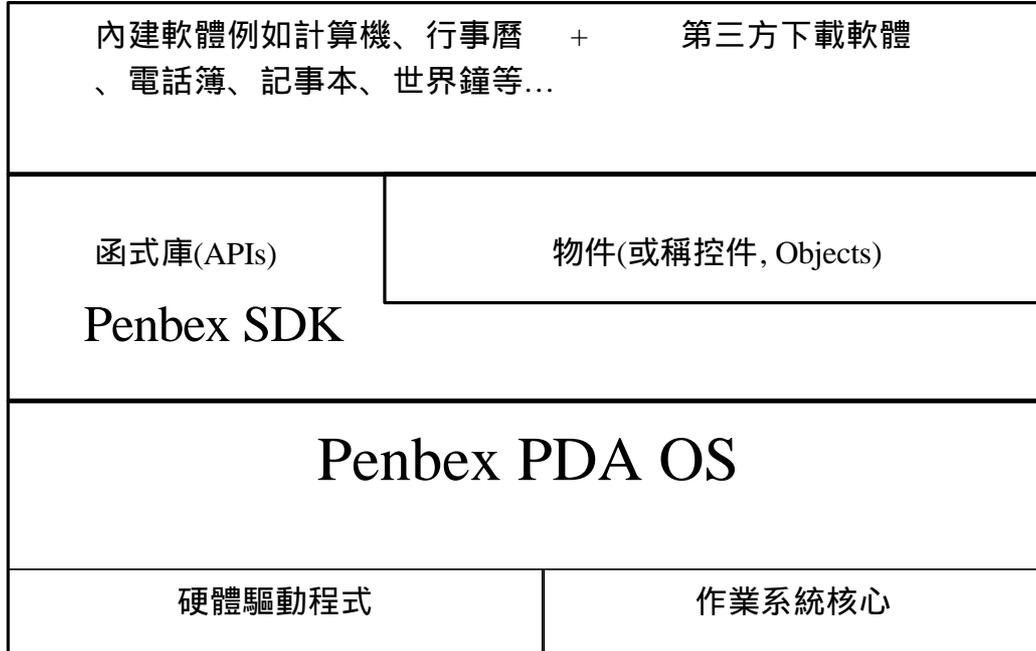


圖 1.4.1 PENBEX OS 軟體架構

但畢竟 PDA 和 PC 的 OS 架構還是有所不同。第一、PDA 的資源有限,但又要做到幾乎和 PC 一樣的功能,很多東西必須取捨。例如,PDA CPU 和記憶體資源都不足,所以不適合使用多工作業系統;螢幕太小,使用者介面也不適合使用多視窗畫面。第二、Penbex OS PDA 的程式是在 PC 上利用 Visual C 開發與除錯,並利用 PDA 模擬器在 PC 上作測試完成後再經過 GNU 68K 的交叉編譯器轉譯而成,不像 PC 程式就在 PC 上直接開發。第三、PDA OS 和筆記型電腦一樣,對電源管理要特別考慮,甚至於連內建程式都要考慮處理斷電前的問題。

下載應用軟體						內建應用軟體				
850 左右個 APIs								系統用 APIs		
電源管理	檔案管理	繪圖管理	應用軟體管理	觸筆管理	按鍵管理	事件管理	資源管理	語音管理	TCP/IP 管理	紅外線管理
Micro Kernel: 包含 task, timer, interrupt, 以及記憶體管理										
驅動程式						硬體平台: DragonBall/ 其他 CPUs				

圖 1.4.2 PENBEX OS 軟體模組

5. Penbex SDK 包含哪些東西？

Penbex SDK 包含五大部分：以 Visual C 為工具之開發環境、PDA 軟體模擬器、GNU 開發環境、程式範例代碼、以及其他工具等；依次說明如下：(如圖 1.5.1)

一、以 Visual C 為工具之開發環境：

以一個叫”PenbexOS.dsw”的專案工作區(Project Workspace)的方式呈現。內含一個專案名為”PenbexOS SDK files”，提供完整 Penbex OS PDA 程式的開發環境所需要的檔案：包括”Include”目錄中所有的檔頭文件(header files), “PDA AP”目錄中的範例程式源代碼, “Win”目錄下則包含模擬器本身的程式碼和資源檔與文件, “SDK SYSTEM”目錄下包含所有控制台(Console)程式的源代碼以及桌面(Desktop)程式本身的所有源代碼。經過每次重新編譯(或按 F7), 會產生一個帶有所有開發中程式的 PDA 模擬器(PenbexOSEmu.exe), 亦可直接按(Ctrl+F5)重新編譯後啟動該模擬器。由於整個 Penbex OS 的模擬器(包含開發中的 PDA 程式)是 Intel x86 執行碼的視窗程式, 所以能隨著開發中的程式能做原始碼除錯(source code level debugging), 這點和 Palm OS 的 Co-pilot 模擬器(執行 DragonBall 指令集)做法截然不同。

由於微軟的 Visual C (VC)已經是非常好的 C 語言開發環境, 也是非常多人已熟悉的開發工具, 加上 Penbex SDK 就是架構在 VC 環境上, 所以開發 Penbex OS PDA 的程式就能完全利用到 VC 的編輯, 除錯, 編譯等好處, 例如快捷鍵, 代碼管理, 設斷點, 查看變數內容, 呼叫堆疊, 代碼檢查, 步進除錯等。我想最重要的是 Penbex OS PDA 程式開發者不需要再學一套新的開發環境。

二、PDA 軟體模擬器：

Penbex OS PDA 模擬器是直接從 VC 的開發環境編譯而成, 讓 PDA 程式可以在沒有 PDA 實體的環境下, 也能進行開發及測試。當程式在模擬器裡執行都沒問題時, 再經過 GNU 交叉編譯器將該 PDA 程式轉譯成 PDA 實體能接受的執行檔, 也就是 DragonBall CPU 所能執行的指令, 在 Penbex OS 以 pbx 為副檔名的可下載執行檔, 進一步再做最後的實機測試。由於 PDA 硬體本身雖然有作業系統, 但畢竟它還是類似”內嵌式系統”, 非常難在 PDA 上直接開發程式及除錯, 沒有在 PC 上的模擬器來做驗證, 讓 PDA 製造商以外的軟體公司或個人開發者來開發外掛軟體, 幾乎是比登天還難。模擬器本身還有許多功能在本章第七節會做更詳細的介紹。

三、GNU 開發環境：

由於 PDA 實體是採用 Motorola DragonBall 的 CPU，它採用 68000 指令集，與個人電腦 Intel x86 CPU 的指令集完全不同，若想再 PC 上開發 PDA 的程式，最終還是要把程式碼轉譯成 68k 的執行檔。因此 Penbex SDK 內含一套 GNU 標準的 68k 轉譯器，加上轉譯成 Penbex 的執行檔所需要的東西，例如給 GNU 用有關 Penbex 環境的檔頭文件及副程式庫，當然為了能讓開發者馬上使用，GNU 編譯所需要的工具如編譯器(compiler)，連接器(linker)，製作器(make)等，皆一並提供。當然還有 pbx 產生器(makepbx)也屬於這個部分，讓 makefile 產生最後的 Penbex 執行檔。

如果沒有 VC 環境的人，只能單獨安裝 Penbex SDK 的 GNU 開發環境。GNU 是一套在 Unix 環境下的產物，所以在視窗環境裡只能在 DOS 視窗裡執行，而且使用者還需要了解如何使用所謂的 makefile 來製作編譯流程，以下指令的方式來進行轉譯和開發過程。不過還好，Penbex SDK 也替 GNU 用戶準備了同一套程式範例源代碼及相對應的 makefile 檔 只要在 DOS 視窗裡，於 C:\Penbex\pbxgsdk\usr\src 的目錄下輸入”makeall”，即可自動完成所有範例程式執行檔的產生。

對有安裝 VC 環境的開發者而言，可以不必學習使用 GNU 或 DOS 的指令了。在 VC 的工具條(Tools bar)裡可客戶化(Customize)加入一個 Penbex Developer Studio Add-in。工具條上就會有一個 Build Penbex Project 的圖標(icon)，能自動替 PDA AP 目錄下的每一個程式目錄製作所需之 makefile 檔，而且自動做出 pbx 檔來。所有的 DOS 下的轉譯過程，也會在 VC 輸出視窗(按 Alt-2 快捷鍵)的 Macro 子視窗中顯示出來。對不熟悉 GNU 及 DOS 環境的人，真是一大福音。

四、程式範例代碼：

為了加快讓程式開發者了解 Penbex 程式設計原理，對初學者提供了 Container, TabContainer, Button, Date, Editor, List and Combo, Table, Menu, Message Box, Timer, Progress, Tree, and Whiteboard 等物件相關的範例源代碼。皆以每一支範例程式一個目錄的方式擺在 VC 環境下 PenbexOS 專案的 PDA AP 目錄下。

對進階程式開發者想了解具有挑戰性的程式代碼與系統相關的函式時，可以參考同一個專案檔案 SDK SYSTEM 目錄下的 Console 和 Desktop 兩支系統程式源代碼。從中不難了解 Penbex OS 系統的程式精華，以及體會 PDA 超強功能的做法。

五、其他工具:

其他工具有包含: 影像檔轉換器(Bmp2ArrayPro.exe), Flash ROM 燒寫器 (XB.exe), 以及 ROM image 製作器(PU.exe), (第四章第十節會介紹)等。

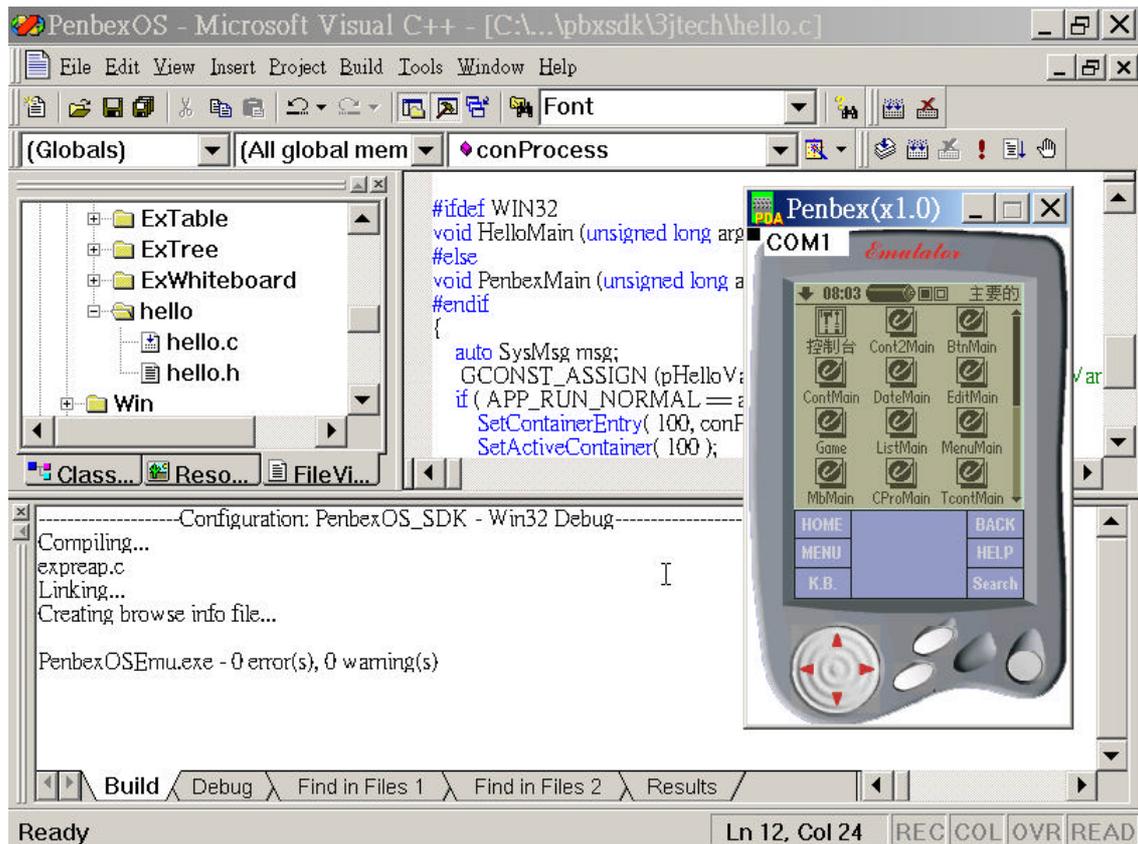


圖 1.5.1 PENBEX 的 SDK 開發環境

6. 如何安裝 Penbex SDK 軟體開發工具？

先準備好一台已安裝 Win98 或 WinMe 的個人電腦 (不能是 NT 或 Win2000 作業系統)，而且已經安裝微軟的 Visual C++ 6.0 (需個人版以上，不含個人版)。

請先從網路上 <http://www.penbex.com.tw/developer> 下載 Disk1.zip 到 Disk9.zip 檔，將它們解壓縮在同一個目錄下(參見圖 1.6.1)。再按滑鼠左鍵兩下，執行 Disk1 目錄下的 Setup.exe (參見圖 1.6.2)，開始進行 Penbex SDK 的安裝，並且預設的安裝路徑為 “c:\Penbex”。

安裝完畢並且重新開機之後，在 “c:\Penbex” 會多出兩個子目錄：“Pbxsdk” 及 “Pbxgsdk”。

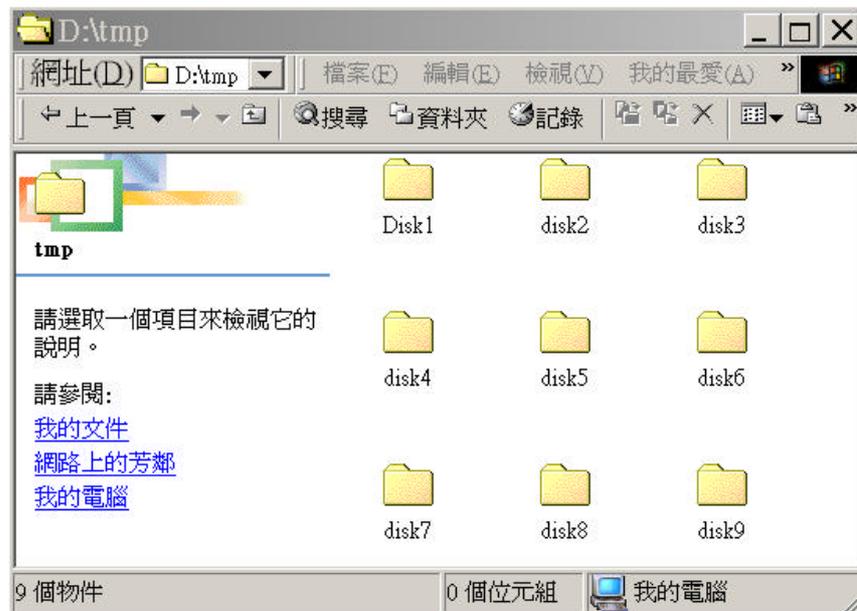


圖 1.6.1 解壓縮後的 SDK 目錄

值得注意的是，C:\penbex\pbxgsdk 目錄下是 GNU 的開發環境，如果機器沒有 VC 只能安裝 GNU 環境，再 DOS Command Mode 視窗下進到 usr\src 目錄下打指令 makeall，會自動做出所有程式範例目錄下的每一個 pbx 執行檔。您也可以只利用 GNU 環境來開發 PDA 程式，只可惜沒有 VC 的除錯環境可以結合模擬器使用。但很快 Penbex 版的 Co-pilot 仿真器(模擬 DragonBall CPU 的 PDA)也會推出，喜歡 GNU 的您就可以直接讓 pbx 檔再 Co-pilot 裡執行。

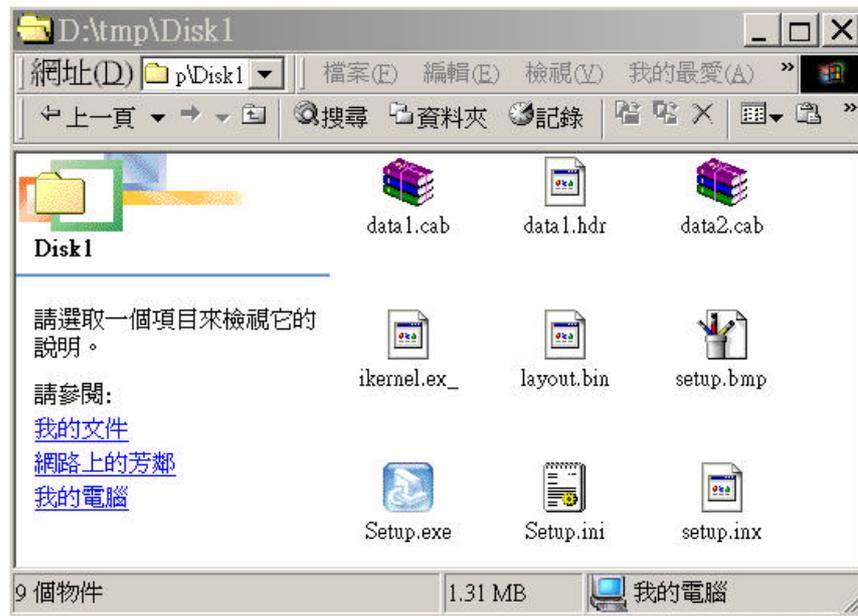


圖 1.6.2 安裝 SDK Disk1 目錄

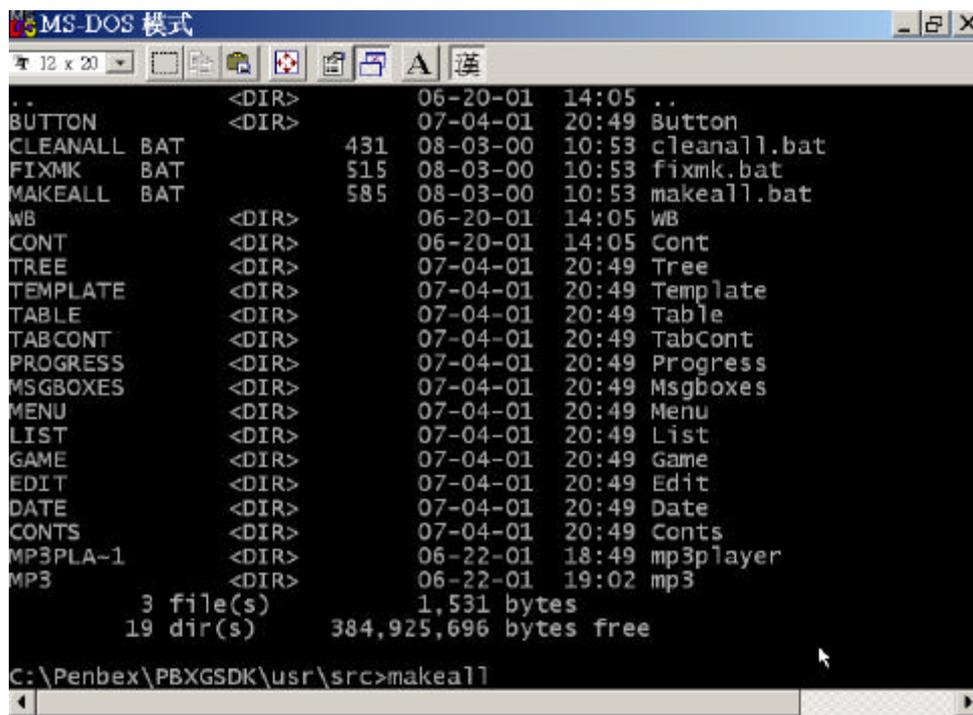


圖 1.6.3 PENBEX SDK GNU 環境

7. 如何使用 PDA 模擬器？

PDA 模擬器對於 PDA 程式開發者是非常必要的工具,除了及時測試軟體功能及配合 VC 環境除錯以外,模擬器本身也有自我測試的方法,以及可以改變外觀以配合不同廠牌之 PDA 而修改。本身幾乎對 PDA 實體做了 90%以上的模擬,其中包括所有系統軟體功能,LCD 觸控螢幕,背光,系統聲音,系統時間與日期,RS232 的 COM port,紅外線,以及透過 RS232 撥接數據機後連上的網路及 Internet, 銘版, 按鍵, 電源鍵等。唯一沒有模擬的是,CPU 速度,以及 PDA 電池等,也就是說 PC 上模擬器的運算是以 Intel x86 的指令與速度在執行,相對 PDA 68K 的 DragonBall 來講,速度快上百倍。至於模擬器的電池顯示永遠是滿格,所以軟體無法在模擬器上測試碰到沒電的狀況。因此,大部分軟體最後還是需要安裝到實際機器上測試才能確定是否完全正確。



圖 1.7.1 PENBEX PDA 模擬器

若要在 PDA 模擬器上開發及直接測試網路軟體,必須先將 PC 從 COM1 或 COM2 接上一台數據機(56Kdps 以下),與申請一個網路 ISP 的撥接帳號。PDA 模擬器裡可以設定撥接帳號及密碼,和 ISP 電話與數據機設定,利用正常撥接方式 PPP 上網,當 PDA 模擬器連上 Internet,所有 PDA 上所開發的網路軟體就能直接測試,而不須實際 PDA 來運行,也就是說,其實 PDA 模擬器 100% 模擬了實際 PDA 的 RS232 串口,所有 RS232 的周邊,包括條碼機,衛星定位系統,讀卡機等 RS232 裝置,都能在 PDA 模擬器透過 PC 的 COM port 來執行或寫軟體來控制。下冊第八章,我會以

GPS 的接收程式範例來介紹 RS232(又稱 UART)的程式設計。

如何及時擷取模擬器畫面呢? 在制定 PDA 程式操作畫面, 或撰寫手冊時常常需要擷取 LCD 畫面。以下介紹利用模擬器預留的程式測試把螢幕擷取當成外加功能加到你的模擬器中。將以下程式片段取代 C:\penbex\pbxsdk\src\runtest.cpp 檔案裡 void RunT1()的部分, 您可以從 VC 裡工作區(workspace)裡 FileView 下的 Win\WinCCP 目錄下找到 runtest.cpp 檔。

```
void Capture (void)
{
    static    unsigned short usCount = 0;
    char     cPathA[255];
    time_t   lTime;
    struct tm *pTime;

    SetScreenColorLevel(0,255,255,255);
    Time (&lTime);
    pTime = Localtime (&lTime);
    Sprintf (&cPathA[0], "C:\\Penbex\\Pic-%02d-%02d-%02d-%02d-%03hu", \
        pTime->tm_mon+1, pTime->tm_mday, pTime->tm_hour, pTime->tm_min, \
        usCount++);
    Strcat (&cPathA[0], ".bmp");
    SaveScreenToWinBmp(&cPathA[0]);
}

void RunT1 (void)
{
    Capture();
}
```

重新編譯後, 模擬器就有產出螢幕畫面的功能。先執行模擬器後按 F7 鍵, 會產生模擬器本身的測試畫面(如圖 1.7.2), 再點 T1 按鈕, 模擬器就會將當時的 LCD 畫面內容轉為黑白色, 且在 C:\Penbex 目錄下產生以 Pic 開頭的 bmp 影像檔。以當時時間為暫時檔名, 所以不會重複, 若需要你可以再改其檔名。

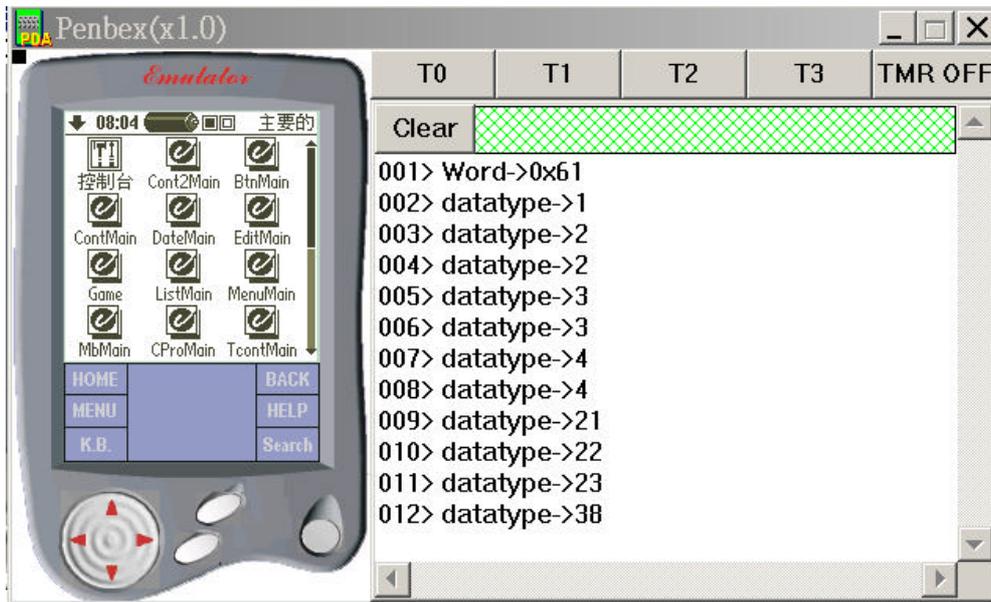


圖 1.7.2 模擬器 F7 測試畫面

如何改變模擬器的外觀(skin)? 也許你會想為 PDA 設計外觀, 我們也曾經舉辦過 PDA 外觀設計大賽, 有非常多的創意作品來自於優秀的 ID 設計師或學生之手。但如何配合市面上 Penbex 的 PDA 或心目中的概念機種做一個模擬器的外觀呢? 以下為您一一介紹。



圖 1.7.3 修改模擬器外觀圖檔

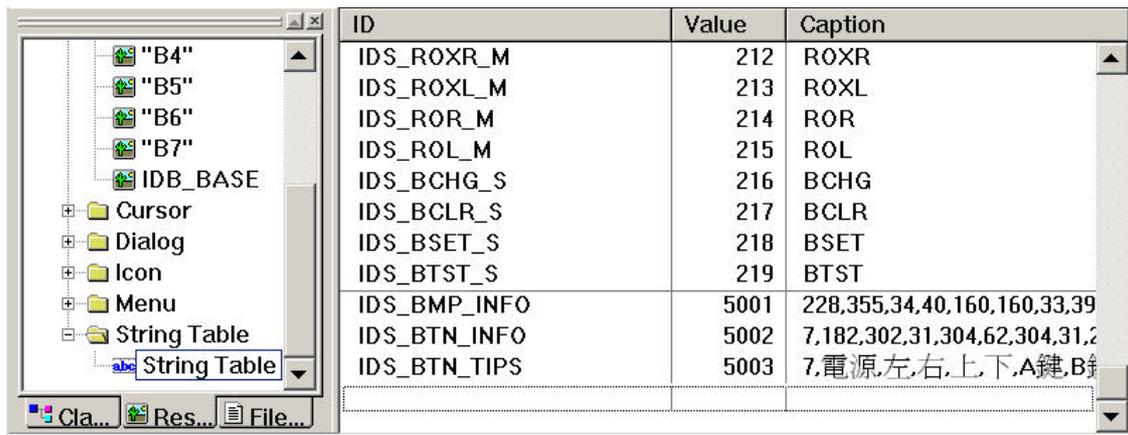


圖 1.7.4 修改模擬器資源檔

第一、(如圖 1.7.3)打開工作區(Workspace)的 ResourceView 視窗, 如圖 1.7.3 打開 VPDA resources 目錄下的 Bitmap 目錄, 修改 IDB_BASE 圖檔會改變整個模擬器外觀。B0 到 B7 則是每個按鍵包括電源的個別三段畫面。目前只用到後兩個畫面, 分別為按鍵”未按”與”按下”的畫面。

第二、如圖 1.7.4 修改 ResourceView 裡的 String Table 目錄下的 String Table 的值, 只需要修改 IDS_BMP_INFO、IDS_BTN_INFO、IDS_BTN_TIPS 三個變數。依次介紹如下:

A、IDS_BMP_INFO: (參見圖 1.7.5)

其 Caption 的內容是放一些有關 IDB_BASE 那張圖的資訊, 也就是底圖的資訊, 共需 14 個: w,h,Lcdx,Lcdy,Lcdw,Lcdh,Tpx,Tpy,Tpx2,Tpy2,Hwx,Hwy,Hwx2,Hwy2

- w,h :代表底圖的長和寬。
- Lcdx,Lcdy :代表 Lcd 螢幕(文字所要顯示的地方)最左上角的位置。
- Lcdw,Lcdh :代表 Lcd 螢幕的長和寬, 現行必須為 160, 160。
- Tpx,Tpy :代表壓力板最左上角的位置。
- Tpx2,Tpy2 :代表壓力板最右下角的位置。
- Hwx,Hwy :代表手寫板最左上角的位置。
- Hwx2,Hwy2 :代表手寫板最右下角的位置。

註: 上述其單位皆為像素。

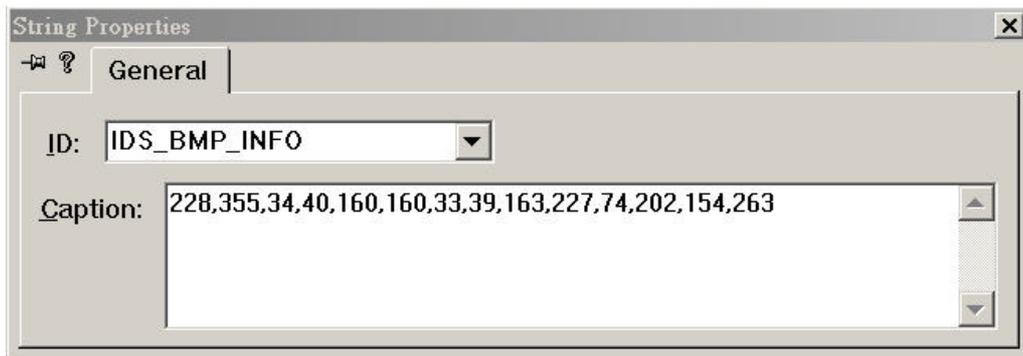


圖 1.7.5 IDS_BMP_INFO

B、IDS_BTN_INFO: (參見圖 1.7.6)

其 Caption 的內容是放一些有關 B0~B6 圖形的資訊，也就是按鍵的的資訊，共需十五個: Total,B0x,B0y,B1x,B1y,B2x,B2y,B3x,B3y,B4x,B4y, B5x,B5y,B6x,B6y

Total :代表共有多少個按鍵。

B0x,B0y~B6x,B6y :分別代表按鍵所要顯示的位置。

註: 上述除 Total 的單位為個，其餘皆為像素。

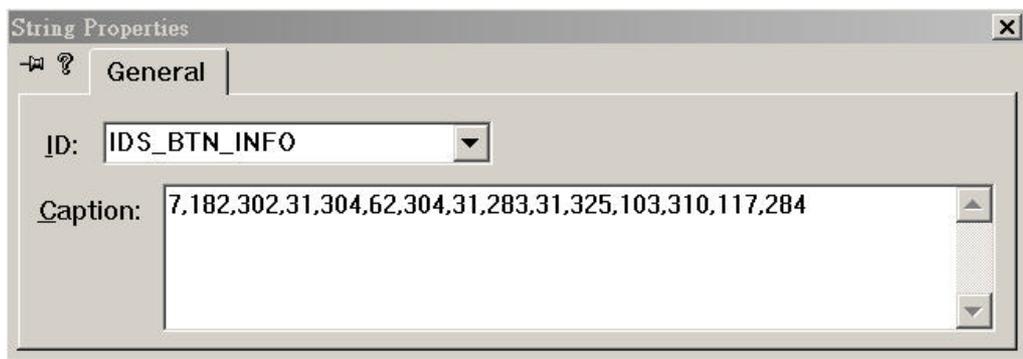


圖 1.7.6 IDS_BTN_INFO

C、IDS_BTN_TIPS : (參見圖 1.7.7)

其 Caption 的內容是放一些有關 B0~b6 圖形的說明，也就是當滑鼠移到按鍵上時所出現的提示資訊，共需八個: Total,B0,B1,B2,B3,B4,B5,B6

Total :代表共有多少個按鍵。
B0~B7 :分別代表按鍵的提示資訊。

註:Total 的單位為個。

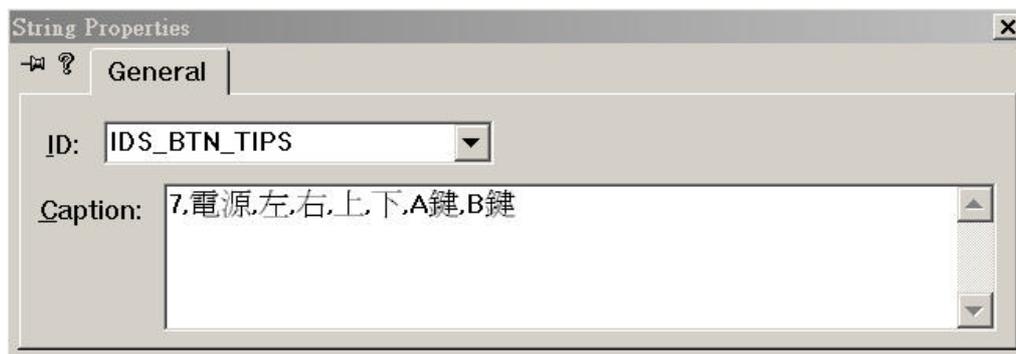


圖 1.7.7 IDS_BTN_TIPS

8. 如何使用 GNU Add-in 自動生成 pbx 執行檔？

開發 PDA 軟體，最終目的還是要在 PDA 上執行，而不是在 PC 的 PDA 模擬器上使用，所以如何快速的產生執行檔變的非常重要。在 Penbex OS 的環境下至少有兩種執行檔存在的方式，一種是與作業系統一起以內建的方式存在唯讀記憶體 (ROM) 裡，另一種則可由外界下載到 PDA 隨機存取記憶體 (RAM) 裡執行。不論是那一種，都可先做成 pbx 執行檔，讓 PDA 使用者來加到 PDA 上使用，就像 PC 能安裝 OS 以外其他軟體一樣有擴充功能，當然在不需要時也可以刪除。對軟體整合商或垂直應用者而言，則可利用工具將 pbx 檔加入 ROM image 裡 (詳見第四章第十節之說明)。好處是該軟體永不流失，即使在沒電源的情況下，它還是像內建軟體一樣永遠存在。

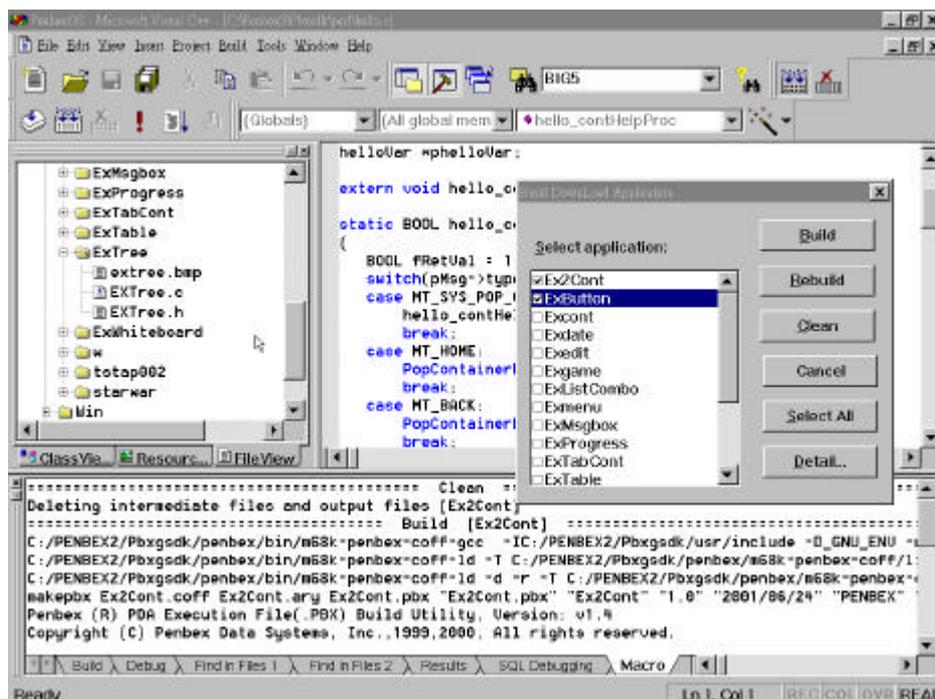


圖 1.8.1 GNU ADD-IN 功能

在圖 1.8.1 中 VC 工具條右上方有兩個按鈕，，左邊按鈕就是用來叫起 "Build Download Application" 的視窗，該視窗是一個 VC 的 Add-in 外掛功能，用來替程式開發者直接叫用 GNU 轉譯器，針對所勾選的程式產生 (build) 其 pbx 執行檔。當轉譯過程中想放棄時，可利用右邊的中止鈕中止之。

“Build Download Application” 視窗中”Select Application”的項目是相對應程式碼視窗中的”PDA AP”目錄下每一個程式的目錄名自動產生。因此需要開發新程式時，就從”PDA AP”目錄下使用一個新目錄，把目錄名稱定為未來 pbx 檔想要的名稱即可。例如如果該目錄名稱為”電子書”，最終產生的執行檔名則為”電子書.pbx”。當然，若臨時想產生另一個 pbx 檔名，也可將該目錄名更改後，再重新啟動 Add-in，新的選項則自動產生。當你勾選其中之一或全部(Select All)後，再按(Build)製造或(Rebuild)再製造，VC 的結果輸出視窗(Output Window)就會產生一個”Macro”新視窗(如圖 1.8.1 所示)，點開該視窗即可看到整個 GNU 轉譯的過程，成功與否？或有任何警告(Warning)訊息都可以一目了然。”Clean”按鈕只是把舊的 pbx 檔及轉譯中的中間檔給清除。”Rebuild”按鈕只是先做”Clean”再做”Build”。而”Detail”按鈕能利用來選擇不同的 GNU 工具和副程式庫來源。”Cancel”按鈕則是用來關閉及離開該視窗的方法。

最後 pbx 檔到底生成在哪裡？答案是與程式碼同一目錄下，在生成的過程中，第一個生成的是一個副檔名為 mak 的 makefile 檔，這是 GNU 編譯器所需要的製作方法檔。Penbex 的 VC 開發環境下的 Add-in 來自動生成，有必要時程式開發者也能將它改名後自行修改。Add-in 會繼續自動依該 makefile，執行 make 的動作，進而產生 pbx 執行檔，過程中還會產生.o 檔，coff 檔，以及 ary 檔等中間檔。如 hello 程式為例，原本目錄中只有 hello.c 以及 hello.h 兩個檔案，如圖 1.8.2。先產生 hello.mak，隨後才生成 hello.o, hello.coff, hello.ary，最後才是 hello.pbx。(結果如圖 1.8.3)

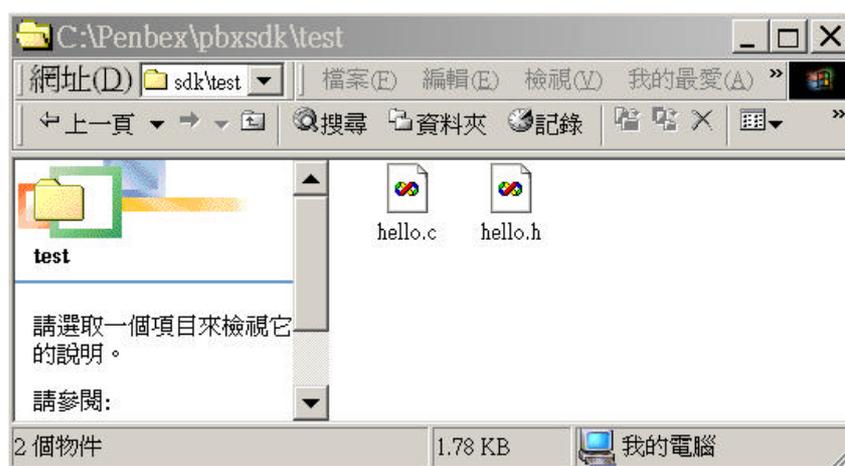


圖 1.8.2 HELLO 程式原本目錄

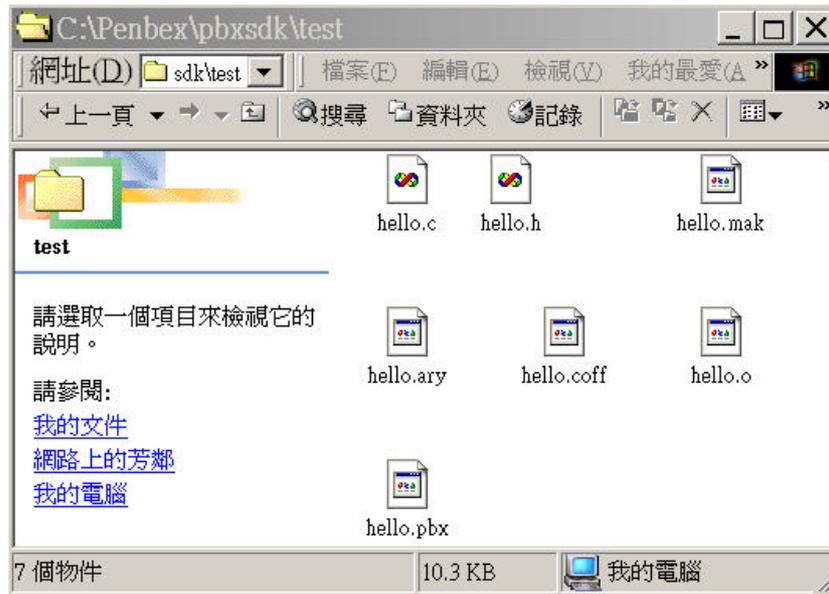


圖 1.8.3 HELLO 轉譯後目錄

同時在 Output 視窗的 Macro 分頁裡會看到以下 make 的過程, 表示成功:

```

===== Clean =====
Deleting intermediate files and output files [hello]
===== Build [hello] =====

C:/PENBEX/Pbxgsdk/penbex/bin/m68k-penbex-coff-gcc -IC:/PENBEX/Pbxgsdk/usr/include -D_GNU_ENV -mPIC
-m68000 -c hello.c
C:/PENBEX/Pbxgsdk/penbex/bin/m68k-penbex-coff-ld -T C:/PENBEX/Pbxgsdk/penbex/m68k-penbex-coff/lib/ldscript.dat
-o hello.coff C:/PENBEX/Pbxgsdk/penbex/m68k-penbex-coff/lib/crt0.o hello.o
C:/PENBEX/Pbxgsdk/penbex/m68k-penbex-coff/lib/libc.a
C:/PENBEX/Pbxgsdk/penbex/lib/gcc-lib/m68k-penbex-coff/2.96/libgcc.a
C:/PENBEX/Pbxgsdk/penbex/bin/m68k-penbex-coff-ld -d -r -T
C:/PENBEX/Pbxgsdk/penbex/m68k-penbex-coff/lib/ldscript.dat -o hello.coff
C:/PENBEX/Pbxgsdk/penbex/m68k-penbex-coff/lib/crt0.o hello.o
C:/PENBEX/Pbxgsdk/penbex/m68k-penbex-coff/lib/libc.a
C:/PENBEX/Pbxgsdk/penbex/lib/gcc-lib/m68k-penbex-coff/2.96/libgcc.a
makepbx hello.coff hello.ary hello.pbx "hello.pbx" "hello" "1.0" "2001/07/07" "PENBEX" "TRADITIONAL" 160 160 "" ""
Penbex (R) PDA Execution File(.PBX) Build Utility, VersionCopyright (C) Penbex Data Systems, Inc.,1999,2000, All rights
reserved.

```

PBX Format Version: v1.1

Warning ! No small ICON.

Warning ! No large ICON.

File Size = 1390(0x0000056e)

Crt0 (Offset,Size) = (0x00000114, 0x00000020)

Text (Offset,Size) = (0x00000134, 0x00000228)

Data (Offset(bss),Size(data,bss))

= (0x0000035c(0x00000364), 0x0000000e(0x00000008,0x00000006))

RelocText (Offset,Size) = (0x0000036a, 0x00000008)

RelocData (Offset,Size) = (0x00000372, 0x00000000)

Res (Offset,Size) = (0x00000372, 0x00000018)

Successed

Batch Build Complete

9. 移植除錯小秘訣

由於 PDA 程式的開發過程, 同一支程式代碼經過了兩種不同 CPU 編譯器的轉譯。不只編譯器不同, 一個是超強的微軟 Visual C Intel x86 編譯器, 另一個是開放的 GNU 68K 編譯器, 連最終執行程式指令集的 CPU 特性也有所不同, 造成程式設計師必須小心規劃。以下幾點特別值得注意:

(1) Intel x86 CPU 採用 Little Endian, 而 Motorola DragonBall 採用 Big Endian 對 word 的資料格式裡每個 byte 的順序是顛倒的, 所以一不小心非常可能造成資料的錯誤, 尤其是資料直接對 I/O 或網路做讀寫時, 需要做 byte swap 或 bit swap 的動作。

(2) 因此也盡量避免使用以 bit 為單位(bit-fields)的資料結構, 可以避免轉譯後在不同 CPU 上執行結果的不同。

(3) DragonBall 有兩個特性要注意, 一、它只能讀寫偶數位址。如果不小心讀寫到奇數位址, 在 Intel x86 CPU 不會有明顯錯誤發生, 但移值到 DragonBall 時馬上發生 Address Error。二、它的資料結構長度固定為四個 byte 的倍數, 也就是所謂的 "structure alignment" 為四個 bytes。在 VC 環境以及 GNU 的編譯器, structure alignment 都已經預設為 4, 不要更改之即可。

(4) Integer 長度在標準 ANSIC 並沒有定義的很死, 在不同編譯器可能定義不同長度的 integer size, 所以使用整數最好仔細定義, 使用 short, long 等固定長度的資料種類 (types)。這類問題不一定會出錯, 只要小心就好。

(5) 也因此盡可能不要用 enum 定義資料結構, 因為怕有 integer type 在其中, 造成跨平台時資料長度結果不同。

(6) Assertion 只用在開發程式中除錯狀態的 VC 環境, 在 GNU 轉譯時目的在產生最終執行檔, 所有除錯狀態會被消除, 因此 Assertion 效果在 GNU 轉譯後也會喪失, 所以不可以把 Assertion 當成一般 if 使用。(Assertion 在第五章之第 12 節有詳細說明)

(7) 有些地方模擬器並沒有模擬到, 例如 CPU 的運算速度, 以及電池使用狀態等。

所以最後還是要做 PDA 實體測試, 才能確保結果是否完全正常。

10. 練習題

二、我的第一支 PDA 程式

1. 前言

如果這是您第一次接觸 PDA 而且是第一次撰寫 PDA 程式,我想這個章節是必讀且重要的一章。因為想寫好一支 PDA 程式,除了了解程式撰寫方法,懂得 C 語言及開發環境的操作和除錯方式以外,了解 PDA 運作原理及 PDA 的特性也非常重要,有助於開發出一個非常好的 PDA 程式。由於它的特性也決定了程式的規格,操作方式,使用者介面,資料架構,備份方式,通訊方式及說明的撰寫等。以下從這幾個方向來討論:

程式規格 – 雖然 PDA 程式的規模相對於在 PC 上開發的程式會比較小,原因是沒有必要設計的太複雜,PDA 本身就是設計成夠用就好,好用才重要。所以在程式設計之初訂定規格就要以此為方針,以下幾個方向都需要個別考慮。但有人也許會說既然程式不大,仔細制定規格就沒有那麼必要,不是嗎?正好相反,從幾個角度來分析:第一、由於 PDA 程式架構簡單,內容就會大同小異,且容易以每個操作畫面來切割訂定流程,由於 Penbex 的程式設計架構與此流程非常容易一對一對應,簡單講畫面跟流程制定好了,程式也差不多完成一半了。第二、相反的,PDA 能力有限,軟體規格與功能必須也要考慮 PDA 硬體及 PDA OS 是否做得到。有整體的程式規劃,就能事先驗證在 PDA 上執行是否可行。最後一個好處是,有了細部規劃使程式更可以分工,更能套用可重複使用之模組,使程式開發更準確、快速、以及順利完成。

操作方式 – 第十節會更仔細的介紹 PDA 軟體的特性,其中一些特性都與操作有關,所以撰寫 PDA 程式就必須為其操作方式做細部規劃。比如說,(1)為了方便使用者快速操作,通常只要以輸入完成或點選後的資料,不需再點選類似“存檔”或“確認”等按鈕。那是典型 PC 程式的做法,但在 PDA 使用上最好少用。能利用一次筆觸完成的事情,就不要用兩個筆觸的設計,以此類推。(2) PDA 使用者常常需要在幾個工具程式間迅速切換,所以程式被呼叫回來時,程式是重新執行,應該自動跳到之前該程式離開時的狀態及位置。比如說正在看某本電子書的某一頁時,突然接到電話,需要使用其他軟體如記事簿而跳回桌面時,電子書瀏覽器突然被系統關掉而離開(程式結束)。當您辦完事想再去開啟電子書瀏覽器時,應該是直接就跳到剛才在看的那本書的那一頁,還是像一般 PC 程式一樣重新開檔,再去尋找上次正在看的地方呢?別忘了我們金玉良言:“一次筆觸能完成的事情,就不要用兩個以上的筆觸來設

計”。(3)不管資料多寡,筆觸後 PDA 反應時間不可以太長,更不能無可預期。PDA 是我們的個人助理,反應要快,是它等我們提出需求,而不是我們花時間等它答案。

使用者介面 – PDA 螢幕很小,最常見的是 160x160 的像素(pixel)。相較於現在一般 PC 的螢幕解析度 1024x780 或 800x600,真是小巫見大巫。又要滿足好用的原則,操作方便,顯示內容又要快又要多,又要夠清楚,真是設計使用者介面的最高挑戰。所以事先規劃好使用者介面,也會影響整個操作模式,甚至於影響整個程式的規格和流程。所以 PDA 程式設計者不能不事先就重視其未來程式的使用者介面,也就是每個視窗畫面操控物件的擺放位置及資料顯示的方法。設計的越簡潔有力越好,不能太複雜,又看不清楚或不易點選等。

資料架構 – 比如說 PDA 最常見的儲存記憶體只有 8MB,對於電子地圖的應用程式為例,光光台灣地圖壓縮後還需要 16MB,不適合全部放在記憶體。一者,選擇只能在更大記憶體容量的 PDA 才能執行,或放在類似 CF 卡(Compact Flash)等的外接儲存器上,也只能給有 CF 卡或其他擴充介面的 PDA 使用,通常價位都比沒有擴充介面的 PDA 貴。所以它資料架構最好能從新思考,以部分(Paging)的方式儲存在伺服器上,加上 PDA 有無線上網及移動的特性,若由 PDA 能透過 GPS (衛星定位系統)取得所在位置,透過無線上網從地圖服務伺服器上及時下載附近的地圖,甚至可以提供附近旅遊資訊不是更有意義及效率嗎?PC 由於硬碟為主要儲存空間,全世界的地圖和旅遊資料都放在幾十 GB 的硬碟中,也許還放不完。可見在 PDA 上設計程式,資料切割及存取方式都與 PC 截然不同。另一個典型的例子是 PDA 電子字典程式,不再像 PC 上的電子字典動輒幾十萬字,非常不適合全部一起存放在 PDA 上以 RAM 或 ROM 的方式儲存。PDA 是隨身攜帶,但又不像純電子字典一樣功能是固定的,也就是說常用字一定要有,因為隨身要用,因為 PDA 就像一台隨身電腦一樣,可以下載多樣化程式或字典資料,比如同一個字典軟體可依用戶需求更新或增加詞彙,例如要考 TOFEL 的學生可以增加 TOFEL 常考單字;商人可以下載商用英語等。所以同樣是字典軟體,其資料架構上的規劃在 PC 與 PDA 程式上也完全不同。

備份方式 – PDA 記載的都是隨身的重要資料,不管是原始參考資料,或是使用者輸入的個人資料,都必須妥善保存且自動備份。原因是不能假設 PDA 不會沒電資料永不流失,更不能假設使用者會主動且常常做備份。所以程式設計最好能利用系統的備份機制,自動為用戶備份資料到 PC 端或像 CF 卡記憶體等周邊儲存器,讓用戶毫無後顧之憂。

通訊方式 – 大部分的 PDA 硬體都至少有串列埠(RS-232)以及紅外線(IrDA)兩種標準對外溝通介面,也就是程式可以利用串列埠及紅外線透過各種方式對外傳輸

資料或與其他裝置溝通。加上 Penbex 等作業系統所提供的通訊協定，程式可以很容易撰寫網路程式，或紅外程式。PDA 程式就應該盡量考慮利用無線傳輸，行動裝置會顯得更活用有趣。例如，(1) PDA 和手機就是天生的一對，又可以傳短訊，又可以上網收發電子郵件，又能替手機鈴聲編曲，也有可能製作手機開機畫面，全是靠 PDA 與手機之間的紅外線來溝通。(2) PDA 之間也互相交換電子名片，也能互傳程式、遊戲、電子書、影像、甚至於 MP3 檔，當然要有附帶 MP3 功能的 PDA 才能直接播放。

說明撰寫 – 理論上 PDA 程式最好能設計的非常容易使用，甚至利用直覺就會操作最理想。就像手機和錄放影機一樣容易上手，而說明書只是非不得已才需要查閱，PDA 程式中的電子說明書也是一樣，只是以備不時之需，簡單扼要就好，也只在不容易理解的地方直接加以補充而已。另一個說明要越少越好的原因是，不管是燒寫在 ROM 裡的內建程式，或是下載在 RAM 的儲存區，都應該越小越好。因為 PDA 的記憶體容量實在是小的可憐，能省則省，需大家共同來省吃簡用。

所以從我的第一支 PDA 程式開始，若能依照這七大原則去著手設計，必能開發出一流水準的作品，殺手級的應用，所謂的 killer application。

2. 又是拿 Hello World 程式為例

每次學習一種新語言，都常常用 Hello World 這個程式範例來做程式架構的解說，我也不例外。因為它是一個能執行的最小程式，也可藉以了解程式的運作流程。



圖 2.1.1 HELLO WORLD 程式畫面

參考 `hello.c` 的原始碼如下：

```
#include "pbxall.h"
#include "hello.h"
GCONST HelloVar cHelloVar={0};
HelloVar *pHelloVar;

static BOOL conProcess (SysMsg *pMsg)
{
    register BOOL retValue=0;
    UIOBJ *pLabel;
    switch(pMsg->type )
    {
        case MT_SYS_CONT_BEGIN:
            ContSetTitle(" Hello World Demo");
            GmTextOut(60, 50 ,"Hello World !!!");
            pLabel = labNew( 40, 50, 100, 60, 19, "OK" , LABSTYLE_STRING, \
                LABSTRPOS_MIDDLE,GM_FONT_MIDDLE , LABFRAME_NO );
```

第六節會詳細介紹
header 檔的用法

全區域變數需要特殊
方式定義，詳見第八節

事件訊息以 MT_開頭表
示，說明詳見第七節

```

ContAddObj( pLabel);
ContRedraw( );
retValue=1;
break;case MT_SYS_CONT_END:
retValue=1;
break;
}
return retValue;
}

#ifdef WIN32
void HelloMain ( unsigned long argc,void *argv )
#else
void PenbexMain ( unsigned long argc,void *argv )
#endif
{
    auto SysMsg msg;
    GCONST_ASSIGN( pHelloVar, cHelloVar);
    if ( APP_RUN_NORMAL == argc ) {
        SetContainerEntry( 100, conProcess );
        SetActiveContainer( 100 );
        while( GetMsg( &msg ) ) {
            if( !SysProcess( &msg ))
                AppProcess( &msg );
        }
    }
    NOUSE(argv);
}

```

粗字體代表各種 Penbex APIs 函式

在 VC 環境下使用任意名稱取代 C 語言的 Main()

GNU 環境裡每個程式都是用 PenbexMain 代替 Main()

這個地方要定義所有 Containers 的入口, 且定義程式開始的第一個 Container 是誰?

事件迴路就是這三行程式, 細節詳見第五節

Penbex 的程式架構也是採用 Event driven 的方式, 也就是事件導向的方式建構程式, 所以在程式中必需在主程式裡定義事件迴路(Event Loop), 而在副程式中定義相對每個視窗所要處理的事件, 事件訊息都以 MT_開頭定義(詳見第七節)。由於在 VC 的環境下每個 PDA 程式不能用 Main()來做主程式名稱, 這是和一般 C 語言程式最大不同的地方, 所以每個程式必須各取一個不同的名稱, 例如這裡的 HelloMain, 而在 GNU 環境下轉譯後的程式就個別以 pbx 檔方式存在, 主程式名稱都相同以 PenbexMain 取代 C 語言標準的 Main()。所以在 hello.c 範例中我同一支程式以#ifdef WIN32與否來判斷何時用 PenbexMain, 何時用 HelloMain (註: 在 VC 環境下 WIN32

始終是被定義的, 除非被`#undef`)。

在進入事件迴路以前, 最重要的是必須先定義好所有視窗入口, 我姑且把 Container 稱之為 PDA 的視窗。利用 `SetContainerEntry`, 或另一種叫 `TabContainer` 的視窗則利用 `SetTitleTabContEntry` 或 `SetTabContEntry` 來設定視窗的入口, 也就是其代碼(ID)以及相對應的副程式名稱。最後一定要指名第一個出現的視窗, 利用函式 `SetActiveContainer` 將其 ID 告訴系統, 當系統進入該程式的事件迴圈時, 才知道從哪一個視窗(Container)開始。而每一個視窗相對應的副程式會傳入事件訊息的結構, 以供程式判斷及處理。最後它也會傳回一個 `return` 值, 1 表示它處理過了, 0 則表示尚未處理, 或希望系統繼續對該事件訊息做反應。(事件迴路原理請詳見本章第五節) 而 `hello.h` 的原始碼如下:

```
#ifndef helloh/[
#define helloth
#ifdef __cplusplus/[
extern "C" {
#endif/[__cplusplus.
typedef struct tagHelloVar {
    short dummy;
} HelloVar;
extern GCONST HolleVar cHelloVar;
extern HelloVar *pHelloVar;
void HelloMain(unsigned long argc,void *argv);
#ifdef __cplusplus/[
}
#endif
#endif
```

只為了避免重複定義

定義一個 HelloVar 資料結構, 內含所有全區域變數, 以便讓 pHelloVar 指到 RAM 記憶體中 cHelloVar 分配到的位置

定義該程式的主程式名稱

3. 程式基本結構

延續第二節對 Hello Word 程式範例的介紹, 我對程式結構再做更深入的說明。如何將.c跟.h程式加到模擬器中? 每當開始要編譯一個新程式時, 就必須在 VC環境下的 Workspace 裡 PDA AP 的目錄上按滑鼠右鍵選擇增加一個新目錄, 比如範例中取名為 hello (圖 2.3.1), 再按於 hello 上按滑鼠右鍵選擇加入檔案 hello.c 及 hello.h

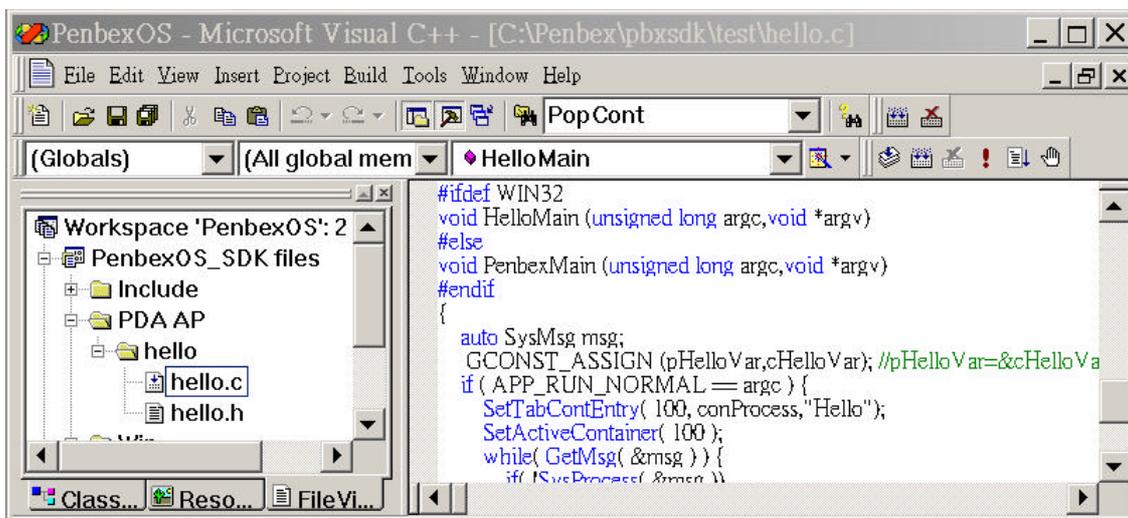


圖 2.3.1 加入新程式 HELLO.C

其次, SDK SYSTEM->Desktop 目錄下的 expreap.c 的內容必須加入:

```
#include "..\test\hello.h"
AppAdd(XX, YY, "Hello.px", "Hello", HelloMain, 0, (void *) \
      &cHelloVar, sizeof(HelloVar), (void **)&pHelloVar);
```

程式本身相對應的 header 檔雖然沒有特別限制, 但如同 Hello World 範例裡的 Hello.h, 定義一個資料結構給全區域變數及定義該程式的主程式名稱還是必要的。因為該 header 檔為了再 expreap.c 檔裡 AppAdd() 會叫用到該程式的主程式名稱及其全區域變數資料結構與指標名稱等, 所以程式的 header 檔須加在 expreap.c 裡。同時利用 AppAdd 函式把 HelloMain 的主程式名告訴模擬器, 並把 HelloVar 等資料結構及全區域變數(cHelloVar)及指標(pHelloVar)告訴它, 如此一來, 模擬器才會正確的呼叫到該 PDA 程式。XX, YY 為大小圖標的定義所在位置, 而"Hello.px"則是模擬下載程式的實際檔名, 第四個參數則是程式的顯示在圖標下的名稱等。

4. 什麼是 Container?

在 PDA 程式裡每一個帶有物件而且會對事件做反應處理的視窗，我們稱之為 Conatiner。顧名思義 Container 是個容器，可以加物件在裡面，所以所有物件(下一章會仔細介紹所有 Penbex 系統提供的繪圖使用者介面 GUI (Graphical User Interface) 物件)。以下為 Container 的幾項定義：

(1) Container 像 PC 程式的視窗，但不能放大縮小，也沒有一般多視窗的觀念，所以無須處理視窗重疊(overlapping)的問題。而且一般 Container 是全螢幕，也就是 160x160 個像素(pixels)的使用範圍。

(2) 每個 Container 可以包含 64 個以內個物件，而且當它正在執行時它會自動照順序處理事件排(Event Queue)裡的事件。

(3) 每個程式至少要有一個 Container，而且要定義哪一個是程式第一個開始執行的 Active Container。注意，每支程式最多只能定義 64 個 Container。

(4) 另外有兩種特別的 Containers，稱為 TabContainer 以及 PopContainer。其中 TabContainer 與一般 Container 的行為和程式設計方法都一樣，只有定義 Container 入口時有所不同，如果我們將 hello.c 裡的 SetContainerEntry(100,conProcess)改成 SetTabContEntry(100,conProcess,"Hello")結果會變成如圖 2.4.1 所示。當我們又把 SetTabContEntry 改成 SetTitleTabContEntry 結果又是如何? (詳見圖 2.4.2)很明顯 TabContainer 只是將 Tab 的 Title 名稱先行定義，因為同時可以定義很多 TabContainer 在同一個畫面裡，使用效果蠻像多視窗之間的切換。(請詳見第四章第四節)



圖 2.4.1 SETTABCONTENTRY 結果



圖 2.4.2 SETTITLETABCONTENTRY 結果

(5) 另一種很特別的 Container 就是 PopContainer, 程式寫法, 使用的所有 APIs, 還有其運作方式都和一般 Container 或 TabContainer 有所不同(請參考第四章第五節)。也只有特殊情況下才會使用 PopContainer, 但使用 PopContainer 效率比較好。其實一般 Container 之間的切換完全把之前的 Container 關閉, 浪費很多 Content switch 的時間, 每個 Container 所需的資源也很多。PopContainer 並不把前一個 Container 或 PopContainer 關閉, 而是直接疊在該畫面之上, 直到自己以 PopContainerEnd 函式來關閉自己為止, 這和一般 Container 被關閉而收到 MT_SYS_CONT_END 的方式截然不同。系統內建程式使用很多共用對話框(Common Dialog Box), 都是利用 PopContainer 來完成的。而且 PopContainer 也可以不是全螢幕, 又是一個和一般 Container 不同的地方。

5. 事件迴路

從以下程式片段我們可以很明顯的看出整個程式,基本上當正常被執行時會先定一個 Container 入口並設定第一個開啟的視窗(SetActiveContainer)之後,進入一個 while loop, 程式利用 GetMessage 訊息函式得到當時事件訊息疊裡的訊息,先經過 SysProcess()系統處理後,再交由使用者的程式處理,也就是進入該程式正在畫面上的視窗(Container)相對應的副程式處理,如果副程式回傳 0 系統會再做必要的處理,回傳 1 則否。(圖 2.6.1)

```
void HelpMain( unsigned long argc,void *argv)
{   auto SysMsg msg;
    if(APP_RUN_NORMAL==argc) {
        SetContainerEntry(100, conProcess);
        SetActiveContainer(100);
        while(GetMsg(&msg)) {
            if(!SysProcess(&msg))
                AppProcess(&msg);
        }
    }
}
```

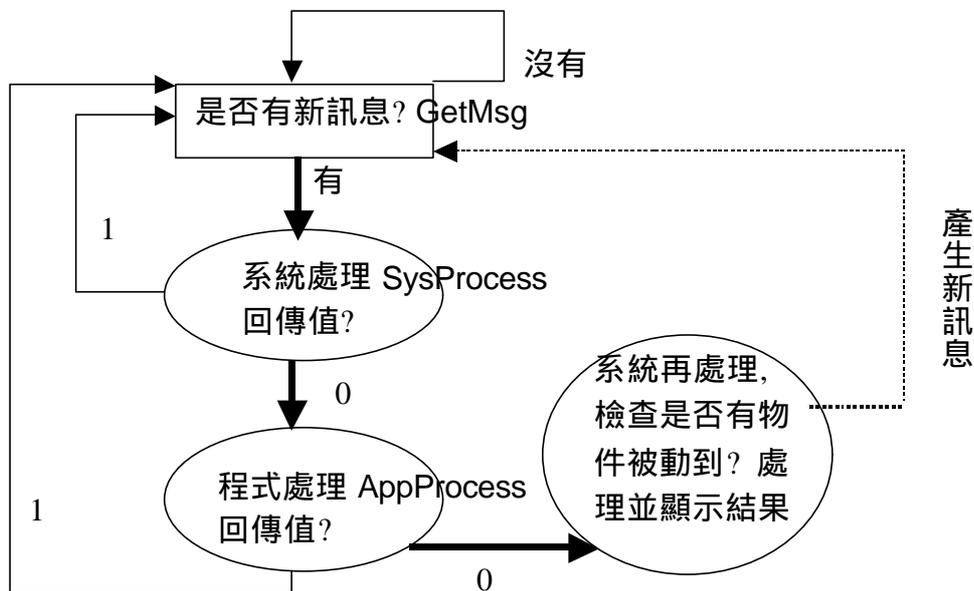


圖 2.6.1 事件迴路示意圖

6. 有哪些 header 檔能用？

在 Penbex 開發環境裡, 只要叫用”pbxall.h”以及程式自己的 header 檔即可。而 pbxall.h 在 VC 環境裡包含了”ansilib.h”, ”pbxos.h”, “UI.h”, 以及“NetAPI.h”檔。在 GNU 環境則自動叫用類似的另一套 header 檔, 其中包含”gansilib.h”, ”gpbxos.h”, “gUI.h”, 以及“gNetAPI.h”檔。

ansilib.h	/* 定義所有跟 Penbex ANSI C 函式有關的東西	*/
pbxos.h	/* 定義了網路 APIs 以外的所有 APIs	*/
UI.h	/* 定義了所有使用者介面物件的 APIs	*/
NetAPI.h	/* 則是定義網路相關 APIs	*/

這些 header 檔會再叫用以下兩個 header 檔;

Typedef.h	/* 包含所有 Penbex SDK 用到的資料種類定義	*/
msg.h	/* 定義所有 MT_* 開頭的事件訊息名稱	*/

7. 有哪些事件訊息種類？

事件訊息大致上分為以下九大類；在本書相關章節會個別介紹：

- 一、系統訊息-又稱 kernel messages, 其中包括: MT_SYS_CONT_BEGIN, MT_SYS_CONT_END 等。還有比較少用的 MT_SYS_POP_CONT_BEGIN, MT_SYS_CONT_UPDATE, MT_SYS_CONT_BEHIND。
- 二、筆觸訊息-英文叫 pen message, 包括: MT_PEN_UP, MT_PEN_DOWN, MT_PEN_MOVE, MT_PEN_REPEAT, 還有比較少用的 MT_PEN_DBCLICK, MT_PEN_CALIBRATE_OK。
- 三、按鍵訊息-MT_KEY_UP, MT_KEY_DOWN, MT_KEY_REPEAT。
- 四、銘版訊息-MT_HOME, MT_MENU, MT_KEYBOARD, MT_BACK, MT_HELP, MT_SEARCH 六種, Penbex 系統規定銘版至少有這六個軟按鈕, 可以利用筆來點選。
- 五、物件訊息-跟 14 種物件有關的訊息。(下一章會詳細對每個 GUI 物件個別介紹)
- 六、UART 訊息-其中又分跟 UrtRcvModeBegin()有關的訊息, 跟 XMODEM 收的 UrtXmodemGet()有關的訊息, XMODEM 送的 UrtXmodemSend()有關的訊息。(UART 程式設計原理, 請詳見下冊第八章)
- 七、紅外線訊息-其中又分跟 Obex 收送有關的訊息, 以及紅外印表相關的 IrLptSend() 有關的訊息。(紅外線程式設計原理, 請詳見下冊第十一章)
- 八、網路訊息-跟網路有關的訊息。(下冊第九章再詳細介紹)
- 九、其他訊息等, 包含 Timer 相關的 MT_TIMER, MT_SLEEP, MT_WAKEUP, MT_BATTERY_LOW, 跟馬錶有關的 MT_RTC_STPWATCH, 跟字串剪貼有關的 MT_CUT, MT_COPY, MT_PASTE, 以及其他如 MT_CHAR, MT_HOT_KEY 等。

8. Penbex 對全區域變數的規定及特性

本章節要重複在介紹一下為何需要用特殊方法定義 Penbex PDA 程式的全區域變數，和其原理與特性。

一般人寫 C 語言程式，對全區域變數(Global Variables)也許會以一般變數的定義方式定義在副程式外，程式開頭附近的地方，而且常常沒有設定其初值（都假設編譯器會預設初值為 0x00）。這個方法沒錯，至少在微軟的 VC 開發環境裡在 PC 上執行該程式時沒問題（VC 編譯器會對未給值的全區域變數設定初值）。但不表示所有的編譯器都如此，我們 PDA 的執行檔最終將經由 GNU 編譯器轉編為 68k 的 pbx 執行檔，其所受的待遇就未必和 VC 相同，更何況該程式執行碼可能會在不同的記憶區被執行，在 PDA 上就有好幾種截然不同的方法，以下會一一介紹。

一般方法所定義全區域變數的位址，會隨著執行檔預留在資料區(data section)，相對於程式的執行碼區(code section)一樣在執行檔裡已預留。一般程式在執行時都會被下載到 RAM 中執行(內建軟體除外)，對下載到 RAM disk 的 PDA 程式而言，也是如此，資料區的地方，資料是可以被正常存取的。但 Penbex PDA 程式還有可能被燒錄在 Flash ROM 或 ROM mask 裡當成內建應用程式，如果該程式被硬體廠商整合到 PDA 裡或在垂直市場應用必須保存程式在 ROM 裡時，該程式的全區域變數定義方式就會面臨考驗。

原因是 PDA 在 ROM 裡的程式，為了節省 RAM 的空間，程式部分包括執行碼區和資料區，都直接在 ROM 裡執行。所以唯讀全區域變數還好，不受影響，但存取全區域變數就不得利用前幾節所介紹的方式來定義，因為只要把資料定義成一個 constant 的結構體，再用 pointer 指標方式來使用，不管任何編譯器和執行環境，結果都是相同的，系統都會在執行時，在 RAM 中另闢一塊記憶體給這些在結構體裡的變數。也就是不論該程式是在 ROM 或 RAM 裡執行，全區域變數都可以被正常存取。

9. 如何在 LCD 螢幕上顯示字串？

不管是想要顯示一段字串，或顯示一個值在螢幕上，利用 ANSI C 標準的 `Sprintf` 函式將所想要的顯示的東西包裝成一組字串，再利用 `GmTextOut` 將它畫在 LCD 螢幕上(x,y)的位置即可。

```
char    cBufA[32];
BYTE    fontID=GM_FONT_SMALL;
int     x=100;
int     y=50;
    GmSetFont(fontID);
    Sprintf(cBufA,"x=%03d,y=%03d",x,y);
    GmTextOut(50,60, cBufA);
```

說明以上程式片段，先定義一個長度為 32 位元的字串變數 `cBufA`，如果需要改變字型，還可定義一個設字型變數 `fontID`，預設該值為小字型 `GM_FONT_SMALL`。程式將“x=100, y=050”的字串顯示在 LCD 螢幕(50,60)的位置。(如圖 2.9.1)

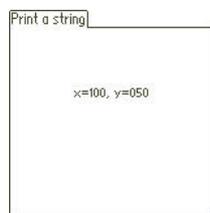


圖 2.9.1 顯示字串

`Sprintf` 第二個參數使用 ANSI C 標準的格式化轉換宣告方法，有 `d, i` 用來宣告整數，有 `o, u, x, X` 用來宣告 `unsigned int`，有 `f, e, E, g, G` 用來宣告 `double` 數目，用 `c` 來宣告 `unsigned char`，以及用 `s` 來宣告字串等。由於 Penbex 都採用標準 C 的定義，所以可參考一般 C 語言的書介紹 `sprintf` 或 `fprintf` 參數的宣告方法。

10. 練習題

三、Penbex 提供哪些 GUI 物件？

1. 前言

Penbex SDK 提供十四種 PDA 最常用的繪圖使用者介面(GUI)，以提高軟體開發者的生產力，就像 Visual Basic，或 Visual C 所提供的 MFC，讓軟體開發者能藉以開發出方便用戶操作的畫面，而不需要所有操作介面都要有軟體開發者一點一畫的做出來。

目前 Penbex SDK 利用物件的方式加上其對應的函式(APIs)，來提供軟體開發者做出程式的 GUI。其中包括按鈕(button)，列表(list)，文字輸出物件(editor)，表格(table)，對話框(message box)，樹狀物件(tree)，白板(white board)，資料庫欄位(field)，還有進度條(progress)，捲動條(scroll bar)，工具條(toolbar)等。本章以下幾節會陸續詳細介紹，並提供程式片段以做說明。參考 SDK 裡的程式範例是最容易了解物件以及學習撰寫程式 GUI 的方法。

2. 按鈕物件

在以下圖 3.2.1 可以了解所有按鈕物件的種類,(a)介紹最常用的一般按鍵,左排五個按鍵使用不同的字型大小,以及文字對齊的方法,和按鍵的內容(字串或圖案)。在 UI.h 裡可以看到 btnNewEx()函式的定義如下:

```
UIOBJ *btnNewEx(WORD id,int x,int y,int w,int h,const void *pData,WORD Style,\
                WORD StrPosition,BYTE FontID,WORD Kind)
```

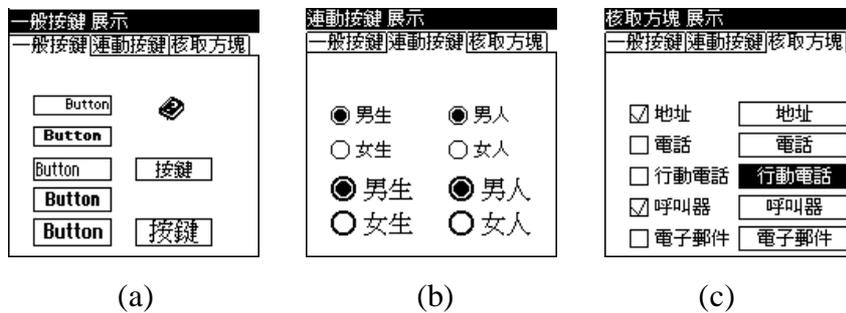


圖 3.2.1 按鈕物件

其中(FontID 變數)字型大小有分以下五種: (但中文字只有中字體與大字體兩種)

GM_FONT_SMALL	小字體
GM_FONT_SMALLBOLD	小粗字
GM_FONT_MIDDLE	中字體
GM_FONT_MIDDLEBOLD	中粗字
GM_FONT_LARGE	大字體

(StrPosition 變數)文字對齊的方法有以下三種:

BTNSTRPOS_RIGHT	靠右
BTNSTRPOS_LEFT	靠左
BTNSTRPOS_MIDDLE	置中

(Style 變數)定義按鍵內容是文字還是影像:

BTNSTYLE_STRING	文字
BTNSTYLE_BMP	圖像(如圖 3.2.1 (a)右排上)

最後一個變數(Kind)決定按鍵的外觀:

BTN_KIND_BUTTON	//一般按鍵(如圖 3.2.1 (a))
BTN_KIND_CHECKBOX_NORMAL	//Checkbox 一(如圖 3.2.1 (c)右排)
BTN_KIND_CHECKBOX_CHECK	//Checkbox 二(如圖 3.2.1 (c)左排)
BTN_KIND_CHECKBOX_RADIO	//Radio button (如圖 3.2.1 (b))
BTN_KIND_CHECKBOX_CANCEL	//Checkbox 二, 被選打叉
BTN_KIND_CHECKBOX_CORRECT	//Checkbox 二, 被選打勾

另一個更簡單產生新 button 的函式叫 btnNew(), 只有最定義是字串方式還是圖像方式, 用法如下: (至於如何產生 BmpBook[] 圖像資料, 請參考第六章第五節)

```
GCONST WORD BmpBook[]={
0x626D,0x0400,0x0010,0x0010,0x0001,0xF000,0x000C,0x7F00,0x0011,0xDDF0,// 10
0x00CD,0x037C,0x011D,0x70DF,0x0C77,0x50DD,0x31DD,0x0371,0x5DDD,0x77C1,// 20
0x7F41,0xDF07,0x53DD,0xDC1C,0x5C3D,0xF070,0x3DC3,0xC1C0,0x03F4,0x0700,// 30
0x003F,0x5C00,0x0013,0xF000,0x0000,0x0000,// 36
};
pBtn=btnNew(60,95,55,20,20,(void *)&BmpBook,LABSTYLE_BITMAP);
或
pBtn=btnNew(60,95,55,20,20,(void *)"OK",LABSTYLE_STRING);
```

另外有兩個最常用的 button 相關函式, 一個就是檢查該 Checkbox 按鈕是否被勾選, 利用 btnGetCheckBoxState() 的回 1 或 0, 其中 1 表示已被勾選, 0 表示未被勾選。另一個是用來把 Radio buttons 設成同一組(Group), 使用 btnSetRadioBtnGroup (UIOBJ *pBtn, WORD GroupID), 第二個變數是一個組的辨識碼, 第一個變數為要加入該組的新按鍵指標。

與按鈕相關事件訊息有: (最常用為 MT_BUTTON_COMMAND)

MT_BUTTON_COMMAND	//當按鈕物件被按到時
MT_BUTTON_FOCUS_IN	//筆滑入按鈕物件上面時
MT_BUTTON_FOCUS_OUT	//筆滑出按鈕物件時
MT_BUTTON_REPEAT	//當按鈕物件被按著不放時

3. 列表物件

PDA 最常見的快速輸入，不外乎利用列表拉選方式，其中有 List 類及 Combo 類兩種。圖 3.3.1 (a)和(b)展示中英文 List 列表, (c)則利用 Combo 的選單方式，其中 Combo 和 List 最大的不同是 Combo 大部分只留下選擇的項目(item)在螢幕上。

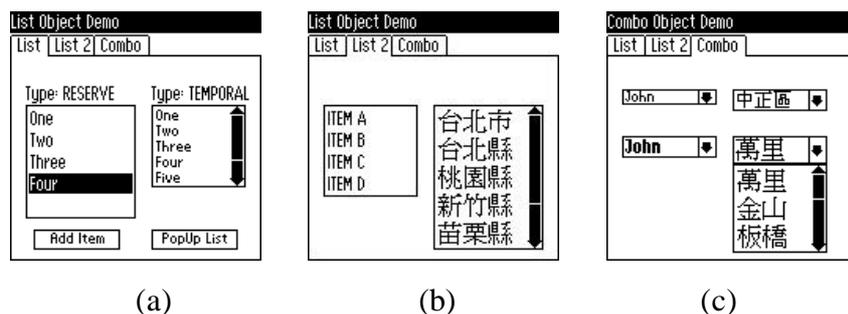


圖 3.3.1 列表物件

listNew 在 UI.h 中定義如下:

UIOBJ *listNew (WORD id, int x, int y, int w, int PageSize, BYTE FontId, BYTE Style);
產生所有新物件都傳回一個 UIOBJ 的資料結構, listNew 也不例外。前四個參數和一般物件一樣, 先給一個新 ID, 再給要放在螢幕 x, y 的位置, 以及寬度 w, 但第五個參數 PageSize 不是高, 而是一次最多能顯示幾個選項, 第四個是字型, 第五個是樣式。

其中樣式(Style)有:

```
LIST_BAR_RESERVE // 會反白  
LIST_BAR_TEMPORAL // 只會暫時反白  
LIST_BAR_ITEM_FRAME // 每個項目都有框  
LIST_BAR_UNDERLINE // 有底線
```

在 ExListCombo.c 範例程式裡, 在 MT_SYS_CONT_BEGIN 的時候利用以下幾行定義了一個”One”, ”Two”, ”Three”, ”Four”的列表物件, 並使用 listSetItemActive() 將第四個選項(item)”Four”反白, 再將物件加入該 Container 中, 並用 uiRedraw()單獨重畫出來。(或最後才一起用 ContRedraw(), 將所有物件一起重畫)

```

UIOBJ *pList;
pList=listNew (10, 10, 60, 60, 5, GM_FONT_MIDDLE, \
                LIST_BAR_RESERVE);

listAddItem (pList, 1, "One");
listAddItem (pList, 2, "Two");
listAddItem (pList, 3, "Three");
listAddItem (pList, 4, "Four");
listSetItemActive(pList,4);
ContAddObj (pList);
uiRedraw(pList);

```

與列表(list)相關的事件訊息有: (最常用為 MT_LIST_COMMAND)

```

MT_LIST_COMMAND
MT_LIST_FOCUS_IN
MT_LIST_FOCUS_OUT
MT_LIST_FOCUS_MOVE

```

還有一種簡易的方式彈出列表就是直接叫用 listPopUpEx()函式, 其定義如下:

```

listPopUpEx (int x, int y, int w, int PageSize, BYTE FontId, int ItemTotal, L_NODE
*Item, int iActiveItem), 其參數與 listNew 和 listAddItemArray 的組合十分相似, 我就
不再重複介紹了。

```

另一種類似的物件叫"選單"(combo), 如圖 3.3.1 (c)所示。其定義如下:

```

UIOBJ *comboNew(WORD id, int x, int y, int w, BYTE FontId, BYTE ItemOfWindow )
參數定義幾乎與 listNew 一樣。而與選單(combo)相關的事件訊息如下, 最常用為
MT_COMBO_COMMAND。

```

```

MT_COMBO_COMMAND           // 當選單被按到時
MT_COMBO_FOCUS_IN         // 當筆滑入某選單時
MT_COMBO_FOCUS_OUT        // 當筆滑出某選單時
MT_COMBO_FOCUS_MOVE       // 選到且滑動選單時

```

4. 文字輸出入物件

文字輸入或輸出物件在 PDA 的應用程式裡也很常用到，由於它可以修改輸出入的內容，我們又稱它為編輯器(editor)。比如單行輸入物件，又稱做“單行編輯器”(single line editor)，如圖 3.4.1 (a)為例，常被用來做資料、密碼或數字等輸入。它也可以同時顯示預設值，或當結果的輸出，但單行意味著只能顯示一行資料，雖然 SDK1.3 版的單行編輯器有再便多行的功能，但畢竟它的用意是拿來做簡單的輸出入用。

另一種編輯器叫“多行編輯器”(multi line editor)，如圖 3.4.1 (b)所示，當多行編輯器輸入超過物件顯示的行數時，捲動條(scroll bar)會自動產生，使用者可以移動捲動條來瀏覽全文。圖 3.4.1 (b)還同時叫出軟鍵盤，便於輸入複雜的內容，其實“編輯器”物件，如果想叫出軟鍵盤，只要筆觸兩下該物件即可，輸入完關閉該視窗即可自動回到原來的畫面，輸入的內容也已顯示在該物件中。

多行編輯器的另一種應用為文章的輸出，下冊第七章 PDA 電子書的程式設計就是利用多行編輯器來顯示電子書的內容。圖 3.4.1 (c)也是多行編輯器應用的範例，當內容超出物件所能顯示的行數時，整個上下翻頁的處理全是物件自己完成。

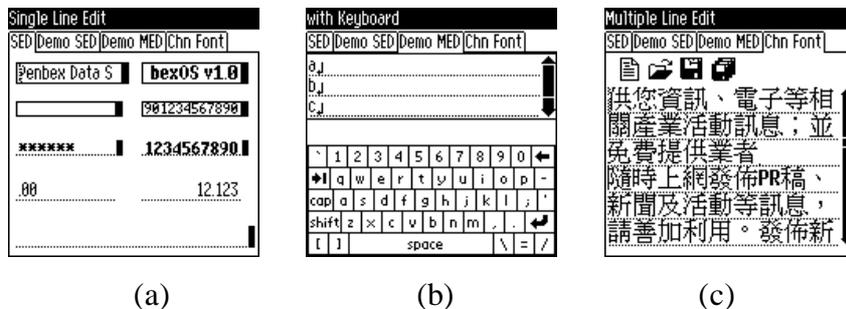


圖 3.4.1 編輯器物件

以程式範例 Exedit.c 為例，產生一個簡單的單行編輯器方法如下：

```
UIOBJ *pEdit1;  
pEdit1 = sedNew(100,5,35,64,ES_STYLE_NORMAL|ES_STYLE_READONLY, \  
                GM_FONT_MIDDLE, ES_UI_DOCK_TOP);  
sedSetEditText(pEdit1, "Penbex Data Systems, Inc. PenbexOS v1.0");  
ContAddObj( pEdit1 );
```

其中也是先設一個 pEdit1 指標指向 UIOBJ 資料結構, 再用 sedNew() 產生新的單行編輯器, 且用 ConAddObj() 函式將它加入該 Container 中, 為了顯示預設值於該單行編輯器, 還可用 sedSetEditText() 函式來將預設值加到該 pEdit1 編輯器裡。sedNew 函式的定義如下:

```
UIOBJ * sedNew(unsigned short id, short x, short y, short cx, \
               WORD Style, BYTE fontID, BYTE DockMode)
```

其中 cx 表示該單行編輯器的寬度(以 pixel 為單位), Style 為該編輯器的模式分以下 9 種: (可以合用, 用 | 將所需模式合併使用)。

```
ES_STYLE_NORMAL           //一般
ES_STYLE_PASSWORD        //密碼方式以(*)顯示
ES_STYLE_NUMERIC         //只有數字
ES_STYLE_ALIGN_RIGHT     //內容靠右
ES_STYLE_NO_BORDER      //沒有邊界
ES_STYLE_NO_EXTERN      //輸入不超出物件寬度
ES_STYLE_READONLY       //唯讀模式
ES_STYLE_NO_POPUP_KB    //不允許跳出軟鍵盤
ES_STYLE_NO_ENTER       //不顯示 enter 的符號
```

最後一個參數 DockMode 則代表捲動條顯示的方式。

```
ES_UL_DOCK_TOP           // 其值為'T', 捲動條在上方
ES_UL_DOCK_RIGHT        // 其值為'R', 捲動條在右方
ES_UL_DOCK_BOTTOM       // 其值為'B', 捲動條在下方
```

那麼如果有人修改該編輯器的內容, 程式如何取得? 在收到事件訊息時可以利用以下方式找回編輯器裡的值, 別忘了利用該編輯器的 ID 喔! 由於物件都採用最佳化記憶體管理, 傳回的記憶體指標需利用 MmLockH 來獲得 (請參考第四章第三節, 對 Handler 的詳細說明), 再利用 Strcmp() 字串拷貝函式將該內容從該記憶體指標複製出來, 最後別忘了使用 MmUnLockH 將該 Handler 結構指標釋放掉。

```
static char StringA[128];
char *pStr;
```

```
VHANDLE ppH;
```

```
ppH = sedGetEditText(ContGetObj(100));          // 100 為該單行編輯器的 ID  
pStr = MmLockH(ppH);  
Strcmp(pStr, StringA);  
MmUnLockH(ppH);
```

還有幾個跟”單行編輯器”有關的函式，比較常用的有 sedSetMaxCnt (pEdit, max_count)用來限制最多能輸入的字母數量 max_count。sedSetFocus (pEdit)則是用來將游標強制設到該編輯物件上。medNew 函式的定義如下：

```
UIOBJ *medNew (unsigned short id, short x, short y, WORD cx,WORD lineNum, BYTE  
Style, BYTE fontID)
```

至於多行編輯器範例程式片段如下：(其中 Book 為某些資料內容，會從多行編輯器顯示)

```
UIOBJ pEdit2;  
pEdit2 = medNew(100,1,50,147,30, EM_STYLE_NO_BORDER| \  
EM_STYLE_NO_SHOWENTER, GM_FONT_MIDDLE);  
medSetEditText (pEdit2, (char *)Book);  
ContAddObj( pEdit2 );
```

5. 表格物件

在某些情況下，有些只用按鈕或用單一 label 是不夠的，或用 list 列表選單也不方便時，還有一種物件叫 table(表格)非常好用。它類似按鈕物件，但作成一維或二維的連續按鈕，被筆點選其中之一時，物件會傳回其座標值。

在顯示方面，也有幾種組合，可以是有框的表格(如圖 3.5.1 (a)及(c))。或是無邊框的表格(如圖 3.5.1 (b))，點選後是否會反白，而且反白是否會留住在該選項，都可以用顯示模式來定義之。

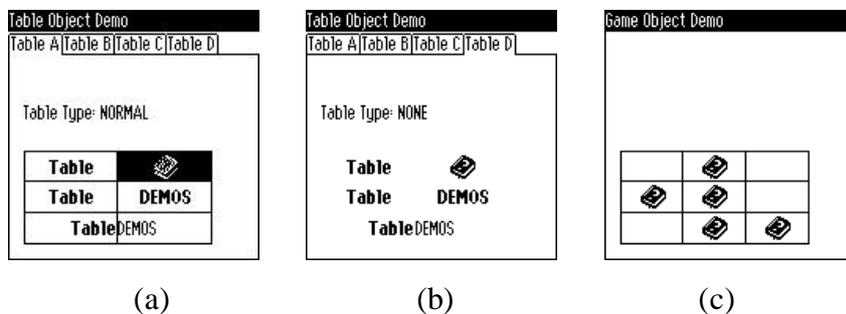


圖 3.5.1 表格物件

程式範例如下: (如圖 3.5.1 (a)結果)

```
pTable=tblNew(10,10,90,120,60,2,3,TS_NORMAL); // 新建立一個 table
tblSetReserve (pTable,1,0); // 設座標 1,0 成反白
tblTextOut (pTable,0,0,"Table",GM_FONT_MIDDLEBOLD);// 給座標 0,0 "Table"
tblBmpOut (pTable,1,0,(LBmp *)&BmpBookB); // 給座標 1,0 一個圖案
tblTextOut (pTable,0,1,"Table", GM_FONT_MIDDLEBOLD);
tblTextOut (pTable,1,1,"DEMOS", GM_FONT_MIDDLEBOLD);
tblTextOutEx
(pTable,0,2,"Table",GM_FONT_MIDDLEBOLD,TS_STRING_RIGHT);
// 字串靠右
tblTextOutEx (pTable,1,2,"DEMOS",GM_FONT_MIDDLE,TS_STRING_LEFT);
// 字串靠左
ContAddObj(pTable); // 加到 Container
```

以下為 BmpBookB 圖案的資料,

```
WORD BmpBookB[]={
0x626D,0x0400,0x0010,0x0010,0x0001,0xF000,0x000C,0x7F00,0x0011,0xDDF0,// 10
0x00CD,0x037C,0x011D,0x70DF,0x0C77,0x50DD,0x31DD,0x0371,0x5DDD,0x77C1,// 20
0x7F41,0xDF07,0x53DD,0xDC1C,0x5C3D,0xF070,0x3DC3,0xC1C0,0x03F4,0x0700,// 30
0x003F,0x5C00,0x0013,0xF000,0x0000,0x0000,// 36
};
```

當表格被點選時, 程式會收到 MT_TABLE_COMMAND 訊息, 從事件迴圈裡可加以下幾行, 從 pMsg->id 能得知該表格的 ID, 並從 pMsg->data.w 及 pMsg->data.pos.y 能獲得該物件被點選到的座標, 也就是 3x3 的表格, 最左上角 item 為 0,0, 最右上角 item 為 0,2 等。

```
case MT_TABLE_COMMAND:
    Sprintf(cBuf, "id=%d,item=%d, %d", pMsg->id, \
            pMsg->data.w, pMsg->data.pos.y);
    GmTextOut(40,40,cBuf);
    retValue=1;
    break;
```

以下為表格的種類, 用於 tblNew()的最後一個參數。

```
TS_RESERVE                // 會保留反白, 其值為 1
TS_FRAME                  // 有邊框, 其值為 2
TS_NORMAL                 // 正常, 其值為 3
TS_FRAME_3D              // 尚未提供, 其值為 4
TS_NORMAL_3D             // 尚未提供, 其值為 5
```

tblTextOutEx()比 tblTextOut()多最後一個參數能決定字串的位置, 顯示在靠左, 靠右, 還是對中等等。其定義如下:

```
TS_STR_HRZ                // 尚未提供
TS_STRING_LEFT           // 靠左對齊
TS_STRING_MIDDLE         // 靠中
TS_STRING_RIGHT          // 靠右對齊
```

所以有以下三種內容的設定方法，有 `tblTextOut()`, `tblTextOutEx()`, 以及 `tblBmpOut()`。其中 `tblTextOut` 與 `tblTextOutEx` 都是用來加字串，而 `tblBmpOut` 是用來加圖案，其定義如下，它們都有共同參數，就是第一個參數是一個表格物件指標，第二和第三個參數是寫入該表格的哪一個欄位，`sCol` 為縱列，`sRow` 為橫排，`fontID` 跟其他物件一樣，決定字串使用的字型。

```
tblTextOut (UIOBJ *pThis,int sCol,int sRow,char *szString,BYTE fontID)
tblTextOutEx (UIOBJ *pThis,int sCol,int sRow,char *szString,BYTE fontID,int
type)
tblBmpOut (UIOBJ *pThis,int sCol,int sRow,LBmp *bmp)
```

其他還有許多表格物件相關函式，不勝枚舉。請參考‘Penbex SDK Reference Manual’

6. 對話框物件

對話框是一種特別的物件，它無須附著在任何 Container 之上，但它本身也有固定功能的按鈕物件，也會處理其按鍵被選後的結果。像圖 3.6.1 (a)就是一種“是”，“否”，以及“取消”三個按鈕的對話框，它可以當成確認對話框，或其他用途。圖 3.6.1 (b)則是只有“是”和“否”兩個按鈕，而圖 3.6.1 (c)則是只有“確定”一個按鈕的對話框。

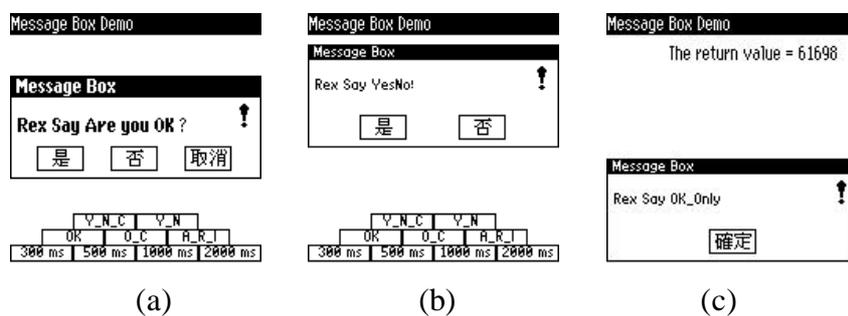


圖 3.6.1 對話框物件

以下程式將啟動一個警告對話框，名稱為“Message Box”，內容顯示“Rex Say Yes No!”，如範例字型用 GM_FONT_SMALL 小字型，以及圖案為警告標誌。

```
retValue=MsgBox(20,"Message Box","Rex Say Yes No!",YesNo, \
MODE_WARNING, GM_FONT_SMALL);
```

除 WARNING 警告標誌以外，還有 MODE_ERROR, MODE_DOUBT 兩種個別顯示錯誤或疑問的對話框符號。

還有一種不需回答的框框稱為 floating box，可以在螢幕上顯示小視窗短暫時間，其程式寫法如下：

```
FloatBox("Test DELAY 300 ms!",DELAY300ms,GM_FONT_SMALL);
```

當該行程式執行時，螢幕上顯示“Test DELAY 300 ms”小畫面 0.3 秒。(如圖 3.6.2) 其中 DELAY300ms 值為 30，單位為 tick，一個 tick10ms。

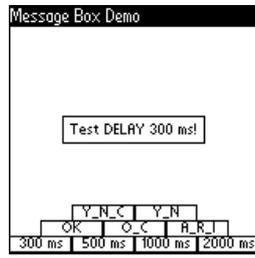


圖 3.6.2 FLOATBOX 的畫面

7. 樹狀物件

樹狀物件在 PC 的視窗環境裡很常見到, 如同 window 的檔案總管, 就是利用樹狀物件來瀏覽目錄和檔案。所以要有系統的以樹狀分類作成選項在螢幕有限的 PDA 上是非常好用, 如圖 3.7.1 所示, 樹狀物件可以以樹狀分類方式, 當內容展開超過顯示範圍的大小時, 捲動條會自動產生, 所有點選和利用捲動條的移動, 都是由物件自動處理。只要先宣告好物件內容, 其他的事就交由樹狀物件處理了, 直到用戶點選到某個項目, 程式會收到 MT_TREE_COMMAND 事件訊息。

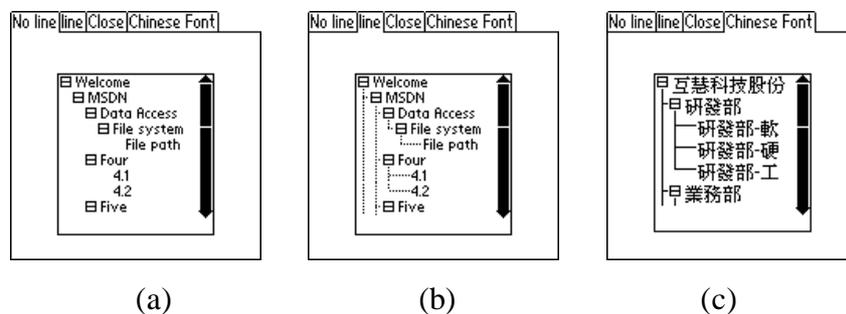


圖 3.7.1 樹狀物件

圖 3.7.1 裡(a)使用的無線條方法顯示,(b)則以虛線連結其內容,(c)除了用實線表示以外還以中文為樹狀物件之內容。以下是圖(c)的程式片段:

第一, 先定義好一個陣列, 代表這樹狀物件的內容如下: (每個元素又是一個陣列, 該陣列第一個元素為數目表示在樹狀的第幾層, 第二個元素為其內容字串, 別忘了最後要以 {0,} 當成結尾!)

```
TREENODE TreeChinese[]={
    {0, "互慧科技股份有限公司"},
    {1, "研發部"},
    {2, "研發部-軟體"},
    {2, "研發部-硬體"},
    {2, "研發部-工業設計"},
    {1, "業務部"},
    {2, "業務部-應用"},
    {2, "業務部-通路"},
```

```

    {2, "業務部-公關"},
    {2, "業務部-亞太區"},
    {1, "生產部"},
    {2, "工廠"},
    {2, "品保"},
    {2, "測試"},
    {0, },
};

```

第二，把它加到 Container 裡，畫出來即可：

```
UIOBJ pTree;
```

```

pTree = treeNew (400, 30, 40, 100, 7, \
                OPEN_ALL|LINK_LINE|TREE_SOLID_LINE, \
                GM_FONT_MIDDLE, (TREENODE *)TreeChinese);
ContAddObj (pTree);
ContRedraw();

```

它是一個 ID 為 400，寬 100，高 7 行畫在螢幕座標 30, 40 的地方，使用實線 TREE_SOLID_LINE。一開始就全展開，所以用 OPEN_ALL 及 LINK_LINE 連起來。使用 GM_FONT_MIDDLE 中字型，內容給先前定義 TREENODE 結構體的 TreeChinese。

在收到樹狀物件事件時，顯示被點選的項目，程式範例如下：

```

case MT_TREE_COMMAND:
    GmTextOut (10,30, "                ");
    GmTextOut (10,30, (char *)pMsg->data.p);
    break;

```

事件訊息裡 pMsg->data.p 包含的就是被點選到項目的內容。將它在座標 10,30 的地方顯示出來。例如：(筆點到研發部時，顯示內容如下)

互慧科技股份有限公司\研發部

8. 白板物件

除了一般繪圖函式能用來在 PDA 螢幕上畫圖以外，配合筆在螢幕上滑動其實很容易做出類似 PC 上小畫家的軟體，但是 Penbex 為了提供程式開發者更快開發出類似畫板的功能，提供一種稱為“白板”(whiteboard)的物件。類似電子簽名，速記本，甚至所有需要繪圖的軟體，都能使用“白板”物件來開發。如圖 3.8.1 就是一個白板物件的展示軟體，有兩個畫板，還可改變筆的粗細，圖案，畫法與橡皮擦等等。

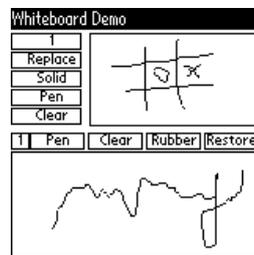


圖 3.8.1 白板物件

```
UIOBJ *wbNew(WORD id,int x,int y,int w,int h) // 在 x,y 做出一個寬 w, 高 h 的白板  
void wbClean(UIOBJ *pThis) // 清除白板內容  
void wbSetPenWidth(UIOBJ *pThis,BYTE PenWidth) // 設白板筆的寬度
```

筆的寬度定義如下：

```
WB_PW_LEVEL1 // 筆的寬度，其值為 1  
WB_PW_LEVEL2 // 筆的寬度，其值為 2  
WB_PW_LEVEL3 // 筆的寬度，其值為 3  
WB_PW_LEVEL4 // 筆的寬度，其值為 4  
WB_PW_LEVEL5 // 筆的寬度，其值為 5  
WB_PW_LEVEL6 // 筆的寬度，其值為 6
```

wbSetPenColor(UIOBJ *pThis,BYTE PenClr)函式用來設定筆的顏色，目前只有四色灰階，所以可定義成以下 4 種顏色：

```
PENCLR_LEVEL0           // 白色
PENCLR_LEVEL1           // 淺灰色
PENCLR_LEVEL2           // 深灰色
PENCLR_LEVEL3           // 黑色
```

wbSetPenMode(UIOBJ *pThis,BYTE PenMode)函式用來設定筆畫顏色和現有顏色的混合方式, 有分成以下 3 種:

```
WB_PENMODE_REPLACE     // 取代
WB_PENMODE_XOR         // 用 XOR 的方式
WB_PENMODE_TRANSPARENT // 透明
```

wbSetPenStyle(UIOBJ *pThis,BYTE PenStyle)函式則用來設定筆的種類, 有分以下 3 種線條表示方法:

```
WB_PEN_PATTERN_SOLID   // 實線
WB_PEN_PATTERN_DOT     // 點線
WB_PEN_PATTERN_DASH    // 虛線
```

wbSetDrawStyle(UIOBJ *pThis,int DrawStyle)函式用來改變該白板的畫法 (DrawStyle), 其中畫法模式有以下 5 種:

```
WB_DrawStyle_NORMAL    // 一般正常模式
WB_DrawStyle_LINEW     // 畫線模式
WB_DrawStyle_RECT      // 畫方形模式
WB_DrawStyle_CIRCULAR  // 畫圈圈模式
WB_DrawStyle_GETBMP    // 設定成剪貼的模式
```

wbCopyBmp(UIOBJ *pThis)函式將整個白板的內容切入剪貼區, (參考第四章第十三節, 有文字 Cut 與 Paste 的介紹)。

wbPasteBmp(UIOBJ *pThis,WORD x,WORD y)函式則是將剪貼區的影像再貼回 pThis 白板座標 x, y 的位置。

9. 資料庫欄位物件

PDA 上資料庫的應用，還是以顯示為主。查詢或從伺服器下載資料，最終還是需要以欄位的方式顯示在螢幕上。在 PDA 上通常螢幕都很小，所以 Penbex OS 提供一個非常精緻的物件叫 Field(欄位物件)。如圖 3.9.1 所見，含意與號碼是兩個資料庫的欄位，以下所列為其資料紀錄(records)，如果紀錄太多而超過螢幕或該物件範圍所能顯示，捲動條就自動產生，這是它第一個功能。

其次它可以自動對每個欄位資料做排序(sort)，當使用者點選每個欄位的表頭(title)時，所有資料自動依該欄位做排序，很類似 PC 上檔案瀏覽器，選擇檢視詳細資料時，點選“名稱”、“大小”、“類型”或“修改日期”，其目錄所有檔案就會以該方式做排序。所以要在 Penbex PDA 上做這個類似功能等於是輕而易舉。

它還有第三個功能是欄位顯示的寬度也可以隨用戶需要隨時放大縮小，以便於在這麼小的螢幕下能快速瀏覽某些過長的資料，以圖 3.9.1 倒數第四行的資料為例，只有該筆資料含意太長，只要拉動每個欄位之間的邊線，可以自由改變大小。這其實跟 PC 的很多欄位物件也很相似。在 Penbex OS 環境下，只要利用資料庫欄位物件撰寫程式，所有三個功能就自動完成，這也就是為什麼 Penbex 程式的執行檔跟別人比起來最小，程式比別人好寫的原因之一吧。



含意	號碼
你是我唯一	04551
你是我最愛	04592
你就是愛	0942
一生一世就愛你	1314920
愛你	20
想你的親想你的掛	30273028
深深愛上你	33230
想死你了	3406
時時想你	4430

圖 3.9.1 資料庫欄位物件

以下介紹如何定義及使用一個資料庫欄位物件，程式一定要先準備好一個 dbaseIII 的資料檔，再把資料檔名稱對應給該物件，通知物件顯示哪幾個欄位即可。

從*fldNew(WORD id,FldDes *pFldDes,WORD x,WORD y,WORD w,WORD h)的函式定義中，可以看出來，如果要產生一個新 Field 物件以前，必須先準備好它要的值，也就是一個 FldDes 資料結構體的位置，而且要把其內容填滿。FldDes 結構有

點複雜, 詳細說明如下:

先定義一個相對應的資料庫說明變數 `gtDbDes`, 其中第一個值是資料庫檔名, 第二個值為要顯示幾個欄位, 第三個值為一個記載每個相對欄位顯示寬度的陣列, 以 Pixel 為單位, 第四個值為一個記載相對欄位名稱的陣列, 這些值是在 `fldNew` 之前隨時被更改的, 以下三行為程式範例片段:

```
static WORD gwWidthA[]={50,70};           // 以 Pixel 為單位
static char *gpcTitleA[]{"姓名","電話"};
static FldDBDes gtDbDes={"ExDb.dbf", 2, gwWidthA, gpcTitleA};
```

定義從 `ExDb.dbf` 資料庫中, 將挑出"姓名"即"電話"兩欄位資料, 以 50 個 pixels 的寬度顯示"姓名"欄位, 以 70 個 pixels 的寬度顯示"電話"欄位, 但會自動把剩下的寬度留給最後一個欄位, 所以"電話"欄位會以 110 個 pixels 寬來顯示。

```
static FldDes gtFldDes={0,&gtDbDes,0,3,2,0,FLD_STYLE_FRAME| \
    FLD_STYLE_TITLE|FLD_STYLE_LINEMODE|FLC_SYTLE_RESERVE};
```

再把資料庫說明變數 `gtDbDes` 加入一個欄位說明變數中第二個值, 上述範例為 `gtFldDes`。其他值依次為

```
typedef struct tagFldDes {
    WORD SrcType;           // DB 時用 0
    void *pSrcDes;         // DB 時用 FldDBDes 資料結構
    BYTE FontID;
    BYTE TitleFontID;     // Style FLD_STYLE_TITLE 時使用
    BYTE FrameColor;      // Style FLD_STYLE_FRAME 時使用
    BYTE Unused;
    WORD Style;           // 參考 Cfield 種類定義
} FldDes;
```

Cfield 種類有:

```
FLD_STYLE_FRAME           // 畫邊框
FLD_STYLE_TITLE          // 畫欄位名稱(title)
FLD_STYLE_LINEMODE       // 可選行模式
```

FLD_STYLE_HIDE_DELITEM // 不顯示已刪除資料
FLC_SYTLE_RESERVE // 會顯示保留項目
FLC_SYTLE_MULTI_RESERVE // 會有多個保留項目

10. 練習題

四、進階程式技巧

1. 前言

看完前幾章節，寫出一個 Penbex PDA 程式大致上不成問題，但是這還不夠。您馬上會碰上很多除錯的問題，功能上的問題，甚至是技術上的問題，令你深感疑惑。本章也許可以幫您解決一些問題，替您解解惑。

在從 VC 開發環境要轉譯成 PDA 上能執行的檔案時，最容易出記憶體讀寫或配置上錯誤的問題，嚴重情況會造成當機。所以本章節介紹一些技巧，第二節介紹如何利用 Penbex 專屬的 MmNewP 及 MmDelP 函式來配置記憶體。而第三節更以記憶體管理的方式，使用記憶體 handler 來配置及使用記憶體，由於所有 handler 配置的記憶體都是由系統直接管理，能達到最佳化使用效果，有些 Penbex 自己提供的物件也是利用 handler 的方式使用記憶體，所以當您需要從該物件取值時，一樣需使用 handler 的方式來叫用。第四節介紹如何使用 TabContainer 做到類似多視窗的功能，由於 PDA 螢幕很小，同一個程式裡想要快速切換 Container(視窗)，最快的方法就是用 Tabcontainer 來達成。第五節介紹另一種特別的 Container，稱作 PopContainer，大部分的對話框或幫助說明視窗都使用 PopContainer 即可，它不會把當前的 Container 關掉，只是疊在前一個 Container 之上，不只減少系統切換視窗的資源，更能保留所有上一個視窗的操作畫面，其實是一舉兩得，當然也只有必要時才會用到它，一般情況還是使用 Container 或 TabContainer。

第六節介紹一組 SDK1.3 才有的語音相關函式，它可以撥放 MP3, WAV 檔等音樂資料。第七節介紹如何使用圖案資料，由於 PDA 程式內部常用到一些圖案，這些圖案都是事先做好利用函式去叫用，所以除了下一章第五節介紹如何使用影像轉換工具來產生這些圖案資料以外，這裡會介紹應用的方法。

如果有人想改變 PDA 的桌面，第八節也會教您怎麼做，桌面的程式源代碼都在 SDK 裡，只要您有興趣就能為自己訂做 PDA 的操作畫面，但它可是個大工程，沒有十足的把握，可別輕易嘗試。

第九節是高手必看的一節，介紹甚麼是 Active Command，也就是所謂的”執行狀態”，每個 PDA 程式都有可能因不同狀況而被系統叫用，例如一般正常被執行稱

為 APP_RUN_NORMAL。系統通知有你的紅外線資料接收到，會用一種稱為 APP_IRDA_NOTIFY 的執行狀態通知執行你的程式。還有當系統重新冷開機(cold reset)時，或程式被第一次下載時，都會先以 APP_RUN_COLD_RESET 的方式被通知執行。程式要被刪除時，也會先被通知用 APP_RUN_KILLED 狀態執行，程式可以在被刪除前把相關不要的檔案也一並刪除。有特別處理不同 ActiveCommand 的程式會顯得功能超強且好用，覺得 PDA 程式沒有挑戰性的人可以考慮本節介紹的方法，讓您程式發揮最大的功能。

2. 記憶體配置不再造成當機!

因為記憶體配置或使用不當, 最容易造成 PDA 當機。本章節仔細介紹 Penbex 的記憶體管理的機制, 如何利用 MmNewP MmDelP 等函式來實現記憶體的分配, 增加或刪除。Penbex OS 中的記憶體操作事實上與標準 ANSI C 的記憶體操作非常類似的(其實, 我們也可以直接使用 Malloc、Calloc 等函式來完成), 在 Penbex OS 上分配並使用記憶體的過程與通用的習慣是完全吻合的。下面我們利用一個程式片斷來說明 Penbex OS 上的記憶體的操作:

```
static BOOL MsgHandler(SysMsg *pMsg)
{
    static char *ptr;

    if ( MT_SYS_CONT_BEGIN==pMsg->type ) {
        ptr=MmNewP(10000);           /* 分配 10K 記憶體 */
        if(ptr==0) FloatBox("錯誤",300,GM_FONT_LARGE);
                                           /* 如果失敗, 顯示錯誤 */
    }
    else if ( MT_SYS_CONT_END==pMsg->type) {
        MmDelP(ptr);                 /* 釋放記憶體 */
    }
}
```

MmNewP 被叫用時, 系統會尋找一個連續的記憶體區塊供該程式使用, 但如果失敗會傳回 0, 否則傳回該區塊的指標。最好要檢查是否成功, 否則非常容易造成錯誤! MmDelP 就像 ANSI C 的 Free 函式一樣用來釋放剛被配置的記憶體, 若失敗也會傳回 0, 成功則傳回 1。

當然還有 MmResizeP 函式類似 ANSI C 的 Realloc 函式用來改變先前配置的記憶體區塊大小, 如果無法依要求變動, 也會傳回 0 表示失敗, 或傳回 1 表示成功。

還有一個特殊的函式叫 MmSizeofP, 用來取的該指標所指的記憶體區塊大小。

3. 記憶體 Handler 又是什麼？

前面我們介紹了如何透過 MmNewP 和 MmDelP 來進行記憶體配置和刪除的操作，本節我們將介紹如何用 MmNewH 來分配記憶體。原因是只利用指標的方式配置記憶體，該記憶體區塊是無法被移動的，當記憶體不足時系統想幫忙都沒有辦法。Penbex OS 雖然小巧，但當記憶體不夠配置時，也會對記憶體做整理碎片的服務，就是英文所謂的 de-fragment。

首先，我們看一個記憶體管理中普遍存在的問題，也就是所謂“記憶體碎片”(memory fragments)的問題。下圖是一段用戶可以使用的記憶體：



在程式執行中，首先要求系統分配 20K 記憶體，分配之後，系統記憶體佔用如下所示：



此時 A 區域的記憶體 程式佔用，然後接著程式連續兩次要求系統分配 20K 的記憶體。分配之後系統記憶體佔用如下所示：



此時 A、B、C 區域的記憶體均 程式所佔用，假設 D 區域的記憶體 剩下的記憶體區塊，大小 10K

接著程式將 B 區域的記憶體釋放，此時記憶體佔用如下所示：



此時的記憶體空間區塊 B、D 兩塊，共有 30K。但是若再要求系統分配一塊

介於 20K 和 30K 之間的記憶體，則系統會發現沒有一個連續且空間的記憶區塊能滿足需求，這就是記憶體碎片問題。有的 OS 在遇到這種問題時，簡單的將記憶體分配函式返回 0 代表配置失敗。但這樣處理是不恰當的，我們可以將記憶體區塊 C 的位置進行移動，使得 A 和 C 兩塊記憶體在實質上是連續的，然後就可以分配一塊 20K 以上 30K 以下的記憶體了。

所以 Penbex OS 為了解決記憶體最佳化配置的問題，提供另一套記憶體管理函式 MmNewH、MmDelH、MmResizeH、MmSoftofH 以及 MmLockH 和 MmUnlockH。

了使用 Penbex OS 中這種對記憶體進行成功配置及改變大小的功能，必須使用記憶體控制碼(handle)來進行記憶體分配，相對應的函式就是 MmNewH。當我們調用 MmNewH 後，系統會分配一塊記憶體區域，並且返回該記憶體區的控制碼(handle)，需要注意的是 handle 並不是記憶區的頭端指標，事實上，handle 是一個指向記憶體區頭端指標的指標。例如我們調用了 hHandle=MmNewH(1000)後，系統的記憶體模組會分配 1K 的記憶體區塊，例如記憶體區塊 A，而 MmNewH 返回的 handle 是指向記憶體區塊 A 的指標，如下圖所示：



我們應該如何使用記憶體區塊 A 呢？當然最直觀的方法是將 *hHandle 作 指向記憶體塊 A 首位址的指標來使用，但我們要求使用 MmLockH(hHandle)這個函式來獲得指向記憶體塊 A 首位址的指標，同時鎖住該區塊資料，然後對該記憶體區塊 A 中的資料進行讀寫，最後再調用 MmUnlockH(hHandle)函式來解開它。記住，使用 handle 的好處是當該區塊沒有被鎖住時，其內容實際存放位址是允許系統搬動的。所以系統才有機會為程式做整理記憶體碎片的服務，對記憶體作最佳化的配置和使用。

許多系統物件都是利用記憶體區控制碼的方式來處理，也就是當它傳回其內容資料時，可能只傳回一個 handler，而不直接是一個記憶區塊指標。所以別忘了使用 MmLockH 函式來取得，最常見的例子就是如何取回單行編輯器的資料：

```
char *pStr;
char *pData[100];
VHANDLE ppH;

if(sedGetTextLength(ContGetObj(10))) { /* 如果有內容 */
    ppH = sedGetEditText(ContGetObj(10)); /* 先取得 handler */
    pStr = MmLockH(ppH); /* 鎖住資料, 並取得指標 */
    Strcpy(pData, pStr); /* 處理該資料內容 */
    MmUnLockH(ppH); /* 不用資料時, 別忘了釋放鎖 */
}
```

4. 多視窗環境怎麼實現？

還記得第二章第一節所提到需重視使用者介面及操作方式的設計,PDA 程式要力求簡潔,而且便於快速點選嗎?如果一個程式的複雜度很高,需要在很多個畫面之間做快速切換,蠻像在 PC 上的程式需要好幾個視窗同時存在,PDA 螢幕這麼小要如何實現呢?Penbex OS 提供了一種特殊的 Container 叫 "Tab Container"。它只有在開始定義時稍有不同,之後的使用和效果都和一般的 Container 相同。但它的顯示方式卻很特別,也就是同時好幾個 Containers 看似都在螢幕上,但同時只有其中之一能被點選且執行,且以點選露在上方的名稱來做 Tab Containers 間的快速切換。

以下是定義一個 Tab Containers 的程式範例片段及結果畫面(如圖 4.4.1):

```
if(APP_RUN_NORMAL==argc) {
    SetTitleTabContEntry(100,conAProcess," A ");
    SetTitleTabContEntry(200,conBProcess,"Container B");
    SetTitleTabContEntry(300,conCProcess,"Container C");
    SetActiveContainer(100);
    while(GetMsg(&msg)) {
        if(!SysProcess(&msg))
            AppProcess(&msg);
    }
}
```

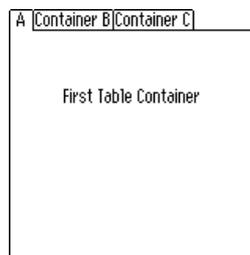


圖 4.4.1 TABCONTAINER 範例

5. 什麼時候用 Popup Container?

前面已經介紹過普通 Container, 還有類似多視窗的 TabContainer。那甚麼是 Popup Container (簡稱為 PopContainer)? 許多人也誤以為每次叫起第二個視窗都要用 PopContainer, 其實不然。一般正常需要切換視窗都用 Container 或 TabContainer 就好, 而且程式一開始在主函式裡就要定義好所有要用的 Containers。

PopContainer 就不同了, 它不須事先定義, 但也要宣告相對應的副程式, 來處理 PopContainer 的所有事件訊息, 所以它要用的時候才直接叫用。其方法如下:

```
static BOOL DeskHelpProc (SysMsg *pMsg)
{
    BOOL fRetVal = 1;

    switch(pMsg->type) {

    case MT_SYS_POP_CONT_BEGIN:
        ContAddObj (btnNewEx (10, 130, 144, 30, 16, "OK", \
            BTNSTYLE_STRING, BTNSTRPOS_MIDDLE, 2, \
            BTN_KIND_BUTTON));
        ContRedraw ();
        break;
    case MT_BACK:
        PopContainerEnd (1);
        break;
    default:
        fRetVal = 0;
        break;
    }
    return (fRetVal);
}
```

通常像程式”操作說明”的視窗就會用 PopContainer, 如以上程式片段是一個 Popup Container 相對應的副程式 DeskHelpProc(), 它不用 MT_SYS_CONT_BEGIN, 而是使用 MT_SYS_POP_CONT_BEGIN, 因為它本身是一個簡易的 Container, 和一

般 Container 截然不同。一般視窗在切換時, 會做整個資源切換, 而且之前的畫面和內容都會清除, 切換到另一個 Container, 之前的 Container 就完全結束。若在某些情況是暫時跳出, 但關閉後還要跳回上一個畫面, 才會用 Popup Container。而且用 PopContainerEnd(return)來結束該視窗, 而非等待結束訊息 MT_SYS_CONT_END。它也可用 return 來回傳值。叫用 PopContainer 時方法如下:

```
PopContainer (DeskHelpProc," Help" , 0);
```

它不需要 ID, 只要直接給對應的副程式名稱(如範例 DeskHelpProc), 以及其視窗名稱(如範例"Help"), 以及最後一個參數還可以用來傳資料。而該資料可從訊息結構裡的 data.p 獲得, 例如 pMsg->data.p。

例如被叫用者: (相同結構資料可以從 pMsg->data.p 獲得)

```
static DlgFontParam *gptFontParam;

if(MT_SYS_POP_CONT_BEGIN==pMsg->type)
{
    gptFontParam=pMsg->data.p;
}
```

而呼叫 PopContainer 者: (可以從資料結構給值, 再由 PopContainer 第三個參數傳遞, 如以下程式範例)

```
DlgFontParam tFontParam;

tFontParam.pbFontInput=&bFontId;
tFontParam.bUserSel=bFontId;

PopContainer(DlgFontSelectCont,(char *)0xFFFFFFFFUL,&tFontParam);
```

SDK 源代碼裡有許多共用對話框都是用 Popup Container 的做法, 值得您參考, 例如 P2ComDlg.c 裡有處理 Group 的對話框, 選字型的對話框, 以及紅外線傳輸等對話框都是由 PopContainer 來完成。

6. PDA 能播放音樂嗎？

答案是: YES, 中環有一款 Penbex OS 的 PDA, 內建 MP3 播放器, 而且還具備 CF 卡擴充槽。使用者能將 PC 上下載之 MP3 音樂檔案存放在 CF 記憶卡中, 再插入 PDA CF 槽中直接播放。其他還有 ADPCM 及 WAVE 聲音檔案可以用同一組函式來播放。WAVE 檔是一般 PC 視窗常用的語音格式, 而 ADPCM 是硬體錄音產生的原始檔。

很可惜這一系列的語音功能並不能在 PDA 模擬器來測試與執行, 因此一定要用 pbx 執行檔在實機上作測試。但 Penbex 也提供一套完整的函式來做錄音或播放音樂的功能, 所有函式以 snd 開頭名稱定義如下:

```
BOOL sndInit(int Command,void *buf,DWORD Len)
BOOL sndClose(void)
```

sndInit()及 sndClose()是用來開啟一個或一組相同格式的音樂檔來使用或關閉。其中 buf 為開檔案名稱或從記憶體播放位置, Len 為其音樂長度。而 Command 指令有以下幾種:

```
SND_COMM_PLAYBACK           // 播放
SND_COMM_RECORDING          // 錄音
SND_COMM_PLAY_WAVE_BUF      // 從緩衝區播放 WAVE 音樂檔
SND_COMM_PLAY_WAVE_FILE     // 從檔案直接播放 WAVE 音樂檔
SND_COMM_PLAY_MP3_BUF       // 從緩衝區播放 MP3 音樂檔
SND_COMM_PLAY_MP3_FILE      // 從檔案直接播放 MP3 音樂檔
SND_COMM_RECORD_BUF         // 錄到緩衝區
SND_COMM_RECORD_FILE        // 錄 WAVE 檔的格式
SND_COMM_PLAY_WAVE_FILE_ARRAY // 播一組 WAVE 檔的歌
SND_COMM_PLAY_MP3_FILE_ARRAY // 播一組 MP3 的歌
SND_COMM_RECORD_ADPCM_BUF   // 錄 ADPCM 格式音樂到緩衝區
SND_COMM_RECORD_ADPCM_FILE  // 錄音成 ADPCM 檔
SND_COMM_PLAY_ADPCM_BUF     // 從緩衝區播放 ADPCM 音樂檔
SND_COMM_PLAY_ADPCM_FILE    // 從檔案直接播放 ADPCM 音樂檔
```

sndStart(), sndStop, 以及 sndPause()則是用來開始播音, 停止或暫停的函式。其定義如下:

```
BOOL  sndStart(void)
DWORD sndStop(void)
DWORD sndPause(void)
```

設播放速度用 sndSetSpeed()函式可以控制, 其中 Speed 速度又分: 兩倍, 四倍, 正常, 慢一倍或 1/4 的速度。定義如下:

```
BOOL sndSetSpeed(int Speed)

SND_PLAY_2X           // 兩倍的速度
SND_PLAY_4X           // 四倍的速度
SND_PLAY_NORMAL       // 正常速度
SND_PLAY_D2           // 1/2 的速度
SND_PLAY_D4           // 1/4 的速度
```

設定播放方向用 sndSetDirection(), 定義如下:

```
BOOL  sndSetDirection(BOOL IsForward)

SND_DIR_FORWARD       // 是正常播放, 其值為 1
SND_DIR_REVERSE       // 倒著播放, 其值為 0
```

用 sndSetMode 來設定播放方式, 其中包含跳上一首, 或跳下一首共 11 種, 定義如下:

```
BOOL  sndSetMode(int Mode)

SND_MODE_ONCE         // 只播一次, 其值為-1
SND_MODE_REPEAT       // 一直重播, 其值為-2
SND_MODE_NEXT         // 下一首, 其值為-3
SND_MODE_NEXT_SONG   // 下一首歌, 其值為-4
SND_MODE_LAST_SONG   // 上一首歌, 其值為-5
SND_MODE_SONG_REPEAT // 重複播同一首歌, 其值為-6
SND_MODE_LIST_REPEAT // 重複播一組歌, 其值為-7
```

```

SND_MODE_SONG_NO_REPEAT // 播一首但不重播, 其值為-8
SND_MODE_LIST_NO_REPEAT // 播一組歌但不重播, 其值為-9
SND_MODE_LIST_ENTRY_DEMO // 歌與歌之間有前奏音樂, 其值為-12
SND_MODE_LIST_LEAVE_DEMO // 歌與歌之間有結束音樂, 其值為-13

```

如果 PDA 可以調大小聲, 軟體用 `sndSetVolume()` 函式也可以調大小聲, 而 `sndGetVolume()` 則用來獲得目前大小聲的值。其定義如下:

```

BYTE sndGetVolume(void)
BOOL sndSetVolume(BYTE Volume)

```

另一個常用的聲音函式是 `sndStatus()`, 它用來詢問目前播放狀態, 或詢問一些訊息, 只要給不同的 `Type`, 就會傳回不同的結果。例如 `sndStatus(SND_QUERY_IS_PLAYING)`, 如果回傳 1 表示, 目前正在播放音樂, 如果是 0, 則否。還有例如詢問 `sndInit()` 時 `Command` 的類別, 叫用 `sndStatus(SND_QUERY_COMMAND)`, 傳回的值會告知目前是哪一種狀況 `SND_COMM_PLAY_MP3_FILE` 或 `SND_COMM_PLAY_MP3_FILE_ARRAY` 等等。其定義如下:

```

DWORD sndStatus(int Type)

```

問題的種類很多, 有 17 種分別說明如下:

```

SND_QUERY_MODE // sndSetMode()是哪一種模式
SND_QUERY_TIME // 目前已播放的時間
SND_QUERY_COMMAND // sndInit()時用的 Command 是甚麼?
SND_QUERY_IS_INIT // 是否執行過 sndInit()?
SND_QUERY_IS_PLAYING // 正在播放嗎?
SND_QUERY_IS_RECORD // 正在錄音嗎?
SND_QUERY_IS_PAUSE // 正被暫停嗎?
SND_QUERY_IS_STOP // 被停止播放嗎?
SND_QUERY_SONG_LEN // 音樂總長度為何?
SND_QUERY_SONG_POS // 目前播放到的位置?
SND_QUERY_SONG_NUMBER // 目前播整組歌的第幾首?
SND_QUERY_IS_DEMO // 有設播放前奏嗎?
SND_QUERY_IS_SONG_REPEAT // 目前是單首重播嗎?

```

```
SND_QUERY_IS_LIST_REPEAT // 目前是整組歌重播嗎?  
SND_QUERY_DEMO_SECOND // 播放前奏長度被定為幾秒?  
SND_QUERY_LIST_ID // 目前歌組的 ID?  
SND_QUERY_TOTAL_SONG_NUMBER // 這組音樂總共幾首歌?
```

最後一個常用函式是當所有音樂相關函式，發生錯誤時，可以利用 `sndGetLastError` 去詢問系統錯誤為何發生? (錯誤種類，請參考 SDK1.3 Reference Manual) 其定義如下:

```
int sndGetLastError(void)
```



```

0x00,0xC8,0x00,0x00,0x00,0x02,0x02,0x20,0x02,0x70,0x10,0x0F,0x80,0x00,0x2A,0xFF,0xFF,0xE0,0x01,0x01,//180
0x6A,0x58,0x15,0x45,0x60,0x55,0x00,0x00,0x00,0x41,0x54,0xC2,0x00,0x00,0x00,0x00,0xA0,0x02,0x70,//200
0x14,0x1D,0x80,0x00,0x27,0x00,0x33,0x60,0x00,0x45,0x6A,0x58,0x19,0x46,0x60,0x55,0x00,0x00,0x00,0x00,//220
0x00,0xFE,0x80,0x00,0xA8,0x00,0x02,0xA0,0x02,0x70,0x10,0x0D,0x80,0x02,0xAF,0xFF,0xFF,0x60,0x04,0x00,//240
0x6A,0x58,0x14,0x01,0x50,0x10,0x00,0x10,0x00,0x00,0x7C,0x00,0x80,0x02,0x02,0xAA,0xA0,0x00,0x02,0x70,//260
0x00,0x0D,0x80,0x02,0x70,0x03,0x37,0x60,0x01,0x05,0x6F,0xA8,0x10,0x00,0x00,0x10,0x10,0x00,0x00,0x00,//280
0x00,0x00,0x80,0x02,0x00,0x00,0x20,0x00,0x02,0x70,0x00,0x0D,0x80,0x02,0x70,0x03,0xF7,0x60,0x00,0x04,//300
0x7F,0xFC,0x11,0xAA,0x40,0x10,0x00,0x00,0x00,0x03,0xFF,0xF0,0x80,0x02,0x00,0x00,0x20,0x00,0x02,0x5F,//320
0xFF,0xF5,0x80,0x02,0x70,0x03,0x77,0x60,0x05,0x01,0x6A,0x5C,0x10,0x44,0x04,0x50,0x00,0x00,0x00,0x20,//340
0x00,0x00,0x80,0x02,0x00,0xFF,0xEA,0xA0,0x02,0x55,0x55,0x55,0x80,0x02,0x70,0x03,0x77,0x60,0x01,0x45,//360
0x66,0x49,0x00,0x55,0x10,0x50,0x2A,0x90,0x00,0x23,0xFF,0xF0,0x80,0x02,0x03,0x55,0x55,0x60,0x02,0x5F,//380
0xFF,0xFD,0x80,0x02,0x5F,0xFD,0x77,0x60,0x00,0x00,0x65,0x49,0x06,0xAA,0x50,0x00,0x10,0x50,0x00,0x20,//400
0x00,0x00,0x80,0x02,0x0D,0x55,0x55,0x80,0x02,0x5F,0xFF,0x0D,0x80,0x02,0x55,0x55,0x76,0xA0,0x01,0x00,//420
0x6A,0x58,0x10,0x00,0x00,0x00,0x10,0x40,0x00,0x23,0xFF,0xF0,0x80,0x02,0x35,0x55,0x56,0x00,0x02,0x5F,//440
0xFF,0x0D,0x80,0x02,0x5F,0xFF,0x76,0x00,0x00,0x45,0x6B,0xA9,0x10,0x04,0x14,0x04,0x10,0x50,0x00,0x20,//460
0x00,0x00,0x80,0x02,0xD5,0x55,0x58,0x00,0x02,0x5F,0xFF,0x0D,0x80,0x02,0x5F,0xF3,0x6A,0x00,0x04,0x54,//480
0x3F,0xFC,0x10,0x04,0x00,0x00,0x10,0x00,0x00,0x20,0x00,0x00,0x80,0x02,0xAA,0xAA,0xA0,0x00,0x00,0xAA,//500
0xAA,0xAA,0x80,0x02,0x5F,0xF3,0x60,0x00,0x04,0x00,0x78,0x3C,0x00,0x00,0x10,0x00,0x2A,0x90,0x00,0x2A,//520
0xAA,0xAA,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xAA,0xAA,0xA0,0x00,0x01,0x05,//540
0x50,0x08,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//560
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//580
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//600
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//608
};

```

所以先用 `tbrNew()` 將把工具條的底圖產生，再用 `tbrAddButton()` 一個個將按鈕區域和它相對應的 `Command ID` 和提示內容(`ToolTip`)加到該工具條，再用 `ContAddObj()` 加到 `Container` 即可。

所以其他跟圖案有關的物件，圖案資料的做法都和以上範例類似，不再此重複介紹。

8. 我能修改桌面程式嗎？

以下 PDA 畫面也許是您最常見的 Penbex OS 桌面畫面，雖然設計的與最流行的 Palm OS 有點神似，桌面功能已經齊全而且好用。但 Penbex 還是開放此空間給 PDA 製造商或垂直應用系統整合商，有自行修改或重新設計的權力。



圖 4.8.1 內建桌面操作畫面

但問題是我能修改嗎？答案是：Yes。第一個步驟，Penbex SDK 在 VC 的環境裡，在 Workspace 視窗中的 File View，將 Desktop 檔案夾從 PenbexOS SDK files 專案中 SDK SYSTEM 目錄搬到 PDA AP 目錄。自然就可以利用 Penbex Developer Studio Add-in 的方式產生 Desktop.pbx 下載程式。第二步驟，需要一個 PU.exe 的工具程式和原來 PDA 同一廠牌款式的 ROM image，就可以將新的桌面程式(Desktop.pbx)利用 PC 上執行 PU.exe 將原本的桌面抽換掉，而產生另一個新的 ROM image。第三步驟，利用 XB.exe 工具程式將 ROM image 燒寫回 PDA Flash ROM 即可，(但不適用於使用

Mask ROM 的 PDA, 因為使用 Mask ROM 的 PDA 無法更新 OS)。XB.exe 的操作畫面如圖 4.8.2。

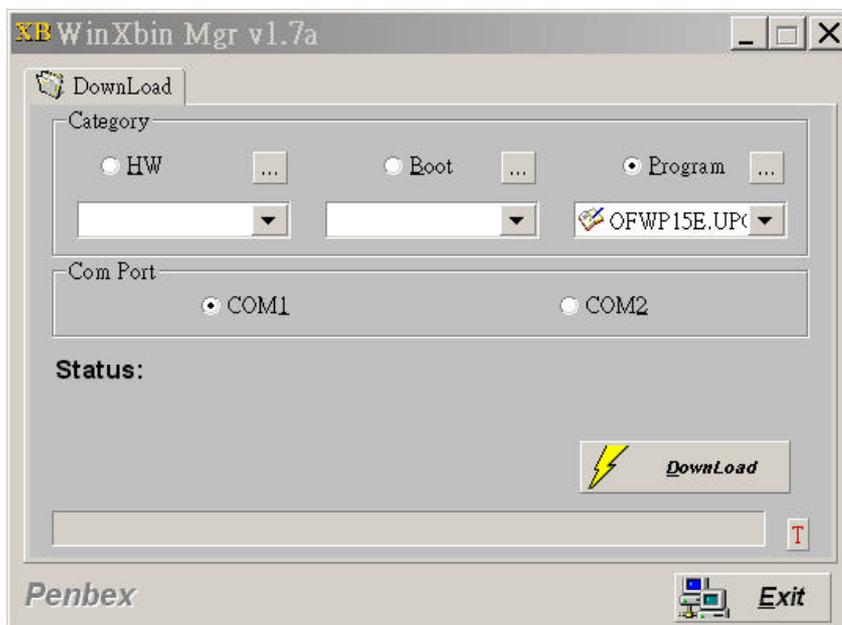


圖 4.8.2 XB.EXE ROM 燒寫器

ROM 燒寫器可以用來對已出貨的 PDA, 或為行業應用而修改的 PDA 做作業系統的更換。先從勾選 Program 及其下方的選項, 選擇欲燒錄的更新檔名, 再勾選是從 PC 的 COM1 或 COM2 與 PDA 的串口埠連線, 在 Download 程式前, PDA 端也要同時按電源和 RESET 鍵進入“Flash Upgrade”的畫面後, 才可以開始更新作業系統。如果想安裝的更新檔不在目前目錄下, 則可按滑鼠右鍵來切換之。

9. 什麼是執行狀態？

“執行狀態”，我們又稱之為”Active Command”。每當程式被叫起時，主函式的第一個參數會告訴程式目前屬於哪一種”執行狀態”？最常見的”執行狀態”就是 APP_RUN_NORMAL(一般正常的執行狀態)，而且每一支程式至少都要有這正常執行狀態。其內容可能先做一些初始值的設定，再定義所有的 Containers 的入口，而且至少要有一個，而後設定第一個啟動的 Container，而且一定要選，接著就是進入事件迴路(Event Loop)。如以下程式片段所示：

```
#ifdef _GNU_ENV
void PenbexMain (unsigned long dwArgc, void *pArgv)
#else
void APDesktop (unsigned long dwArgc, void *pArgv)
#endif
{
    SysMsg tMsg;
    GCONST_ASSIGN (gpDeskVar, gtDeskVar);

    switch (dwArgc) {
    case APP_RUN_COLD_RESET: ← 冷重置執行狀態
        ColdResetDoSub();
        break;
    case APP_RUN_NORMAL: ← 正常執行狀態
        InitialSub();
        SetContainerEntry (CONID_DESK, DeskProc);
        SetContainerEntry (CONID_APP_DEL, DeskApDelProc);
        SetActiveContainer (CONID_DESK);

        while (GetMsg (&tMsg)) {
            if (!SysProcess (&tMsg)) {
                AppProcess (&tMsg);
            }
        }
        EndSub();
        break;
    }
```

```

case APP_RUN_KILLED: ← 程式被刪除時
    APKilledDo();
    break;
case APP_RUN_IRDA_NOTIFY: ← 紅外接收通知
    if(AppIsReEntrant()) {
        DoReEntrantBeamNotify (pArgv);
    }
    else {
        DoNormalBeamNotify(pArgv);
    }
    break;
default:
    break;
}
NOUSE (dwArgc);
NOUSE (pArgv);
}

```

另外比較常見的還有 APP_RUN_COLD_RESET(冷重置執行狀態)、APP_RUN_KILLED(程式被刪除狀態)、APP_RUN_IRDA_NOTIFY(紅外接收通知)。當 PDA 向下鍵(或電源鍵)與重置(Reset)同時被按時，會進入冷重置狀態，而所有軟體會被輪流叫起，且只執行 APP_RUN_COLD_RESET 的狀態，若程式有需要再第一次執行時產生初始檔案或設定初始資料時，可以定義在冷重置的狀態中執行。還有另一種情況會發生冷重置執行狀態，就是在下載完程式之後也會通知該程式執行之。由於 Cold Reset 之後所有 RAM 裡邊的資料全部會流失，所以只有內建在 ROM 裡的程式會先被呼叫做冷重置執行。

當程式被刪除時，程式如果有相關資料庫或檔案留在系統裡，程式有義務清除之。但 PDA 系統理論上始終是單工作業，當某程式被刪除時，系統會通知該程式執行 APP_RUN_KILLED 狀態，好的程式就必須考慮連它產生的相關資料或檔案一並刪除，否則儲存空間就會越來越少，也沒有別的程式會幫你刪除這些檔案而終究佔住系統記憶體或儲存空間資源。

程式向系統登記對紅外線自動接收之副檔名，也常常在冷重置執行狀態時利用 IrObexRegister (char * extname) 函式來完成。以 MP3 播放器為例，登記紅外線若接收到以 "mp3" 為副檔名之檔案時，系統會以紅外線接收通知執行狀態

APP_RUN_IRDA_NOTIFY 叫起該程式來處理之, 通常程式會在該狀態下檢查是否該資料正確且為其所要, 若是, 則將資料取回或儲存, 或甚至跳至該筆資料開始執行或顯示。而紅外線接收之資料則從主函式的第二個參數傳入, 所以處理該資料範例如下:

```
static int ApplrdaNotify (void *pArgv)
{
char *pcData;
IrReceived *ptlrdaObj;
ptlrdaObj = (IrReceived *)pArgv;
pcData = (char *)ptlrdaObj->pReceivedData;
AppProcessData (pcData);           // 處理 IrDA 資料
return (1);
}
```

還有其他執行狀態以其涵義如下:

```
APP_RUN_SYNC_NOTIFY           // 同步時接受通知
APP_RUN_WARM_RESET           // 暖開機時會被通知
APP_RUN_SYSTIME_CHANGE       // 系統時鐘被改變時
APP_RUN_GOTO                 // Global Search到東西時, 會通知該程式
APP_RUN_SEARCH               // Global Search時每支程式都會被通知
APP_RUN_ALARM                // 系統時鐘改了, 如果有加 alarm的程式
                             // 會被通知
```

另外還有與呼叫程式的相關指令如 AppGetIndex()以及 AppCallAp(), 其程式範例如下:

```
int iNdx;
iNdx = AppGetIndex (" Note.pbx");
AppCallAp (iNdx, APP_RUN_SYNC_NOTIFY, 0);
```

以上AppCallAP函式通知 Note.pbx 程式, 該做 sync 的動作。先從檔名得到執行檔的ID, 再透過ID呼叫APP_RUN_SYNC_NOTIFY執行該程式。該函式也可以通知程式執行其他”執行狀態”, 甚至在第三個參數給值, 例如當名片簿軟體接到紅外線管理通知有人beam電子名片給你, 名片簿軟體會收到 APP_RUN_IRDA_NOTIFY,

當程式願意接受後，紅外線管理者又會問用戶是否要跳到該筆資料，如果使用者選“是”，紅外線管理程式會再用 APP_RUN_GOTO 通知名片簿軟體跳到該筆資料並且顯示之。所以擅用系統的服務功能以及處理各種“執行狀態”可以讓程式變的非常友善好用。

10. 如何做自己的專屬的 ROM image?

甚麼是 ROM image? 它就是 PDA 的作業系統, 以 binary 的方式保存在僅讀記憶體裡 (ROM)。該資料是否可以獲得, 對大部分的 PDA 使用者而言並不是很重要, 除非他想更換作業系統。由於 PDA 基本上沒有硬碟, 所以若要更換作業系統, 只能直接更換 ROM 的內容, 只要 PDA 是採用 flash ROM 的記憶體, 就可以利用 ROM 燒寫器直接從 PC 的 RS-232 串口埠對 PDA 作更換。(操作方式, 請參考本章第八節)

當您開發特殊應用軟體, 而且又想作成內建軟體, 或配合適合特殊用途的操作畫面而需要修改桌面, 都能利用 PU.exe 的程式重新作一個 ROM image, 我們可以稱它為 OS 的更新檔。比如說為垂直市場-保險業開發專屬保險專員所用的 PDA, 其內建軟體就不應該只含標準的軟體如電話簿, 計算機, 約會等軟體, 更重要的應該是各種保險公式, 手冊, 人脈管理軟體, 保單印製, 名片管理等等。而且更不應該只能把行業應用軟體放在 RAM 的記憶體中, 在不小心沒電時, 會連應用軟體和客戶資料全部流失。

所以 Penbex 提供軟體開發者或整合軟體公司能自行修改桌面及內建軟體, 但適用各家不同 PDA 的 OS File (如下圖 4.10.1 第一個欄位) 還是必要的。先有了 OS 的核心軟體, 又稱為 OFW (Operation Firmware), 再加上所有要做成內建軟體的每一個執行檔 pbx 加入第三個選單, 輸入您 PDA ROM 的大小以及產出的更新檔名稱和目錄, 選擇 BUILD 按鈕即可產生自製的 ROM image。不管是自己企業用, 還是提供客戶化 PDA 的製造, 能做 ROM image 的修改, 對許多人來講是一大好消息, 對生意人來講更是無限商機。

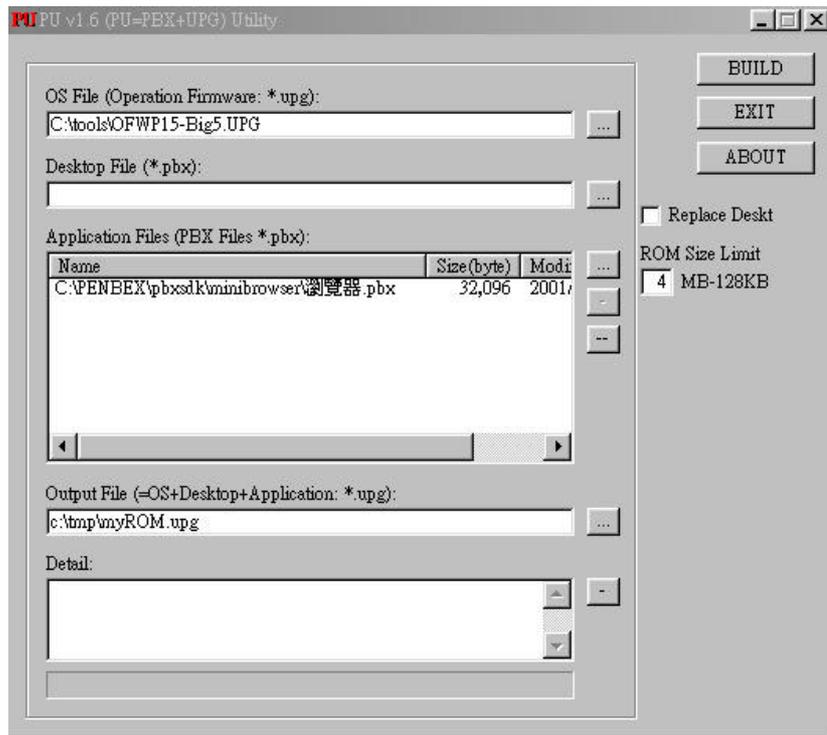


圖 4.10.1 PU.EXE 執行畫面

如圖 4.10.1 所示，如果你已經有新桌面的執行檔，Desktop.pbk，更可以加入第二個欄位，並勾選 Replace Desktop，就可以同時在新的 ROM image 裡更換桌面。加上各家 PDA 製造商不同規格的樣式，您幾乎可以做出自己心目中理想的 PDA。

11. 模擬器使用 C 硬碟檔案的小技巧

常常有人會問到, 為什麼模擬器硬碟裡的檔案, 重新執行時老是清的乾乾淨淨。因為 Penbex PDA 的虛擬硬碟, 就是利用 RAM 記憶體做成的 RAM disk, 在 COLD RESET 冷開機的情況下, 當然所有 RAM 資料都會清除, 重新產生。而每次模擬器重新執行, 就等於實體 PDA 的 COLD RESET, 當然模擬器的 RAM disk 也重新再來, 所有舊資料都不見了。

所以為方便大量資料的測試, 或產出檔案的保留, 都可以用以下的函式來做到。其原理是模擬器本身就是一個 PC 的程式, 所以它當然可以對 PC 的硬碟做讀寫。在適當的時機將 PC 的檔案搬入模擬器的 RAM disk。程式片段範例如下:

```
#ifdef WIN32
    GenEmuFile("C:\\midi\\002.mid", "A:\\download\\002.mid");
#endif
```

因為該函式只適用於 PC 上的模擬器, 所以只在 WIN32 的情況下執行該行, 其目的就是將 C:硬碟裡 midi 目錄下的 002.mid 檔搬至模擬器的虛擬硬碟 A:的 download 目錄下。

相反的若要將模擬器虛擬硬碟的檔案, 複製到 C:硬碟裡, 則利用 GenWin32File 函式, 其用法如下:

```
#ifdef WIN32
    GenWin32File("A:\\download\\002.mid", "C:\\tmp\\002.mid");
#endif
```

其他如將模擬器畫面如何存成 C:硬碟的影像檔, 請參考第一章第七節有詳細說明。

12. Cut 與 Paste

剪貼是電腦使用者常用的編輯功能, 在整台 PDA 裡也是可以在程式之間作 cut & paste。但原則上系統只提供一個所謂的”剪貼區”(copy pool), 且提供相關函式來對剪貼區做動作。

Penbex 提供 4 個與剪貼區相關的函式, 假設所有軟體共用一個系統的剪貼暫存區, CPClean()是用來清除剪貼暫存區的內容。如果想放資料到剪貼區, 利用 CPPutData(Type, p, sz), 它將 p 所指的內容開始, 搬動 sz 長度到剪貼區中。如果想知道目前剪貼區的內容, 可以用 CPGetDataInfo(&pType,&sz); pType 會回傳現在剪貼區內容的類別, sz 會傳回其資料量大小, 大小以 byte 位元為單位。當然也有取回剪貼區內容的函式 CPGetData(pBuf), 系統會把剪貼區的內容寫入 pBuf 之中。以下為這些函式的定義:

```
void CPGetDataInfo(WORD *pType, DWORD *sz)
BOOL CPPutData(WORD Type,void *p,DWORD sz)
BOOL CPGetData(void *buf)
void CPClean(void)
```

Type 的種類有以下四種:

```
CP_EMPTY           // 無資料, 其值為 0
CP_BMP             // 影像資料, 其值為 1
CP_STRING          // 字串資料, 其值為 2
CP_BIN             // 數位資料, 其值為 3
```

所以我用一個簡單的多行編輯器 (med), 和”Cut”, “Paste”, 以及”Copy” 三個按鈕。做一簡單的文字剪貼或拷貝的範例說明:

```
UIOBJ *pMed;
```

```
case MT_BUTTON_COMMAND:
    if (BTN_COPY_ID == pMsg->id) {           // 做 Copy 動作
        pMed = ContGetObj(MED_ID);          // 先從 ID 獲得 pMed 指標
        medGetSelect (pMed, &wStart, &wEnd); // 檢查是否有文字被從第
```

```

wSelectLen = (WORD)(wEnd - wStart); // wStart 個字選到 wEnd
if (wSelectLen) { // 如果有, 做 CPPutData
    hText = medGetEditText (pMed);
    pcBuf = MmLockH (hText);
    pcCopy = MmNewP (wSelectLen+1);
    Strncpy (pcCopy, pcBuf+wStart, wSelectLen);
    pcCopy[wSelectLen] = 0;
    CPPutData (CP_STRING, pcCopy, (DWORD)(wSelectLen+1));
    MmDelP (pcCopy);
    MmUnLockH (hText);
}
}
else if (BTN_PASTE_ID == pMsg->id) { // 做 Paste 動作
    pMed = ContGetObj(MED_ID);
    wLength = medGetTextLength (pMed); // 獲得現有 med 內容長度
    CPGetDataInfo (&wType, &dwSize); // 檢查剪貼區內容
    if (dwSize && ((dwSize + wLength) <= (SIZE_CONTENT+1))) {
        pcBuf = MmNewP (dwSize+1);
        if (wType == CP_STRING) { // 如果剪貼區內容為字串
            CPGetData (pcBuf); // 拷貝至 pcBuf 中
            medGetSelect (pMed, &wStart, &wEnd);
            wSelectLen = (WORD)(wEnd - wStart);
            if (wSelectLen) { // 如果目前有選到字串
                medDeleteStr (pMed); // 刪除 Highlight 的部分
            }
            medInsertStr (pMed, pcBuf); // 原地加入 pcBuf 內容
        }
        MmDelP (pcBuf);
    }
}
}
else if (BTN_CUT_ID == pMsg->id){ // 做 Cut 動作
    pMed = ContGetObj(MED_ID);
    medGetSelect (pMed, &wStart, &wEnd);
    wSelectLen = (WORD)(wEnd - wStart);
    if (wSelectLen) { // 如果有 Highlight 部分
        hText = medGetEditText (pMed); // 從 wStart 的地方拷貝到
    }
}

```

```

        pcBuf = MmLockH (hText);           // pcCut 裡, 長度為
        pcCut = MmNewP (wSelectLen+1); // wSelectLen
        Strncpy (pcCut, pcBuf+wStart, wSelectLen);
        pcCut[wSelectLen] = 0;           // 字串加結尾"\0"
        CPPutData (CP_STRING, pcCut, (DWORD)(wSelectLen+1));
                                           // CPPutData 放入剪貼區
        medDeleteStr (pMed);             // 並把 Highlight 部份刪除
        MmDelP (pcCut);                  // 釋放記憶體
        MmUnlockH (hText);               // handler 開鎖
    }
}
break;

```

值得一提的地方是, 文字編輯器都是用記憶體 handler 的方式處理其內容, 所以當我們要從某編輯器內要存取資料時, 就必須用 ContGetObj(ID)從其 ID 找回該編輯器的物件結構體 pMed (如範例), 在從 pMed 結構指標利用 medGetEditText 取得其記憶體 handler 的值 hText (如範例), 使用 MmLockH 函式可鎖住該記憶區塊並且傳回該記憶體的指標 pcBuf (如以上範例), 有了該物件真正內容的記憶體位置, 就可以對它做讀取。做完動作後別忘了用 MmUnlockH 將該記憶體 handler 開鎖, 免的如果系統需要移動該記憶區塊時無法動作。

13. 練習題

- (1) 您覺得哪種情況下可以使用 MmNewP, 而不需用 MmNewH?

五、我能用 ANSI C 標準副程式庫嗎？

1. 前言

對於 C 語言的程式設計師來說, ANSI C 所定義的一些標準副程式庫應該不會陌生。不論在 Unix 或 PC 的環境下開發 C 語言程式, 都幾乎有 ANSI C 函式可以叫用, 使得開發程式速度更快, 且保持程式相當的可攜性。Penbex 提供幾乎所有與系統底層無關而又常用的 ANSI C 標準函式, 而且使用相同的函式名稱及參數, 為了與標準 ANSI C 函式名稱區隔, 只將第一個字母改為大寫 例如原本是 malloc, 改成 Malloc

另一個不同的地方是檔頭名, 不像在 ANSI C 中需#include 相對應的每一個檔頭名, 例如最常見的<stdio.h>, 或當需要使用數學運算函式時要先#include <math.h>。Penbex DK 環境下只要#include “pbxall.h”, 其中就會代為#include “ansilib.h” (在 VC) 或 ”gansilib.h” (在 GNU), 就可以使用所有 Penbex 所提供 ANSI C 標準所對應的函式庫了。

在第二節討論 ANSI C 浮點運算<math.h>中包含的 Acos, Asin, Atan, Atan2, Ceil, Cos, Cosh, Cotan, Exp, Fabs, Floor, Fmod, Frexp, Hypot, Log, Log10, Modf, Pow, Sin, Sinh, Sqrt, Tan, Tanh 函式。第三節介紹跟整數有關之標準函式。第四節則介紹如何使用 ANSI C 函式作字串的 sort 或 search 字的功能。第五節教您如何使用 Rand()及 Srand()函式產生亂數。第六節介紹很多跟字串運算有關的函式。第七節則介紹如何在數字和字串之間做轉換。第八節介紹如何判斷該 char 是屬於 ASCII 字母的哪一種分類。第九節介紹 ANSI C 標準的記憶體配置方法, 它和上一章第 2,3 節所介紹的 Penbex 記憶體管理都可以使用, 唯一不同是, 使用 ANSI C 的 Calloc, Malloc, Realloc, 以及 Free 函式時要不到記憶體就不失敗不能用, 系統不會替你想辦法, 而且利用這些函式做的記憶體配置也無法移動, 當系統需要”片段整理”時也無法充分利用, 第十節則介紹 Penbex的檔案系統所需的函式, 由於這些函式取代來原本 ANSI C 的標準檔案處理函式, 所以也在本章節做介紹。第十一節則介紹常用到的時間函式, 使用 ANSI C 標準日期和時間函式, 絕對有助於程式跨平台的設計原理。第十二節還完整的介紹除錯時最常用的 Assertion 函式, 讓你程式在執行時有預警的檢查措施, 不只能檢查到錯誤可能發生的地方, 更還能必免嚴重錯誤而造成更當機。

最後一節, 還把 Setjmp()以及 Longjmp()的 ANSI C 函式都做了說明。

2. Penbex 不需再外掛 Math 副程式庫

數學或工程應用甚至於遊戲軟體撰寫常常需要使用到浮點運算, 許多數學運算副函式就變的非常重要。Penbex 作業系統延用 ANSIC 的<math.h>所有標準浮點運算函式, 使得軟體開發或移植更顯得方便容易。

其中三角函數包括: Acos(arc cosine)、Asin(arc sine)、Atan(arc tangent)、Atan2(arc tangent of y/x)、Cos 餘弦(cosine)、Sin 正弦(sine)、Tan 正切(tangent)。

雙曲線函數有: Conh(hyperbolic cosine)、Sinh(hyperbolic sine)、Tanh(hyperbolic tangent)。

指數(exponential)及對數(logarithmic)相關函式有: Exp 指數(exponential)、Fexp 將一個浮點(floating-point)數目分成(normalize fraction)以及(integral power of 2)、Ldexp 傳回一個浮點數目乘以一個(integral power of 2)、Log 傳回 natural logarithm of x、Log10 傳回 base-ten logarithm of x、Modf 將數字分成整數(integer)及分數(fraction)兩部份。

次方(Power)函數有: Pow(x, y)表示傳回 x 的 y 次方、Sqrt(x)傳回 x 的平方根。

還有 Ceil(x)傳回大於 x 的最小整數、Floor(x)傳回小於 x 的最大整數、Fabs(x)傳回浮點數目的絕對值、以及 Fmod(x,y)傳回 x/y 的餘數。

此外 Penbex SDK 裡還增加 Hypot(x,y)及 Cotan 餘切(cotangent)兩個函式。

3. 整數運算相關函式

標準 ANSIC <stdlib.h>裡有四個跟整數有關的函式, Abs, Div, Labs 以及 Ldiv。Abs(int x)回傳 x 的絕對值, Labs(long x)傳回 x 的絕對值但和 Abs 不同的是它用於 long 的整數, 傳回值也是 long 的整數。Div(int x, int y)是用來的到 x/y 的商和餘數, 回傳的值是 div_t 資料結構:

```
typedef struct {
    int quot;           // 商
    int rem;           // 餘數
} div_t;
```

Ldiv(long x, long y)和 Div()很像, 但使用於 long 的整數, 傳回的值也是 long 的整數。其傳回值使用 ldiv_t 的資料結構:

```
typedef struct {
    long quot;
    long rem;
} ldiv_t;
```

以下是使用 Div 的範例:

```
char *cBuf;
int x=7;
int y=2;
div_t z;
z=Div(x, y);
Printf(cBuf, "商=%d, 餘數=%d", z.quot, z.rem);
GmTextOut(0,0, cBuf);
```

結果螢幕上會秀出:

商=3, 餘數=1

4. 收尋與排序

<stdlib.h>還包含了常用的運算法 Bsearch, Qsort, Qsort()函式對任何兩個陣列作排序, 其定義如下:

```
Qsort(void *, size_t, size_t, int (*)(const void*,const void*))
```

使用方法如下: 先定義一個比較函式如 cmp(),

```
static int cmp(const void *p1, const void *p2)
{
    unsigned char c1 = * (unsigned char *) p1;
    unsigned char c2 = * (unsigned char *) p2;

    return(*(unsigned char *) p1 - *(unsigned char *) p2);
}
```

```
char Cbuf[32];
```

執行 Qsort(Strcpy(Cbuf, "mishmash"), 9, 1, &cmp);後 Cbuf的內容會變為"0ahhimmss"

另一個函式為 Bsearch(); 其定義如下:

```
void *Bsearch ( const void *,const void *,size_t,size_t, int (*)(const void*,const void*))
```

```
static char abc[] = "abcdefghijklmnopqrstuvwxy";
```

執行 Bsearch("d", abc, sizeof(abc) -1, 1, &cmp); 回傳值等於&abc[3], 也就是 abc 字串第四個字的位址。

5. 如何產生亂數？

<stdlib.h>也包含兩個亂數產生器, Rand 以及 Srand。Rand()的函式定義如下:

```
int Rand(void)
```

也就是 Rand()會傳回一個正整數的亂數, 在某些情況下, 程式需要隨機變化的值, 就可以利用系統的亂數函式來產生。以下為程式片段範例:

```
char cBuf[32];  
Sprintf(cBuf, "1st 亂數=%d, 2nd 亂數=%d", Rand(),Rand());  
GmTextOut(0,20, cBuf);
```

結果螢幕上顯示: 1st 亂數=18467, 2nd 個亂數=41

但是每次系統重新開機重新執行該程式都會一樣, 不太適合遊戲軟體, 因為每次重新開機重新執行結果都可能一樣。

另一個亂數相關函式為 Srand()定義如下:

```
void Srand(unsigned int)
```

它是用來更改亂數的 seed 值, 與 Rand()搭配使用 大家也常用系統時間當成 seed 值, 讓亂數產生的變化更大。隨著不同執行的時間產生不同的亂數, 所以程式範例可改成:

```
char cBuf[32];  
WORD hour, min, sec;  
  
RtcGetTime(&hour, &min, &sec);           // 取得系統當時的秒值  
Srand(sec);                               // 用當時的秒數來當 seed 值  
  
Sprintf(cBuf, "1st 亂數=%d, 2nd 亂數=%d", Rand(),Rand());  
GmTextOut(0,20, cBuf);
```

此時亂數的結果會變的更無法預期。

6. 有字串及記憶體相關函式真方便

原本在定義在 ANSIC 裡 <string.h> 中有許多與字串和記憶體有關的運作函式, Penbex OS 中也有提供, 對 C 的程式設計師來講, 這是非常好的消息。所有的變數及資料結構都相同, 對軟體移植更是快速又方便。

跟記憶體相關有, Memcpy, Memmove, Memchr, Memcmp, Memset 五種 Memcpy 是用來拷貝某一個字串到另一個字串, 假設字串都在記憶體裡, 所以名稱為記憶體拷貝 Memcpy, 其定義如下:

```
void *Memcpy(void * s1, const void * s2, size_t n)
```

它的意義就是將字串 s2[n]拷貝到 s1[n]。但切記, Memcpy 假設這兩個區塊是不重疊的, 如果有重疊必須用 Memmove 取而代之。其定義如下:

```
void *Memmove( void * s1, const void * s2, size_t n)
```

它的意義也是將字串 s2[n]拷貝到 s1[n], 但不管兩區塊是否重疊, 都可以安全拷貝, 或稱之為記憶體移動。

```
void *Memchr( const void * s, int c, size_t n)
```

它是用來在記憶體區塊中尋找字母, 也就是在 s[n]字串中第一次找到字母 c 的地方, 而該字母 c 就不限定是 ABC..等英文字, 所有 ASCII code, 只要是 unsigned char 即可。

```
int Memcmp( const void * s1, const void * s2, size_t n)
```

它是用來對 unsigned char 的兩字串 s1[n]及 s2[n]作比較, 如果完全相同傳回 0, 如果 s1 比 s2 小傳回 -1, s1 比 s2 大傳回 1, 而 Strncmp 和 Memcmp 很像, 唯一不同是 Strncmp 會停在”\0”。

```
void *Memset( void * s, int c, size_t n)
```

它是用來將 c 值填入 unsigned char 字串 s[n]。

其他的就只針對字串處理, 字串的特性就是以”\0”, 也就是 NULL 為字串的結束字母。所有字串處理就會以”\0”為處理動作的結束。本章節介紹 ANSI C 常用與字串處理有關的函式 `Sprintf`, `Strcat`, `Strchr`, `Strcmp`, `Strcpy`, `Strcspn`, `Strerror`, `Stricmp`, `Strlen`, `Strncat`, `Strncmp`, `Strncpy`, `Strpbrk`, `Strrchr`, `Strspn`, `Strstr`, `Strtod`, `Strtok`, `Strtol`, `Strtoul`。

它們的定義如下: (其個別用法請參考 `Standard C Library` 一書)

```
int Sprintf( char *, const char *, ... )
char *Strcat( char *, const char * )
char *Strchr( const char *, int )
int Strcmp( const char *, const char * )
char *Strcpy( char *, const char * )
size_t Strcspn( const char *, const char * )
char *Strerror( int )
size_t Strlen( const char * )
char *Strncat( char *, const char *, size_t )
int Strncmp( const char *, const char *, size_t )
char *Strncpy( char *, const char *, size_t )
char *Strpbrk( const char *, const char * )
char *Strrchr( const char *, int )
size_t Strspn( const char *, const char * )
char *Strstr( const char *, const char * )
char *Strtok( char *, const char * )
```

其他 Penbex 還多定義了以下三種:

```
char *Str2lower( char * )
char *Str2upper( char * )
int Stricmp( const char *, const char * )
```

7. 資料間的轉換

常常有人會問數字怎麼轉成字串, 答案是用 `Sprintf` 就可以了。但反過來, 字串要換成數字呢? 答案就在以下幾種資料轉換函式:

```
double Atof( const char * )           // 字串轉 double 浮點數
int Atoi( const char * )             // 字串轉整數
long Atol( const char * )           // 字串轉 long.整數
double Strtod( const char *, char ** ) // 字串轉成 double 浮點數
long Strtol( const char *, char **, int ) // 字串轉成 long 整數
unsigned long Strtoul( const char *, char **, int ) // 字串轉成 unsigned long
```

其中 `Strtod`, `Strtol` 以及 `Strtoul` 都會做檢查, 檢查是否超出數字的最大值或最小值。此外 `Penbex` 還增加:

```
char *dw2Str( unsigned long )        // 將 unsigned long 轉成字串
int Toascii( int )                   // 數字轉成 ASCII 碼相對的字母
```

例如 `Toascii('P')` 傳回的值會是 'P' 大寫字母的 ASCII 碼 80。

8. 字的分類檢測及轉換

在早期發明 C 時, 很多程式需要檢查某個字是屬於那一類, 比如說是個數字 0 到 9, 或是英文字母, 或是空白。而其中英文字母又可能是大寫或小寫, 空白可能是單純空白鍵, 也可能是 tab 鍵, 也有可能是跳行。

所以在沒有標準程式庫可以叫用時, 程式設計師常寫:

```
if ('A' <= c && c <= 'z' || 'a' <= c && c <= 'z') // 檢查是否為英文字母
```

或

```
if ('0' <= c && c <= '9') // 檢查是否為阿拉伯數字
```

但有了 ctype 這一系列的標準函式, 在 Penbex OS 環境開發的軟體中就能用

```
if (Isalpha(c))
```

或

```
if (Isdigit(c))
```

程式變的簡單易懂, 可攜性高又不容易出錯。

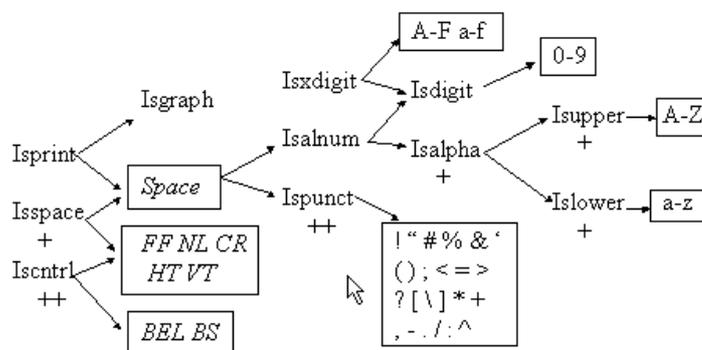


圖 5.8.1 字的分類示意圖

從圖 5.6.1 可以很清楚看出所有的 ASCII 字被依次分類

```
int Isalnum( int )           // 是 0-9, A-Z, a-z 其中之一
int Isalpha( int )          // 是字母
int Iscntrl( int )          // 是控制鍵
int Isdigit( int )          // 是數字
int Isgraph( int )          // 除空白鍵以外的所有可印字
int Islower( int )          // 是小寫字母
int Isupper( int )          // 是大寫字母
int Isprint( int )          // 可印出的字母，包括空白鍵
int Ispunc( int )           // 是符號
int Isspace( int )          // 是空白鍵
int Isxdigit( int )         // 是 0-9, A-F, a-f 其中之一
int Tolower( int )          // 轉換大寫字母成小寫
int Toupper( int )          // 轉換小寫字母成大寫
```

以下不是標準 ctype 函式，但由於中文 Penbex OS 需要而提供：

```
int Isascii( int )           // 檢查是否為 ASCII 碼
int Isleadbyte ( int )       // 檢查是否為中文的第一個 byte
```

9. 如何使用 ANSI C 記憶體管理函式？

記憶體管理相關函式也是 ANSI C <stdlib.h> 中定義的一部分，其中包括 Calloc, Free, Malloc, 以及 Realloc 四個。在 PDA 裡可以使用標準 ANSI C 的記憶體函式，但也可用 Penbex 記憶體管理函式（詳見第四章第二及第三節），其中最大不同在於 ANSI C 的記憶體函式有要到空間，就可以用，要不到時，就不能用；而 Penbex OS 對記憶體管理還做“Memory Defragment”的動作，當你要不到記憶體時，系統會儘可能做記憶體片斷整理，而且當您進一步使用記憶體 handler 時，在 UnLock 的狀態下是允許系統搬移（詳見第四章第三節）。Penbex 物件都用記憶體 handler，所以系統記憶體才會最佳化使用；同時，當從 Penbex 物件存取值時也要用記憶體 handler 的方式做管理。

最常見 C 語言要記憶體的方法是用 *Calloc (size_t count, size_t size) 函式，它會傳回指標，系統挪出 count 個大小為 size 的空間，如果失敗，會傳回 0，所以可以用傳回值做判斷，程式範例如下：

```
ptr = (int *) Calloc(x, sizeof (int));           // 配置 x 個 int 大小的空間
if (!ptr) {
    GmTextOut(30, 30, " Out of memory");       // 如果要不到記憶體，
                                                // 在螢幕上 show 出來
}
```

另一個 Malloc (size_t size) 則跟 Calloc 很像，但 Calloc 會代設初值為 0, Malloc() 要自己設初值。它的參數 size 決定向系統要的記憶體空間。Realloc(void * ptr, size_t size) 則是當上一次所要的記憶體不夠時，若需再多要一些，才使用它。從 ptr 所指的記憶體位置，再多要 size 大小。成功的話傳回新位置，但失敗時則傳回 0。

Free(void *ptr) 則用來釋放 ptr 所指的記憶體空間。不管是 Calloc, Malloc, 或 Realloc 獲得的指標，都可以用 Free() 函式來釋放之。

切記，ANSI C 的記憶體函式最好不要與 Penbex 記憶體管理函式合用，容易造成不明錯誤。

10. 如何使用檔案?

Penbex OS 提供類似 ANSI C 的檔案函式, 但不盡相同, 也不用類似的函式名稱及相同的參數。本章節對檔案的開啟, 關閉, 與讀寫都有詳盡的介紹。跟檔案有關的函式有: FileOpen, FileWrite, FileRead, FileClose 以及 FileTell 與 FileSeek 等。

FileOpen(const char *PathName, WORD flag, WORD mode) 第一個參數為檔案名稱, 可含目錄路徑(pathname), 第二個參數為開檔的方式(flag)有以下八種:

FILE_FLAG_READONLY	// 開啟檔案且唯讀
FILE_FLAG_READWRITE	// 開啟且允許讀寫
FILE_FLAG_NOSHAREANY	// 如果檔案已開啟, 這次會傳回失敗 // 當檔案開啟著, 其他 Open 會失敗
FILE_FLAG_APPEND	// 每次寫, 會從 EOF 開始
FILE_FLAG_CREAT	// 如果檔案不存在, 產生它
FILE_FLAG_TRUNCATE	// 如果檔案已存在, 裁短它
FILE_FLAG_EXIST	// 如果檔案已存在, 會傳回失敗
FILE_FLAG_NOSHAREWRITE	// 如果已開啟且要寫, 這次會傳回失敗 // 當檔案開啟著, 其他打開要寫會失敗

第三個參數為讀寫模式, 讀寫模式分為以下兩種:

FILE_MODE_READ	// 唯讀模式
FILE_MODE_READ_WRITE	// 讀寫模式

所以以下FileOpen的範例為開啟file.ini檔做讀寫, 如果檔案不存在, 產生它; 但如果檔案已存在, 可以裁短它(也就是說長度歸零)。當回傳值tFh不為負值, 表示成功。

```
FHANDLE tFh;
```

```
tFh = FileOpen ("file.ini", FILE_FLAG_CREAT \
                |FILE_FLAG_READWRITE|FILE_FLAG_TRUNCATE, \
                FILE_MODE_READ_WRITE);
```

跟檔案系統(file system)有關的函式如下, 其中跟目錄(directory)有關包含:

```
FileGetCurrentDir(const char *DriverName,char *ReturnBuf)
FileSetCurrentDir(const char *PathDir)
FileNewDir(const char *szPathName)
FileDeleteDir(const char *szPathName)
```

以及跟磁碟機(driver)有關的包含:

```
FileDiskAbort(const char *DriverName)
FileDiskOpen(const char *DriverName)
FileDiskClose(const char *DriverName)
FileDiskFormat(const char *DriverName, FMTEASY *ptformat)
FileDiskDefrag(const char *cDiskPath,int iDefragMode)
FileGetFreeSpace(const char *DriverName)
FileGetDefaultDriver(void)
FileSetDefaultDriver(short DriverNo)
```

由於 Penbex PDA 有不同規格的磁碟機(都是虛擬磁碟), 其中可為, RAM disk, NAND Gate, CF 卡 Flash 記憶體, 甚至於 CF 卡的 Mirco Drive(微硬碟機)等等。所以有以上幾個磁碟相關函式可以使用。

11. ANSI C 與時間有關的函式

時間, 是程式常會用到的值, 如果使用標準函式獲取系統時間或日期的值, 該程式未來的可攜性也相對的提高很多。

其中用到 `clock_t`, `time_t`, 以及 `size_t` 等 types 的定義。還有一個 `struct tm` 的資料結構定義如下:

```
int tm_sec;      // 秒, 值從 0 到 59
int tm_min;      // 分, 值從 0 到 59
int tm_hour;     // 小時, 值從 0 到 23
int tm_mday;     // 每個月第幾天, 值從 1 到 31
int tm_mon;      // 一年的第幾個月, 值從 0 到 11
int tm_year;     // 自 1900 年來的第幾年
int tm_wday;     // 星期天起第幾天, 值從 0 到 6
int tm_yday;     // 自 1 月 1 日起算第幾天, 值從 0 到 365
int tm_isdst;    // 日光節約時間是否開啟
```

時間相關函式有: `Clock`, `Difftime`, `Mktime`, `Time`

時間置換函式有: `Asctime`, `Ctime`, `Gmtime`, `Localtime`, `Strftime`,

Penbex 本身就支援中文系統, 而中國人又分陽曆與陰曆兩種時間, 所以增加了 `Lunar2Solar` 與 `Solar2Lunar` 兩個換算函式。 `Lunar2Solar(*pYear, *pMonth, *pDay, IsLeapMonth)` 用來轉換農曆的年月日成為陽曆的年月日, 而利用 `IsLeapMonth` 告知是否開年有閏月。 `Solar2Lunar(*pYear, *pMonth, *pDay, *pIsLeapMonth)` 用來轉換陽曆年月日成為陰曆的年月日, 而且 `pIsLeapMonth` 值表示是否有閏月。農曆年份定義從 1902 到 2071 年, 而陽曆從 1904 到 2099 年。

其他還有額外定義幾個時間有關的函式, 例如 `SetGmRefTime(seconds)` 用來設定系統所在地與 GMT(格林威治時間)差幾秒、 `CalcYearDay(year, month, day)` 用來推算出某年某月某日是該年的第幾天、 `CalcWeekDay(year, month, day)` 用來推算出某年某月某日是星期幾、 `CalcWeekDay2(year, yearday)` 用來推算出哪一年的第幾天是星期幾。

12. Assertion 是甚麼？

Assertion 就是”說明”的意思，也就是用來”說明”錯誤發生的地方和原因。標準 ANSI C 程式庫中有提供一個巨指令 `assert`，它是被定義在 `<assert.h>` 裡。而標準的 `assert` 是用來加一個判斷式於除錯的程式中，利用 `NDEBUG` 是否被 `#define` 來決定 `assert(expression)` 功能是否開啟，其中的 `expression` 就是其判斷式。

```
#undef NDEBUG          // 將 assertion 功能打開
```

或

```
#define NDEBUG         // 則將 assertion 功能關閉
```

若其中判斷式結果為 `false`，也就是值為 0 時，`assert` 會告知程式用戶或開發者哪一個程式碼錯在哪一行等訊息，對程式開發者而言，是非常好的一個除錯方式。

因此，Penbex 開發環境裡，在 VC 的環境下也提供了一個類似的函式，稱為 `Assertion`，它的功能和 ANSI C 的 `assert` 很像，但它卻有三個參數依次為 `message`，`file_name`，`line_number`，且參數定義為 `Assertion(char *, char *, int)`。另外也可以使用 `Assert(exp)` 函式，也因此錯誤發生時螢幕會顯示哪一個檔案，第幾行的哪一個判斷 `exp` 發生錯誤。它被定義為：

```
#define Assert(exp)    ((exp)||Assertion(#exp,__FILE__,__LINE__))
```

以下程式範例中在增加新按鈕物件前加這一行 `Assert` 做除錯輔助，系統在除錯狀態如果有錯誤，執行時會有類似圖 5.12.1 的訊息視窗會因此發出，但經過 GNU 轉譯後的執行檔，預設值會自動關閉除錯狀態，也就是說 `Assert` 該行在最終 `pbx` 執行檔時，不會再發生作用，也因此最終執行檔相對變小。

目前開發環境中的作業系統部分設有上千個 `Assert` 錯誤檢查點。因此，如果看到類似的警告訊息來自系統的原始碼，原因大多數來自應用程式呼叫時造成，而非系統有蟲(bugs)，請查閱 SDK1.3 手冊中的 `Assertion` 對照表，判斷其錯誤來源。

使用 Assert 函式的程式範例片段如下：

```
#ifndef GM_LCD_HEIGHT
#define GM_LCD_HEIGHT      160    // 螢幕高度
#endif
#ifndef GM_LCD_WIDTH
#define GM_LCD_WIDTH       160    // 螢幕寬度
#endif

int x=50;
int y=150;
int w=100;
int h=20;

Assert(x>=0 && y>=0 && w>2 && h>2 && x+w<=GM_LCD_WIDTH && \
      y+h<=GM_LCD_HEIGHT);
pLabel = labNew( 40, x,y,w,h, " OK2 " , LABSTYLE_STRING, \
      LABSTRPOS_MIDDLE, GM_FONT_MIDDLE , LABFRAME_YES );
```

Assert 該行用來檢查如果之後要在(x, y)的位置畫一個寬為 w, 高為 h 的按鈕 (button)是否會超出螢幕？結果是超過的，在執行時就會發生圖 5.12.1 的 APPLICATION FATAL ERROR 的警訊視窗。



圖 5.12.1 ASSERTION 訊息視窗

13. 其他 Penbex 支援的 ANSI C 函式

類似在 ANSI C 標準裡<setjmp.h>所提供兩個函式 Setjmp (jmp_buf env) 以及 Longjmp (jmp_buf env, int val)。有一個它們共用的資料結構叫 jmp_buf, Setjmp 是用來保存 Longjmp 前的呼叫環境(calling environment)值於某一個 jmp_buf 資料結構, 如 env。而且當它是第一次直接叫用時回傳回 0 值, 或是最近一次 Longjmp 呼叫時所留下來的值 val。每次 Longjmp 被呼叫時會取回上一次 Setjmp 對同一個 jmp_buf 所保存的呼叫環境值。利用這種機制, C 語言程式才有辦法做到所謂的”nonlocal goto”或”label variables”寫法。也就是函數內會 goto 到另一個函數的某個位置, 或 goto 到可能因情況不同而改變的位置。該類似語法在 Pascal 或 PL/I 語言都直接能做到, 但在 C 語言中就需靠 setjmp 系列的函式搭配組合來完成之。

詳細使用方法, 請參見”The Standard C Library, P.J. Plauger, PRENTICE HALL 1992”一書, 內有詳細說明。

14. 練習題

--- 遊戲篇 ---

六、遊戲程式設計範例

1. 前言

遊戲程式已經是最近最紅的軟體工業，日本早已領先全球，包括 GameConsole 的市場，像 N64, DreamCast, PS2 是屬於桌上型遊戲機，Game Boy Advance, GameGear 等手持式遊戲機，還有日本 DoKoMo i-mode 手機上的遊戲，以及未來 Java (爪哇)手機的 Java 遊戲，除了 Java 手機以外，不論是硬體或軟體已經都很成熟，而且遊戲軟體開發人才及公司非常多且經驗豐富。就連鄰國的韓國都已經全國動員寫軟體，當成電子業的下一個新興工業，台灣網路咖啡廳的網路連線遊戲軟體，除了少數台製以外，都是韓國進口。我們能不再重視提倡軟體遊戲工業的重要性嗎？

手持式裝置包括 PDA, 手機，和傳統的手持式遊戲機，加上未來無線上網的普及，想必無線網咖將會是最熱的一行，也就是利用手機加 PDA 直接在網路上玩遊戲，而且是隨時隨地玩，不需到網咖才能玩。舉例日本 i-mode 手機遊戲就有一種網路釣魚，是付費的釣魚場喔！先從手機選擇甩竿拋下魚餌，再等魚上鉤時電話公司會 Call 你，“甚麼魚上鉤了”，如此一個虛擬網路遊戲的世界很快就來臨了。趕快來寫 PDA 及手機遊戲軟體吧！

本章節先以 Penbex PDA 的開發環境來介紹遊戲所需要的一些函式，例如螢幕觸控的處理，Timer 的用法，圖像的製作及顯示，如何利用按鍵設計打擊遊戲，PDA 如何發出聲音或 MIDI 音樂，還有如何透過紅外線互傳資料，適合紅外線對打遊戲所使用等。其中第七節，以電子貓作動畫程式範例來說明，第九、十兩節是乒乓球所需要的兩個功能製作範例。至於網路遊戲以及如何做多人互動，例如聊天室(Chat)等軟體需利用到網路程式及 Client-Server 的架構設計方法，在下冊第九章網路程式設計原理中會討論，就不在本章重複介紹。

2. 遊戲程式設計基本原理

遊戲程式種類繁多, 在 PDA 上的遊戲除了要考慮, (1)螢幕解析度不高, 繪圖速度也不及 PC 快, 顏色也不多, 甚至是黑白或灰階的情況下, 如何處理。 (2)CPU 運算速度很慢, 遊戲內容應如何設計。 (3)儲存區與記憶體皆很小, 遊戲劇本及內容該如何編寫。 (4)PDA 音響也不如手持遊戲機專業, 該如何應用聲音技巧, 也值得深思。其他基本上和 PC 或遊戲機原理相同。

我個人嘗試在 PDA 上開發過, 電子寵物、撲克牌遊戲: 梭哈、乒乓球、紅外對打乒乓等, 也看過許多棋盤遊戲, 智慧遊戲, 打擊遊戲, 體育遊戲, 甚至於 RPG 角色扮演等在 PDA 開發, 基本上都能做到。但最常碰到的問題會是, CPU 不夠快, 以及顯示速度太慢。像五子棋或象棋, 就需要設計成有可選擇對方的思考難度, 在 PC 模擬器上執行也許很快, 因為模擬器是採用 Intel x86 CPU 在運算。當程式真正在 PDA 上執行時, 使用運算能力可能少 100 倍的 Motorola DragonBall CPU 會變很慢。又要回到最基本的運作邏輯去改良, 以達到程式的可用性。例如打坦克車或打飛機遊戲, 在顯示速度與碰撞判斷之間所造成的問題要取得平衡點, 最後可能在複雜度上作最佳化的妥協。

所以遊戲程式不外乎利用最有限的資源, 製造出最有趣好玩的遊戲。在打擊遊戲裡, 利用 Timer 在固定時間的迴路裡一直重複做, 接受訊號, 運算, 判斷, 以及顯示等動作。有人將它做成通用的部分, 稱之為 Game Engine (遊戲引擎)。只要改變遊戲規則, 套用同樣的遊戲引擎即可生產出不同的遊戲程式。在智慧型遊戲裡, 就要多加一個 AI (人工智慧) 引擎在裡面, 來應付外來變化作不同的反應, 或讓使用者覺得對手很聰明。基本上遊戲的內容比程式設計的難易度來的重要, 所以編劇本, 做音效, 動畫及美工, 以及人機介面, 都要先考慮且特別用心規劃, 才能製造出一流的 PDA 遊戲軟體。

3. 怎麼抓到筆觸 LCD 螢幕的位置？

大部份的遊戲軟體都會有與玩遊戲的人互動之方式，其中利用筆觸控 LCD 螢幕是其中之一種，利用上下左右鍵操作也是一種方法。第九節會介紹利用按鍵的程式設計方法，本章節則針對 LCD 的觸控程式設計方法介紹為主。

相關的事件(Events)就是 MT_PEN_DOWN, MT_PEN_UP, MT_PEN_MOVE, 及 MT_PEN_REPEAT 四種。其中 MT_PEN_DOWN 就是當筆觸到 LCD 螢幕時，系統對當時正在執行的程式所發出的事件訊息(Message)，其實更嚴格來講是對該程式正在執行的視窗畫面。MT_PEN_UP 則是當已接觸在螢幕上的筆拿開螢幕時，系統會發出的訊息。當筆在螢幕上滑動而不離開螢幕時，系統會一直發出 MT_PEN_MOVE 的訊息。當筆點在螢幕上但沒移動時，系統則會一直發出 MT_PEN_REPEAT 的訊息。但 MT_PEN_MOVE 及 MT_PEN_REPEAT 訊息要看程式是否要求系統送出該兩種訊息，利用 EnablePenMoveMsg(x)及 EnablePenRepeatMsg(x)，其中 x=0 代表關畢，也就是說從該行程式執行以後，不要再發出 MT_PEN_MOVE 或 MT_PEN_REPEAT 的訊息。當 x=1 剛好相反，表示請當事件發生時系統可以開始發出相對的訊息。

```
static BOOL conProcess (SysMsg *pMsg)
{
    register BOOL retValue=0;
    char cBuf[64];

    switch(pMsg->type) {
    case MT_SYS_CONT_BEGIN:
        EnablePenMoveMsg(1); /* 通知系統開始可以送 Pen Move 訊息 */
        SetPenMoveTick(100); /* 每一個 Tick 10ms, 設 100 表示 1 秒一次 */
        /* 如果設為 0, 用預設值代替 */

        break;
    case MT_SYS_CONT_END:
        EnablePenMoveMsg(0); /* 通知系統不要再送 Pen Move 訊息 */
        break;
    case MT_PEN_MOVE:
        x=pMsg->data.pos.x; /* 獲得筆觸螢幕時 x 的座標值 */
        y=pMsg->data.pos.y; /* 獲得筆觸螢幕時 y 的座標值 */
        Sprintf(cBuf,"X=%d,Y=%d",x,y);
    }
```

```
    GmTextOut(40,40,cBuf); /* 顯示結果在螢幕 (40,40) 的位置上 */
    retValue=1;
    break;
default:
    break;
}
return retValue;
}
```

4. Timer 的應用?

先前已提到, Timer 對遊戲軟體是必要的工具。PDA 提供程式一個方法來取得固定時間點, 比如程式需要倒數計時並且顯示所剩時間在螢幕上, 就可以事先告訴系統每一秒, 也就是 1000 微秒(ms), 通知該程式一聲。當程式需要倒數時, 請系統開始通知, 程式就會再每隔一秒, 收到系統的 Timer 通知, 再把時間減一並顯示在螢幕上。直到時間為零時, 程式可以告訴系統, 停止通知。這樣的動作就是典型 Timer 的應用。

我們可以利用 SetTimeMsg(Xm)函式來事先告訴系統, 每 Xm 微秒通知一聲。當需要開始通知時才執行 EnableTimerMsg()函式, 直到不需要 Timer 時才執行另一個相對函式 DisableTimerMsg()為止。系統是利用 MT_TIMER 這個事件訊息通知該程式, 所以處理方法跟其他事件訊息一樣, 在事件迴路中處理 (第二章第五節對事件迴路有詳細說明)。切記, MT_TIMER 會以固定的時間送到訊息排(queue), 但是否馬上處理就由程式來決定, 在迴路的設計上一樣會影響程式的結果。

```
switch(pMsg->type) {
    case MT_SYS_CONT_BEGIN:
        SetTimerMsg(1000);
        break;
    case MT_BUTTON_COMMAND:
        EnableTimerMsg();
        break;
    case MT_SYS_CONT_END:
        DisableTimerMsg();
        break;
    case MT_TIMER:
        do();                // 做某些事情
        break;
}
```

以上程式片段為典型使用 Timer 的方法, 如果只是要簡單讓程式停幾微秒, 就可以不必那麼大費周章, 只要較用 Sleeping(Xm)即可, 程式會停在該行 Xm 微秒。

5. 影像轉換工具

遊戲軟體常需要用到許多圖片, 不管是底圖、或是移動的物體、或甚至遊戲面板, 都屬於影像的一種, 在 PDA 的程式裡, 通常都會事先把所有所需圖檔轉換成資料格式, 直接叫用 GmDrawBmp 函式來顯示。(例如圖 6.5.1)

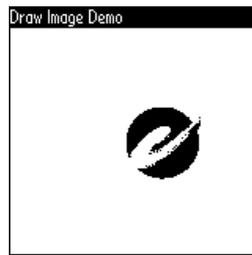


圖 6.5.1 程式貼圖範例

該工具名稱為 Bmp2ArrayPro.exe (可從網路下載), 它可以把黑白或彩色的 Bmp 影像檔轉換成陣列方式的文字資料, 它代表的就是該圖的黑白或灰階的內容。如圖 6.5.2 .執行 Bmp2ArrayPro 的畫面來說明, 將 Bmp 檔名選入第一個欄位, 並替輸出檔名填入第二個欄位, 選擇 Gray(灰階)或 B/W(黑白), 點選 Start Convert 即可。

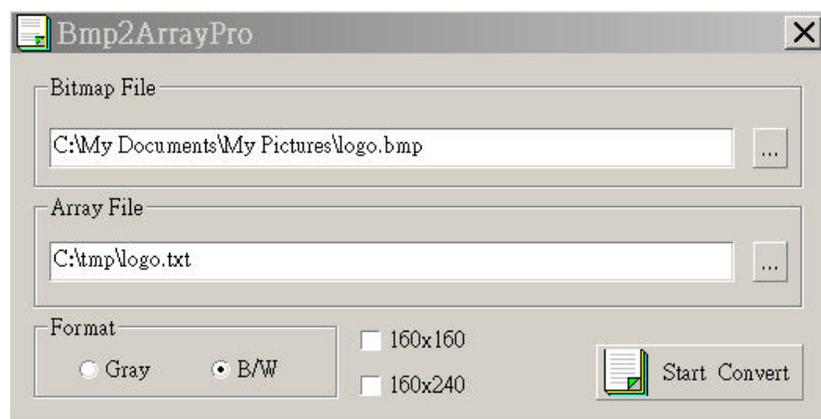


圖 6.5.2 BMP2ARRAYPRO.EXE

如果產生 Convert OK 對話框表示成功, 輸出檔中如圖中範例 logo.txt, 內容如下: 其中第一行記載該圖檔來源。第二行表示轉換後圖檔資料的顏色, B/W 代表黑白(兩色)、Gray 代表 4 色灰階、其他會是彩色的各種色階(未來彩色 PDA 才會支援)。

第三及第四行記載圖檔的寬和高，以像素(pixel)為單位。範例中 logo.bmp 檔為寬 47 pixels 與高 47 pixels。第五行表示以下陣列以 WORD 為單位的方式表示，所以圖檔變數 const WORD Logo[] 才會以 WORD 單位來定義。

```
C:\My Documents\My Pictures\logo.bmp
Colors : B/W
Width : 47
Height : 47
WORD
0x626D,0x0200,0x002F,0x002F,0x0000,0xFFFC,0x0000,0x0007,0xFFFF,0x0000,// 10
0x001F,0xFFFF,0xC000,0x003F,0xFFFF,0xF000,0x00FF,0xFFFF,0xF800,0x01FF,// 20
..(忽略 30 到 130, 其內容與下述 Logo[]相同)
0x001F,0xFFFF,0xC000,0x0007,0xFFFF,0x0000,0x0000,0xFFFC,0x0000,0x0000,//140
0x0700,0x0000,0x0000,0x0000,0x0000,//145
```

將以上範例 logo.txt 第六行起內容擷取到程式中該圖所定義的變數陣列內容如下：

```
const WORD Logo[]={
0x626D,0x0200,0x002F,0x002F,0x0000,0xFFFC,0x0000,0x0007,0xFFFF,0x0000,// 10
0x001F,0xFFFF,0xC000,0x003F,0xFFFF,0xF000,0x00FF,0xFFFF,0xF800,0x01FF,// 20
0xFFFF,0xFC00,0x03FF,0xFFFF,0xFF02,0x07FF,0xFFFF,0xFF00,0x0FFF,0xFFFF,// 30
0xFF82,0x0FFF,0xFFFF,0xFFC2,0x1FFF,0xFFFF,0xFF8E,0x3FFF,0xFFFF,0xFF1E,// 40
0x3FFF,0xF81F,0xFF3C,0x7FFF,0xE00F,0xFC7C,0x7FFF,0x800F,0xF8F8,0x7FFF,// 50
0x000F,0xE1F8,0xFFFC,0x001F,0xC0F8,0xFFF8,0x003F,0x85FC,0xFFF0,0x005F,// 60
0x01FC,0xFFE0,0x00FE,0x13FC,0xFFC0,0x00FC,0x07FC,0xFF80,0x03F0,0x0FFC,// 70
0xFF00,0x07E0,0x3FFC,0xFE00,0x3FC0,0x3FFC,0xFE00,0x7F01,0x7FFC,0xFC00,// 80
0xFE02,0xFFFF,0xF800,0xF805,0xFFFF,0xF001,0xD403,0xFFFF,0xE000,0x502F,// 90
0xFFF8,0x6003,0x000F,0xFFF8,0x4000,0x003F,0xFFF8,0x6000,0x007F,0xFFF0,//100
0x0000,0x01FF,0xFFF0,0x0000,0x07FF,0xFFE0,0x0000,0x1FFF,0xFFE0,0x0000,//110
0x3FFF,0xFFC0,0x0000,0xBFFF,0xFF80,0x000A,0xFFFF,0xFF00,0x0007,0xFFFF,//120
0xFF00,0x01FF,0xFFFF,0xFE00,0x00FF,0xFFFF,0xF800,0x003F,0xFFFF,0xF000,//130
0x001F,0xFFFF,0xC000,0x0007,0xFFFF,0x0000,0x0000,0xFFFC,0x0000,0x0000,//140
0x0700,0x0000,0x0000,0x0000,0x0000,//145
};
執行 GmDrawBmp(x, y, Logo); 即可將該圖案顯示在螢幕上 x, y 的位置。
```

6. 繪圖相關 APIs

本章節簡單敘述每一個 Gm 開頭跟繪圖相關的函式群;

跟畫筆相關的函式:

```
GmSetPenMode(mode) // 設畫筆的模式, 且傳回之前筆的模式
GmSetPenColor(color) // 設畫筆的顏色, 且傳回之前筆的顏色
GmSetPenBkColor(color) // 設畫筆的背景顏色, 且傳回之前筆的背景顏色
GmSetPenPattern(value) // 設畫筆的圖案, 且傳回之前筆的圖案
GmSetPenWidth(width) // 設畫筆的寬度, 且傳回之前筆的寬度
GmSetPenPos(x,y) // 移動畫筆到螢幕座標 x, y 的位置
GmGetPenPos(&x,&y) // 用&x, &y 取回目前畫筆的位置
GmDrawDot(x,y) // 在螢幕座標 x, y 的位置畫一個點
GmDrawLine(x1,y1,x2,y2) // 從螢幕 x1, y1 畫到 x2, y2 的位置
```

跟筆刷相關的函式:

```
GmSetBrushPattern(value) // 設定筆刷的圖案
GmFillRect(x, y, w, h, IsCorner) // 畫寬 w 高 h 的矩形於 x, y 的位置
GmRevertRect(x, y, w, h, IsCorner) // 反色畫寬 w 高 h 的矩形於 x, y 的位置
GmFillScreen(ColorLevel) // 以 ColorLevel 顏色填滿螢幕
```

跟畫字型 and 文字顯示相關的函式:

```
GmSetFont(fontID) // 設字型, 直到另一次設定值才會變
GmTextOut(x,y,*szStr) // 將 szStr 所指的字串畫在螢幕 x, y 的位置
GmTextOutCnt(x,y,*szStr,count) // 在螢幕 x, y 的位置畫出 szStr 字串的前
// 幾(count)個字
GmGetCharWidth(ch) // 得到 ch 字的寬度 (因為英文字寬度不一)
GmGetStrWidth(*szStr) // 得到整格 szStr 字串的總寬度
GmGetFontHeight(fontID) // 得到 fontID 字型的高度 (字型高度是固定)
```

7. 電子寵物貓範例

如圖 6.7.1 所示, 一隻從 PC 的程式移植過來的電子寵物, 我們在本章介紹其設計基本原理。其中重點擺在離開該程式時如何把當時狀態儲存起來, 下次再進入時會從上一次的地方開始, 這種設計方法再遊戲軟體非常需要。當遊戲玩到一半時, 突然需要關閉本程式跳到其他程式執行, 由於 PDA 上不做多工多視窗處理, 所以必須結束掉正在執行的遊戲程式, 只好把當時的所有狀況以檔案方式儲存起來, 下次再執行時會從同一個檔案讀回, 寫回所有變數再執行即可模擬其效果。(很像繼續從剛才離開時的地方再玩起)



圖 6.7.1 電子寵物

最簡單的方式把所有全區域變數以單一結構體定義, 在 MT_SYS_CONT_END 程式結束時以該結構體整個寫到檔案中, 我們也能稱之為“初始檔”, 習慣以 ini 為副檔名。在 MT_SYS_CONT_BEGIN 程式第一個視窗開始時, 做 ini 檔的檢查, 如果不存在, 就表示程式是第一次執行, 自己重新開始即可。但如果 ini 檔存在, 表示上次已執行到某個狀態, 就把 ini 檔的值讀回做程式的初始值, 從該初始值開始執行。

Neko 寵物貓為例, 牠有 33 種狀態圖案, 還有現在的位置, p->m_ptPosition_x, p->m_ptPosition_y 座標值。p->mouse_x, p->mouse_y 代表筆點到的位置 (也就是貓要追的位置), p->m_Action 記載目前貓的狀態, 所有變數都在 p 的資料結構裡。

```
static char gclniName[32] = "A:\\download\\Neko.ini"; // 初始檔的位置
```

```
static int NewIni (void)
```

```
{
```

```
    int iRet=0;
```

```
    FHANDLE tFh;
```

```
    // 檔案 handler
```

```
    // 產生新初始檔
```

```
    tFh = FileOpen (gclniName, FILE_FLAG_CREAT \
```

```
        |FILE_FLAG_READWRITE|FILE_FLAG_TRUNCATE, \
```

```

        FILE_MODE_READ_WRITE);
    if (tFh >= 0) {
        FileWrite (tFh, (BYTE *)p, sizeof(p));           // 將 p 的內容全部寫出
        FileClose (tFh);
        iRet = 1;
    }
    else {
        iRet = 0;
    }
    return (iRet);
}

static int ReadIni (void)
{
    int iRet=0;
    FHANDLE tFh;

    tFh = FileOpen (gclniName, FILE_FLAG_READONLY, FILE_MODE_READ);
    if (tFh >= 0) {
        FileRead (tFh, (BYTE *)p, sizeof(p));           // 將 p 內容從初始檔讀回
        FileClose (tFh);
        iRet = 1;
    }
    else {
        iRet = 0;
    }
    return (iRet);
}

static int WriteIni (void)
{
    int iRet=0;
    FHANDLE tFh;

    // 開啟舊初始檔
    tFh = FileOpen (gclniName, FILE_FLAG_READWRITE, \
        FILE_MODE_READ_WRITE);

```

```

if (tFh >= 0) {
    FileWrite (tFh, (BYTE *)p, sizeof(p));           // 將 p 的內容全部寫出
    FileClose (tFh);
    iRet = 1;
}
else {
    iRet = 0;
}
return (iRet);
}

```

在 PenbexMain() 裡多加六行如下：當程式首次執行時做 NewIni() 動作，當程式刪除時將該初始檔一並刪除。

```

if (APP_RUN_COLD_RESET == argc) {
    NewIni ();
}
else if (APP_RUN_KILLED == argc) {
    FileDelete(gclniName);
}

```

在事件迴路裡，當 MT_SYS_CONT_BEGIN 別忘了執行 ReadIni(), 把狀態值讀回。MT_SYS_CONT_END 時執行 WriteIni(), 將現況寫到初始檔。寵物貓軟體就會更逼真，每次重新執行時，是從上一次位置開始，且動作一樣，寵物貓就像真的活在

PDA 裡。 

8. 如何抓到按鍵訊號?

在打擊遊戲裡，使用 PDA 按鍵會比用筆玩遊戲來的常見，所以想判斷是否有按鍵被按到，程式設計方法如下：在事件迴路裡，當收到 MT_KEY_DOWN 表示有某一個按鍵被按到，在該情況(case)下，再由訊息的 ID(pMsg->id)判斷是哪一個鍵。

```
case MT_KEY_DOWN:
    if ( pMsg->id&KEY_LEFT ) {
        do_something();           // 如果左鍵被按
    }
    else if ( pMsg->id&KEY_A ) {
        do_another();           // 如果 A 鍵被按
    }
    else if ( pMsg->id&KEY_RIGHT||KEY_UP ) {
        do_another();           // 如果右鍵上鍵同時被按
    }
    break;
```

其中按鍵定義如下：

```
KEY_POWER           // 電源鍵，其值為 0x0001
KEY_LEFT            // 左鍵，其值為 0x0002
KEY_RIGHT           // 右鍵，其值為 0x0004
KEY_UP              // 上鍵，其值為 0x0008
KEY_DOWN            // 下鍵，其值為 0x0010
KEY_SEL_A           // A 鍵，其值為 0x0020
KEY_SEL_B           // B 鍵，其值為 0x0040
```

由上述範例中最後一種情況得知，複合鍵是可行的，但在模擬器上由於所有操作皆用滑鼠控制，所以無法測試同時按兩個以上按鍵的效果。但事實上若把 PC 接上遊戲搖桿，它就可以 100% 操控模擬器的按鍵，做最真實的遊戲模擬。

9. 如何畫會動的球?

因為像乒乓球軟體, 球的位置也是遊戲狀態的變數之一, 所以也是被定義在一個結構體裡(如下範例) PopBall3Var, 而指標 pPopBall3Var 則記載球之前的位置 pPopBall3Var->preX, pPopBall3Var->preY。新畫球的位置為 pPopBall3Var->startx, pPopBall3Var->starty。其畫球的函式如下: (drawMovementOfBall())

```
static void drawMovementOfBall(void)
{
    BYTE    oldPattern;        // 暫存舊的筆刷圖案

    if ( pPopBall3Var->startx==pPopBall3Var->preX && \
        pPopBall3Var->starty==pPopBall3Var->preY )
    return;                    // 球位置沒變, 不畫
                                // 用 PATTERN_NULL 畫掉舊圖

    oldPattern=GmSetBrushPattern(GM_BRUSH_PATTERN_NULL);
    GmFillRect(pPopBall3Var->preX, pPopBall3Var->preY, \
        pPopBall3Var->BallWidth,pPopBall3Var->BallHeight, 2);
    GmSetBrushPattern(oldPattern);
                                // 畫新球

    GmDrawBmp(pPopBall3Var->startx,pPopBall3Var->starty, \
        (LBmp *)PopMBall);
                                // 更新球的新位置

    pPopBall3Var->preX=pPopBall3Var->startx;
    pPopBall3Var->preY=pPopBall3Var->starty;
}
```

球的位置會因其移動方向及速度, 判斷是否碰到邊框, 磚塊, 以及球拍而定; 最終位置也會因碰撞後的角度, 時間, 甚至碰撞的位置, 是否有切球等狀況而定。

10. 如何控制球拍？

綜合以上幾個章節，我們可以利用左右鍵來控制球拍左右移動，以下再以乒乓球程式片段來做說明。

```
case MT_KEY_DOWN:
case MT_KEY_REPEAT:
    if (pMsg->id & KEY_DOWN) { // 用來做暫停鍵
        pPopBall3Var->stop=0;
    }
    else if (pMsg->id & KEY_UP) { // 用來做開始鍵
        pPopBall3Var->stop=1;
    }
    else if (pMsg->id & KEY_LEFT) {
        if (pPopBall3Var->BoardLeftEdge>=pPopBall3Var->LeftBd+M_SPEED)
        { //檢查如果左移不超出螢幕
            pPopBall3Var->BoardLeftEdge-=M_SPEED; //左移 M_SPEED
            UpdateBoard(); //更新球拍畫面
        }
    }
    else if (pMsg->id & KEY_RIGHT) {
        if(pPopBall3Var->BoardLeftEdge<=pPopBall3Var-> \
            RightBd-pPopBall3Var->BoardWidth-M_SPEED)
        { //檢查如果左移不超出螢幕
            pPopBall3Var->BoardLeftEdge+=M_SPEED; //, 右移 M_SPEED
            UpdateBoard(); //更新球拍畫面
        }
    }
}
break;
```

至於如何更新球拍畫面, UpdateBoard()程式片段如下:

```
static void UpdateBoard(void)
{
    BYTE oldPattern; // 記住之前的筆刷顏色
                    // 設筆刷為清除
    oldPattern=GmSetBrushPattern(GM_BRUSH_PATTERN_NULL);
                    // 清除整個球拍區
    GmFillRect(pPopBall3Var->LeftBd,pPopBall3Var->DownBd-pPopBall3Var-> \
        BoardHeight, pPopBall3Var->RightBd-pPopBall3Var->LeftBd, \
        pPopBall3Var->BoardHeight,0);
                    // 設實心筆刷
    GmSetBrushPattern(GM_BRUSH_PATTERN_SOLID);
                    // 畫球拍
    GmFillRect(pPopBall3Var->BoardLeftEdge, \
        pPopBall3Var->DownBd-pPopBall3Var-> \
        BoardHeight,pPopBall3Var->BoardWidth,pPopBall3Var->BoardHeight,0);
                    // 設回之前筆刷圖案
    GmSetBrushPattern(oldPattern);
}
```

而 pPopBall3Var 的結構裡記載著, BoardHeight 球拍高度, BoardWidth 球拍寬度, BoardLeftEdge 球拍左緣 x 軸的值, DownBd 為下方球拍下緣的 y 軸位置, LeftBd 為球桌左緣, RightBd 為球桌右緣, 且定義球拍只能在球桌範圍內移動。

11. 如何產生聲音？

遊戲軟體常常需要發音，PDA 上發出聲音很簡單，第一個方法就是利用系統蜂鳴器的聲音，程式只要利用 `ToneSetFrequency` 函式先設好發音頻率，再叫用 `ToneOpen()` 函式，開始發出該頻率的聲音，直到程式叫用 `ToneClose()` 函式為止。其中 `ToneSetFrequency(DWORD Frequency)` 函式中只需 `Frequency` 一個參數值。

第二種方法是直接設定音階，利用 `ToneSetValue(WORD Degree,WORD Step)` 函式，設定音階(`Step`)和高低音的等級(`Degree`)，也是叫用 `ToneOpen()` 開始發出聲音，以及叫用 `ToneClose()` 停止。等級分二到五級，可用預設定義名稱如下：

<code>SECOND_DEGREE</code>	第二級
<code>THIRD_DEGREE</code>	第三級
<code>FOUR_DEGREE</code>	第四級
<code>FIVE_DEGREE</code>	第五級

音階也有預設定義值如下：

<code>DO_VOICE</code>	Do 音
<code>DO_UP</code>	升 Do 音
<code>RE_VOICE</code>	Re 音
<code>RE_UP</code>	升 Re 音
<code>MI_VOICE</code>	Mi 音
<code>FA_VOICE</code>	Fa 音
<code>FA_UP</code>	升 Fa 音
<code>SO_VOICE</code>	So 音
<code>SO_UP</code>	升 So 音
<code>LA_VOICE</code>	La 音
<code>LA_UP</code>	升 La 音
<code>SI_VOICE</code>	Si 音

第三種最簡單，叫用 `ToneBeep()` 函式 PDA 就會發出“嗶”一聲。

第四種較複雜，但可以發出美妙的音樂，利用 `TonePlayMusic (void *MusicFile,BOOL circulate,WORD second)` 或 `TonePlayMIDIFile(char *FileName,BOOL circulate,WORD`

second)。TonePlayMIDIFile 與 TonePlayMusic 最大不同在於 TonePlayMIDIFile 是用來播放 MIDI 檔，而且可以在模擬器中測試；但 TonePlayMusic 則是播放 WAV 檔，而且 WAV 檔的音樂只能在有喇叭的 PDA 機器上才能播放。最後一個參數 second 代表最長這行指令會執行多久，如果該首歌長度大於 second 大小，也只播放到 second 秒，但如果該首歌長度小於 second 大小，重不重播就看第二個參數 circulate 是否為 1。如果給 1，該指令會重播至 second 秒為止，如果給 0，就最多只播一次。這兩種方法都是在背景播放，程式會繼續往下執行，直到叫用 ToneStopMusic() 才會終止。

最後，SDK1.3 版還有一個函式 ToneSetVolume(BYTE Scale) 是用來設定 PDA 的大小聲，如果 PDA 硬體有此功能。也可以叫用 ToneGetVolume() 函式來獲取系統目前大小聲的設定值。聲音大小值 (Scale) 從 1 到 5，最大及最小聲有預先定義如下：

```
TONE_VOLUME_MIN    1
TONE_VOLUME_MAX    5
```

程式範例 Extone.c 一個簡易電子琴(如圖 6.11.1)就是採用第二種方法 ToneSetValue 的方式發音，先前利用表格(table)物件做出 DoReMi 的按鍵，當某一格被筆點選時，或筆在電子琴鍵上滑動時，程式會收到 MT_TABLE_COMMAND，及 MT_TABLE_FOCUS_IN 等事件訊息，處理發音方法如下程式片段：

```
case MT_TABLE_COMMAND:
    ToneClose();
    break;
case MT_TABLE_FOCUS_IN:
    Degree=pMsg->data.pos.x+1;           // Degree 為整數
    Voice=pMsg->data.pos.y+1;           // Voice 為整數
    ToneSetValue(Degree,Voice);
    ToneOpen();
    break;
```

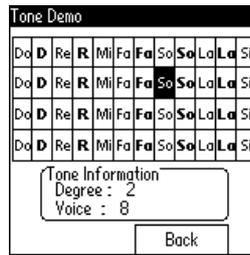


圖 6.11.1 簡易電子琴

至於 MP3 檔及錄音出來的聲音檔，操作方式更為複雜，請參考第四章第六節，有另一組 Snd 開頭的函式來完成之。

12. 如何利用紅外線對打?

其實最簡單的紅外線對打遊戲, 就是自訂傳輸速率及協定內容, 利用紅外線最簡單的方式傳輸, 通知對方 PDA 我方的輸入狀態, 比如說兩台 PDA 對玩井字遊戲, 雙方講好以 1 到 9 代表井字遊戲的九個格子, 如果我方 PDA 選到左上角的格子, 傳 1 給對方, 對方 PDA 就知道我方選擇左上角的格子, 代為顯示 o 或 x 在格子裡。

以下程式介紹如何利用紅外線與另一台 PDA 互傳英文字母, 其中程式片段如下:

```
char *   pBuffer;                               // 全區域變數

BOOL XmdmMainProcess (SysMsg *pMsg)
{
    BOOL      fResult= 0;
    UIOBJ     *pMedt;
    Char      pTemp[100];

    switch (pMsg->type)
    {
    case MT_SYS_CONT_BEGIN:
                                                // 利用一多行編輯器顯示狀況
        ContAddObj (pMedt = medNew(1000, 0, 0, 160, 5, \
            EM_STYLE_NO_BORDER|EM_STYLE_NO_SHOWENTER, \
            GM_FONT_MIDDLE) );
        ContAddObj (SysEngKB (1002));          // 加入一個軟鍵盤
        ContRedraw();
                                                // 先設定 UART 為紅外線雙向模式
        if (UrtSet (UART_MODE_IRCOMM_MDM,UART_WAY_ON_BOTH, \
            9600,8,1))
        {
                                                // 須設暫存區 pBuffer, 並啟動紅外
            if (!UrtRcvModeBegin (pBuffer=MmNewP (1024), 1024))
                medAppendStr (pMedt, "UrtRcvModeBegin fail.");
            else
                medAppendStr (pMedt, "IrComm connected...\n");
        }
    }
```

```

else
    medAppendStr (pMedt, "IrComm initial fail.");
fResult = 1;
break;

case MT_CHAR:
    // 每當軟鍵盤被按某英文字母
    pTemp[0] = (char)pMsg->id;
    if (1 != UrtlmmSendData (pTemp, 1, 1000))
        // 馬上傳送 1 個 byte, 且設定 time out
        // 1000ms 內無法傳出, 傳送失敗
    {
        pMedt = ContGetObj (1000);
        Sprintf (&pTemp[1], "!! Send char (0x%02x)fail. !!", pTemp[0]);
        medAppendStr (pMedt, &pTemp[1]);
    }
    fResult = 1;
    break;

case MT_UART_DATA_COME:
    // 若紅外線有接收到東西
    pMedt = ContGetObj (1000);
    Memcpy (pTemp, pMsg->data.p, pMsg->id); // pMsg->data.p 是東西內容
    pTemp[pMsg->id] = '\0'; // pMsg->id 是內容長度
    medAppendStr (pMedt, pTemp); // 將收到的內容顯示出來
    fResult = 1;
    break;

case MT_SYS_CONT_END:
    MmDelP (pBuffer); // Free 掉暫存區記憶體空間
    UrtRcvModeEnd (); // 關閉紅外線
    fResult = 1;
    break;
}
return fResult;
}

```

其他紅外線相關問題，也請參考下冊第十一章“PDA 紅外線能做甚麼？”

13. 練習題

--- 參考資料 ---

Penbex中文參考手冊

Penbex OS SDK 參考手冊 1.0 (中文版)

Penbex SDK 安裝手冊 1.0 (中文版)

Penbex英文參考手冊

Penbex SDK Reference Manual 1.0 (英文版)

Penbex SDK Installation Guide 1.0 (英文版)

C語言相關書籍

The Standard C Library, P.J. Plauger, PRENTICE HALL 1992

Introduction to Microsoft Visual C++ 6.0 Standard Edition, Wrox Press Ltd 1998

Expert C Programming, Deep C Secrets, Peter Van Der Linden, PTR PH, SunSoft Press

其他PDA程式設計參考書籍

Advanced Palm Programming, Steve Mann, Ray Rischpater, Wiley 2001

--- 下冊預告 ---

--- 應用篇 ---

七、 我會做 PDA 電子書

1. 前言
2. 如何寫 TEXT 閱覽器?
3. 大小字切換很簡單
4. 如何下載電子書資料?
5. 我也想看 Palm 的電子書, 行嗎?
6. 如何製作 AportisDoc 格式的電子書檔?
7. 電子書熱門網站
8. 練習題

八、 PDA 接上 GPS 衛星定位系統

1. 前言
2. 有哪些種 GPS 能接 PDA?
3. GPS 運作原理
4. 電子地圖如何對應?
5. RS232 接收 GPS 訊號程式設計
6. 其他 RS232 週邊有哪些?
7. XMODEM 傳輸原理
8. PDA 能控制天文望遠鏡尋找星球嗎?
9. 練習題

九、 我的 PDA 程式如何上網?

1. 前言
2. 撥接上網程式範例
3. ISP 及 email 帳號設定

4. 手機也能通!
5. TCP/IP 程式設計原理
6. UDP 程式範例
7. 如何開發 PDA Chat 聊天軟體
8. 我也會寫 PDA 電子郵件接收軟體
9. 什麼是 non-blocking 網路 APIs?
10. 練習題

十、 PDA 能用資料庫嗎?

1. 前言
2. Database 程式設計原理
3. dBaseIII 資料格式
4. PDA 資料庫程式範例
5. 如何使用索引檔作搜尋?
6. Field 物件的功能?
7. 如何將 Outlook 電話簿資料轉到 PDA?
8. 什麼是 Data Manager? 它的好處是什麼?
9. 練習題

十一、 PDA 紅外線能做甚麼?

1. 前言
2. 與 PC 無線傳輸
3. 用手機無線上網
4. 和其他 PDA 對傳資料
5. 怎麼寫紅外線傳輸程式?
6. OBEX 是什麼?
7. 藍芽與紅外線的比較
8. 練習題

--- 爪哇篇 ---

十二、 PDA 也能跑 Java 程式

1. 前言
2. J2ME 與 CLDC 介紹
3. PDA 安裝 Penbex KVM
4. PC 安裝 JDK1.3 開發環境
5. 編譯 Java2 Hello World 程式
6. 除錯方法
7. Kjava 標準使用者介面介紹
8. 練習題