Lab-LINK$_{TM}$ for Windows 中文圖控系統

第三部份

Smart Script 使用手冊

超鼎科技股份有限公司

目 錄

第一章　模組概述

SmartScript *for Windows* 是 **Lab-Link** for Windows 中文圖控系統所提供的一套 Script 語言，它具有一般程式語言的強大功能，但由於其語法簡單，同時跟 **Lab-LINK** for Windows 中文圖控系統結合緊密，在圖控系統的應用須涉及較為複雜的邏輯時，它讓系統規畫者可以不需假藉任何其它的程式語言編譯程式或開發工具，耗費最小的成本即可迅速完成控制邏輯的撰寫。

SmartScript 具備諸如資料型別、變數、迴圈、條件判斷、檔案存取與 I/O 通訊等完整的程式語言功能，並提供眾多的函式可供呼叫以滿足不同的程式需求。同時，***SmartScript*** 也提供一簡單的編輯環境，並具有語法編譯與執行偵錯等功能。由於採用類似 BASIC 的語法，並可直接存取圖控系統中的 TAG 資料，粗具程式設計概念的使用者即可迅速上手，撰寫出所需的控制邏輯。

本手冊將就 ***SmartScript*** 模組本身及其程式編輯器的使用加以說明，有關 ***SmartScript*** 的完整語法及所提供的眾多函數的說明，請參閱「***SmartScript*** 控制語言模組參考手冊」。

系統特色

- 完整的程式語言功能。
- 可直接存取圖控系統的 TAG 資料。
- 提供條件分支、迴圈以及副程式呼叫等程式流程控制功能。
- 提供由 TAG 值變化觸發的事件處理功能。
- 提供檔案存取能力，可自行撰寫資料處理程式。
- 提供 I/O 通訊能力，可自行撰寫 I/O 通訊處理程式。
- 提供多種數學函式，滿足複雜數學運算的需求。
- 提供多種字串函式，滿足字串型資料處理的需求。
- 提供多種時間函式，滿足定時動作的需求。

使用模式

SmartScript 模組提供兩種使用模式：編輯模式與執行模式。在編輯模式下，會啟動 ***SmartScript*** 程式編輯器，使用者可以載入程式檔加以編輯，然後加以編譯及執行，程式編輯器並提供各種程式偵錯功能。此模式多用於專案規畫階段，來撰寫發展所需的程式。***SmartScript*** 的執行模式則會直接載入指定的程式檔並立即執行。

SmartScript 模組的執行檔為 CONTROL.EXE，其命令列參數決定其使用模式，詳下節所述。

命令列

以下的命令列參數指令可用在 CONTROL.EXE 指令之後：

CONTROL.EXE [程式檔名 [/R]]

參數	說明	使用模式
無	啟動 ***SmartScript*** 程式編輯器，開啟一未命名之新程式檔	編輯模式
程式檔名	啟動 ***SmartScript*** 程式編輯器並載入指定的程式檔。	編輯模式
程式檔名 /R	執行 ***SmartScript*** 以載入並執行指定檔案。	執行模式

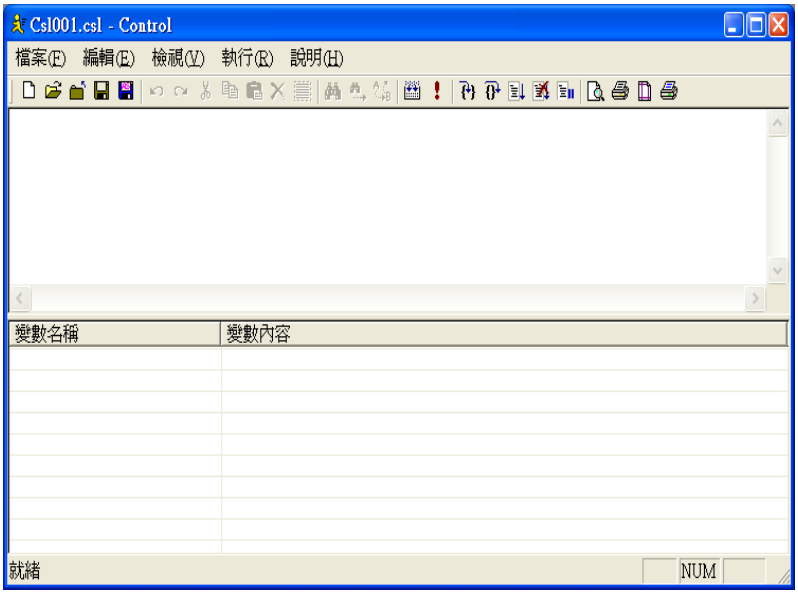
程式檔名的副檔名應為.CSL，其檔案格式應為一遵循 ***SmartScript*** 語法的標準文字檔。

第二章　程式編輯器

程式編輯器是 ***SmartScript*** 模組所提供的一個簡單易用的 ***SmartScript*** 程式開發環境，它提供了基本的程式碼編輯功能，並讓使用者編譯所撰寫的程式碼，以檢查是否有語法(syntex)上的錯誤，此外也提供了配合程式偵錯所需的執行控制，以方便使用者逐步檢查程式執行中所發生的錯誤；偵錯視窗中更可顯示使用的所有變數內容，協助使用者驗證程式的正確性。

操作畫面

以編輯模式執行 ***SmartScript*** 模組後，即出現以下畫面：



若啟動 ***SmartScript*** 模組時已指定程式檔名，則視窗標題中將出現程式檔檔名，否則將顯示(未命名)。視窗上方為下拉式功能表，提供檔案、編輯、搜尋、檢視、執行等功能表供使用者選擇，視窗中則是程式碼的編輯區。以下各節將說明各功能表所提供的功能以及使用方法。

檔案功能表

檔案功能表提供以下的功能：

開新檔案

開啟一個新的空白檔案。

開啟舊檔

開啟一個已存在的舊檔。若先前編輯中的檔案尚未存檔，系統會先詢問是否要將編輯中的檔案存檔。接著將出現「開啟舊檔」對話盒，使用者可選擇要開啟的檔案，或直接輸入檔案路徑與名稱。

儲存檔案

將編輯中的檔案以原來的檔名回存。若編輯的是一個未命名的新檔案，則會出現「另存新檔」的對話盒，讓使用者輸入檔案名稱。

另存新檔

將編輯中的檔案存成另一個檔名。畫面中會出現「另存新檔」的對話盒，讓使用者輸入檔案名稱。

列印

列印編輯中的檔案內容。畫面中會出現「列印」對話盒，使用者可指定指定印表機、設定列印範圍及份數。

直接列印

直接列印程序檔內容。會依照列印設定立即將控制程序檔的內容列印出來。

預覽列印

會顯示列印後的預覽畫面及所需頁數。畫面可拉近拉遠。

列印設定

列印編輯中的檔案內容。畫面中會出現「設定印表機」對話盒，使用者可指定列印紙張的來源及大小、列印方向等。

結束

關閉 ***SmartScript*** 模組程式編輯器。若原先 ***SmartScript*** 模組的程式正在執行，則會先結束正執行中的程式。若原先正編輯或執行行中的程式已被修改，而且尚未存檔，系統會先詢問使用者是否要將修改過的程式回存。

編輯功能表

編輯功能表提供以下的功能：

復原

還原上一個編輯動作。

重做

還原下一個編輯動作。

剪下

將被選取到的文字剪下來，暫存於剪貼簿中。

複製

複製被選取到的文字，暫存於剪貼簿中。

貼上

將前一次剪下或複製的內容(即剪貼簿中的內容)，貼到目前編輯游標所在的位置。

刪除

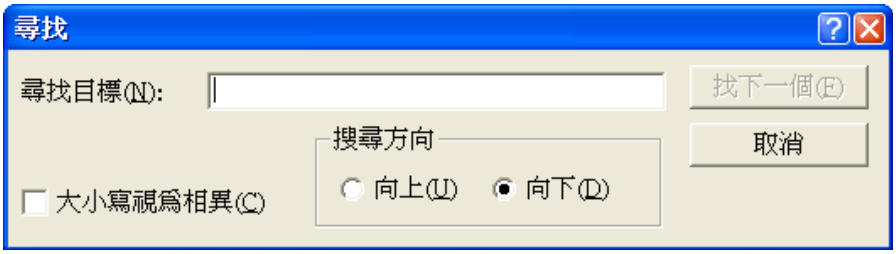
將被選取到的文字刪除。

選取全部

選取程式檔中的所有內容。

尋找

游標移到下一行含有尋找目標字串的文字。畫面上會出現「尋找」對話盒，使用者應在「尋找目標」欄位輸入要尋找的字串，接著可進行以下操作：



找下一個按鈕

系統會將游標移到下一行含有目標字串的文字，並將目標字串選取。使用者可直接編輯文字內容，或再按找下一個按鈕，繼續尋找下一個目標字串。若游標已到達檔案底端，會出現「已完成整份文件的搜尋，找不到所搜尋的項目」的訊息，若須由檔案開頭重新尋找，應將編輯游標移至檔案開頭後，再重新做搜尋的操作。

取消按鈕

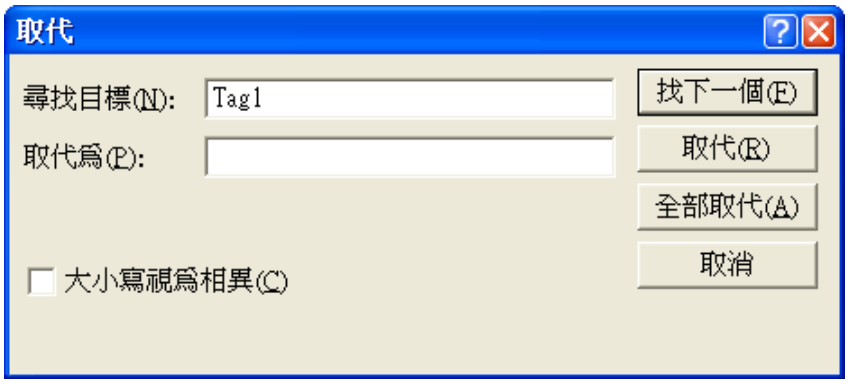
關閉「尋找」對話盒。

找下一個

前一個目標字串做為搜尋目標，將游標移到下一行含有尋找目標字串的文字。若游標已到達檔案底端，會出現「已完成整份文件的搜尋，找不到所搜尋的項目」的訊息，若須由檔案開頭重新尋找，應將編輯游標移至檔案開頭後，再重新做搜尋的操作。

取代

將目標字串取代成為指定字串。畫面上會出現「取代」對話盒，使用者應分別在「尋找目標」及「取代為」欄位分別輸入要尋找的目標字串，以及要取而代之的新字串，接著可做以下之操作：



找下一個按鈕

系統會將游標移到下一行含有目標字串的文字，並將目標字串選取，但不會取代該字串。使用者可直接編輯文字內容，或再按「找下一個」按鈕，繼續尋找下一個目標字串。若游標已到達檔案底端，會出現「已完成整份文件的搜尋，找不到所搜尋的項目」的訊息，若須由檔案開頭重新尋找，應將編輯游標移至檔案開頭後，再重新做搜尋的操作。

取代按鈕

系統會以取代字串來取代下一個目標字串。使用者再按取代按鈕，繼續取代下一個目標字串。若游標已到達檔案底端，會出現「已完成整份文件的搜尋，找不到所搜尋的項目」的訊息，若須由檔案開頭重新取代，應將編輯游標移至檔案開頭後，再重新做取代的操作。

全部取代按鈕

若須取代檔案中的所有目標字串，應先將編輯游標移至檔案開頭後，再按全部取代按鈕開始做全部取代的操作。

取消按鈕

關閉「取代」對話盒。

檢視功能表

檢視功能表提供以下的功能：

工具列– 可決定要不要顯示圖形工具列

狀態列– 可決定要不要顯示狀態列

執行功能表

執行功能表提供以下的功能：

重新編譯– 對編輯視窗中的程式檔內容進行語法的檢查。由於 ***SmartScript*** 事實上是一種解譯語言，程式碼的編譯並非必要。因此，此功能的主要目的係在執行程式之前，先就程式碼的語法(syntax)加以檢查，若有語法上的錯誤會出現訊息視窗，顯示錯誤碼及訊息。有關於錯誤代碼的說明，請參閱 ***SmartScrip*** 控制語言模組參考手冊附錄 D 中的說明。

重新執行– 對編輯視窗中的程式檔內容加以解譯並執行。程式會持續執行直到程式碼終止，程式發生錯誤或使用者加以中斷為止。當程式的執行發生錯誤時，會出現訊息視窗，顯示錯誤碼及訊息。使用者也可以利用本功能表中的「中斷執行」或「停止執行」命令來暫停或終止程式的執行。

單步執行– 以每次僅執行一個敘述句的方式做逐步執行。每次選擇本功能表中的 STEP 命令，會解譯並執行一個敘述句，執行完該敘述句後即暫停，等待使用者的進一步操作。使用者可以再選擇一次「重新執行/單步執行」來執行下一個敘述句。單步執行操作對副程式也一視同仁，當所執行的敘述句為副程式呼叫時，會進入副程式，副程式中的每一個敘述均需一個單步執行操作來逐步執行。使用者可以利用功能鍵來簡化操作動作，每按一次「F8」功能鍵即可執行一次單步執行動作。

翻越執行– 與單步執行類似，以每次僅執行一個敘述句的方式做逐步執行。但主要差別在於對副程式呼叫的處理。當執行到副程式呼叫的敘述時，翻越執行會執行完副程式中的所有敘述，直到 RETURN 敘述返回主程式才暫停。

全速執行– 自暫停處開始，持續執行程式直到程式碼終止、程式發生錯誤或使用者加以中斷為止。

多用於逐步執行至一個階段，希望接下來讓程式自動執行到完成。

中斷執行– 暫停目前程式的執行。使用者在暫停後可以利用單步執行或翻越執行來繼續逐步執行，或選擇全速執行來連續執行。

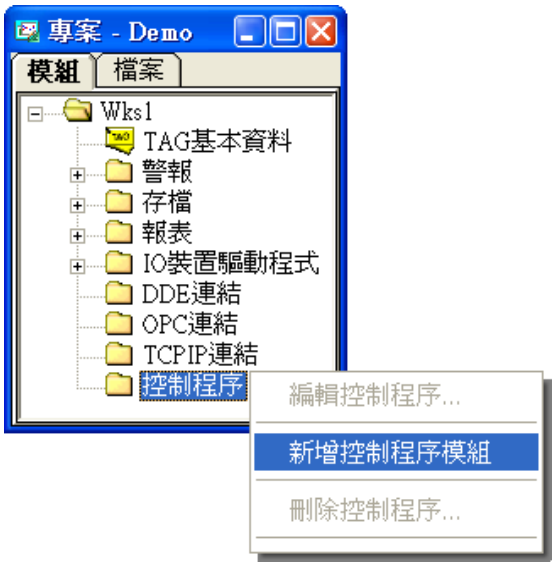
停止執行– 終止目前程式的執行，若程式有開啟檔案，停止執行會關閉所有被開啟的檔案。

第三章　模組應用

利用 ***SmartScript*** 控制語言模組所發展出來的程式碼，是可以單獨利用本模組來執行的。但在實際應用時，由於它與 **Lab-LINK** for Windows 中文圖控系統的緊密結合，通常是用來滿足圖控系統規畫時的特殊需要，例如複雜的控制邏輯與數學運算，自定格式的檔案儲存與讀取，甚至撰寫通訊程式等。本章中將針對常見的應用模式，以簡單的範例加以說明。

圖控系統啟動時執行

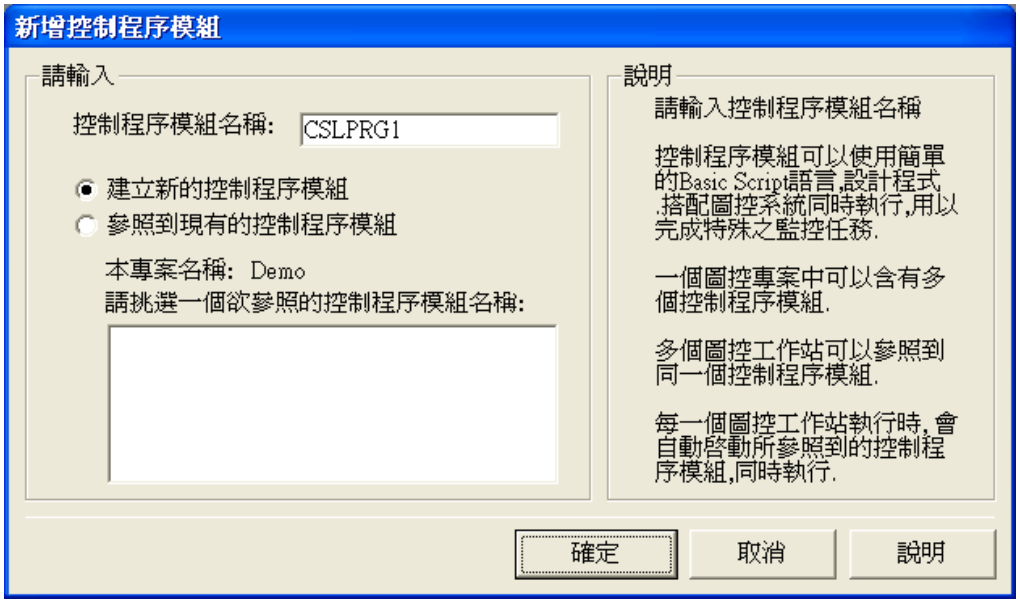
SmartScript 最簡單的應用方式是隨同圖控系統而執行，只要在 **Lab-LINK** for Windows 專案管理模組的專案規畫中加入一個「控制程序」，當圖控系統執行時，該程序名稱所對應的 ***SmartScript*** 程式檔即會被載入指定並執行。程式檔中的敘述執行完畢後，***SmartScript*** 控制語言模組即自動結束。



在圖控專案管理系統中，選擇工作站下的「控制程序」並輕點滑鼠右鍵，由快顯功能表中選擇「新增控制程序模組」，系統會要求使用者做以下的選擇：

- ❒ 建立新的控制程序模組 : 選此項目時應輸入控制程序模組名稱 , 按「確定」後系統會在專案目錄下的控制程序子目錄中以該名稱建立一副檔名為「CSL」的程式檔，並啟動 ***SmartScript*** 編輯程式讓使用者編輯。

- ❒ 參照到現有的控制程序模組： 若該專案的其他工作站已建立某些控制程序模組，可選擇此選項，表該工作站將執行使用與其他工作站相同的控制程序，並由下方選擇一已建立的模組名稱，按「確定」後系統會啟動 ***SmartScript*** 編輯程式並載入該程式檔讓使用者編輯。



建立控制程序模組並編譯專案後，當執行專案時該工作站所關連的控制程序即會隨之啟動。以下的例子會在圖控系統啟始時，將一行圖控系統已啟動的訊息寫入一記錄檔中，完成後即關閉該記錄檔並結束執行。

■ **程式碼**

行號	程式碼
1	FileName$="LABLINK.LOG"
2	OPEN 1, FileName$, "W"
3	IF (!FCHECK(1))
4	CREATE 1, FileName$
5	ENDIF
6	// Write data to the data file
7	SEEK 1, 0, "E"
8	FPRINT 1, YEAR(),"/",MONTH(),"/",DAY()," "
9	FPRINT 1, HOUR(),":", MINUTE(),":",SECOND()," "
10	FPRINT 1, "LABLINK STARTED\r\n"
11	CLOSE 1
12	END

註：上列行號部份的目的只是便於本節中的說明，並不屬於程式碼的一部份。

■ TAG **與變數說明**

名稱	種類	說明
FileName$	字串變數	用以儲存檔案名稱的變數

■ **程式說明**

第 1 行： 指定記錄檔的名稱。***SmartScript*** 控制語言模組的工作目錄即 **Lab-LINK** 的系統目錄(通常為 C:\LABLINK4\SYSTEM4)，若檔案位置位於其他目錄，應包含完整的絕對或相對路徑，例如 FileName$="..\\CSL\LABLINK.LOG"指定檔案路徑為 ..\CSL\LABLINK.LOG (相當於絕對路徑 C:\LABLINK4\CSL\LABLINK.LOG)。注意路徑中的「\\」, 因為「\」通常用於標示特殊的控制字元 , 所以字串須用到「\」時應寫為「\\」。

第 2 行： 以寫入模式開啟檔案，並將此檔案關連到檔案編號 1，此後的讀寫動作均以此檔案編號為依據，直到此檔案被關閉為止。

第 3-5 行： 判斷指定檔案是否成功開啟 , 若不成功則視為檔案不存在 ; 成功則以 CREATE 敘述產生以指定名稱命名的檔案，並將之開啟準備寫入資料。

第 6 行： 「//」之後的文字均被視為註解文字。

第 7 行： 將讀寫指標移至檔案的結尾，換言之即準備由檔案的末端加入新的資料。

第 8-10 行： 寫入系統的日期及時間，以及「LABLINK STARTED」的訊息。其中 YEAR()、MONTH()、DAY()、HOUR()、MINUTE()及 SECOND()分別傳回系統日期時間的年、月、日、時、分及秒的部份，「\r」及「\n」則分別代表「Carriage Return」與「Line Feed」兩個控制字元 , 用以達到換行的目的。此三行敘述也可寫在同一行 FPRINT 敘述中。

第 11 行： 關閉檔案。檔案關閉之後，檔案編號 1 即可被其他檔案的存取指令使用。

第 12 行： 程式結束，***SmartScript*** 控制語言模組也隨之結束。此敘述不可省略。

由圖控物件啟動

SmartScript 也可以配合 ***SmartPanel*** 人機介面模組來使用，利用「程式執行器」物件在適當狀況下(例如有使用者按下某一個按鈕)啟動一 ***SmartScript*** 程式，以完成特定的動作。以下的例子是配合圖控畫面的「LOGIN」按鈕，當使用者按下此按鈕並輸入密碼後，即由 ***SmartScript*** 程式記錄使用者的身份登錄至一文字檔。

■ ***SmartPanel*** 人機介面模組中的設定

物件	參數設定	TAG 設定
按鈕	物件抬頭：LOGIN 物件風格：無段式按鈕，強制密碼 說明：按下此按鈕會將名稱為 **LOGIN** 的 TAG 值設為 1 選無段式按鈕會在使用者放開按鈕後即將 TAG 值復歸為 0 強制密碼會強迫使用者輸入密碼以便確認身份	**LOGIN**
程式執行器	指令行：**CONTROL.EXE ..\PROJECT\PROJ1\CSL\LOGIN.CSL /R** 說明：當 **LOGIN** 值變為 1 即執行指定的 ***SmartScript*** 程式	**LOGIN**

註：控制程序檔通常應位於專案目錄下的 CSL 子目錄中，本例之專案名稱為 PROJ1，控制程序名稱則為「LOGIN」。

■ **程式碼**

行號	程式碼
1	FileName$="LABLINK.LOG"
2	OPEN 1, FileName$, "W"
3	IF (!FCHECK(1))
4	CREATE 1, FileName$
5	ENDIF
6	// Write data to the data file
7	SEEK 1, 0, "E"
8	FPRINT 1, YEAR(),"/",MONTH(),"/",DAY()," "
9	FPRINT 1, HOUR(),":", MINUTE(),":",SECOND()," "
10	FPRINT 1, {$USER.$}, " 登入\r\n"
11	CLOSE 1
12	END

註：上列行號部份的目的只是便於本節中的說明，並不屬於程式碼的一部份。

■ TAG **與變數說明**

名稱	種類	說明
LOGIN	TAG	用以使用者登入程序的 TAG
$USER	系統 TAG	記錄使用者資訊的系統 TAG
FileName$	字串變數	用以儲存檔案名稱的變數

■ **程式說明**

第 1 行： 指定記錄檔的名稱。。

第 2 行： 以寫入模式開啟檔案，並將此檔案關連到檔案編號 1，此後的讀寫動作均以此檔案編號為依據，直到此檔案被關閉為止。

第 3-5 行： 判斷指定檔案是否成功開啟，若不成功則視為檔案不存在，以 CREATE 敘述產生以指定名稱命名的檔案，並將之開啟準備寫入資料。

第 6 行： 「//」之後的文字均被視為註解文字。

第 7 行： 將讀寫指標移至檔案的結尾，換言之即準備由檔案的末端加入新的資料。

第 8-10 行： 寫入系統的日期及時間，以及使用者登入的訊息。其中「**$USER**」為一系統 TAG，「**$USER**.$」代標該 TAG 的訊息欄位，用以記載當時的使用者名稱。注意 TAG 名稱兩邊加上一對大括號，代表這是一個 TAG 變數。

第 11 行： 關閉檔案。檔案關閉之後，檔案編號 1 即可被其他檔案的存取指令使用。

第 12 行： 程式結束，***SmartScript*** 控制語言模組也隨之結束。

迴圈的執行

在前兩個例子中，*SmartScript* 程式在執行完畢後即結束，但在某些應用場合，可能必須讓 *SmartScript* 程式持續執行，以不斷地進行程式的運算。在下面的範例中，當使用者按下「開始積分」按鈕後，即由 *SmartScript* 程式開始執行一個迴圈，對 Tag1 進行積分，直到使用者按下「停止積分」按鈕後，迴圈中止而程式也隨之停止。

■ *SmartPanel* 人機介面模組中的設定

物件	參數設定	TAG 設定	說明
按鈕	物件抬頭：開始積分 物件風格：設定式按鈕	**StartInt**	按下此按鈕會將名稱為 **StartInt** 的 TAG 值設為 1
按鈕	物件抬頭：停止積分 物件風格：清除式按鈕	**StartInt**	按下此按鈕會將名稱為 **StartInt** 的 TAG 值設為 0
程式執行器	指令行： **CONTROL.EXE ..\PROJECT\PROJ1\CSL\INTEGRAL.CSL /R**	**StartInt**	當 **StartInt** 值變為 1 即執行指定的 *SmartScript* 程式
靜態文字顯示器	物件抬頭：瞬間值		顯示「瞬間值」文字

物件	參數設定	TAG 設定	說明
靜態文字顯示器	物件抬頭：積分值		顯示「積分值」文字
編輯器		**Tag1**	讓使用者輸入瞬間值 **Tag1**
文字錶頭		**Int_Tag1**	顯示積分值 **Int_Tag1**

註：控制程序檔通常應位於專案目錄下的 CSL 子目錄中，本例之專案名稱為 PROJ1，控制程序名稱則為「INTEGRAL」。

- **程式碼**

行號	程式碼
1	PrevTime%={$TIME}
2	PrevValue={Tag1}
3	{Int_Tag1}=0
4	WHILE({StartInt})
5	{Int_Tag1}={Int_Tag1}+({Tag1}+PrevValue)*({$TIME}-PrevTime%)/2
6	PrevTime%={$TIME}
7	PrevValue={Tag1}
8	LOOP
9	END

註：上列行號部份的目的只是便於本節中的說明，並不屬於程式碼的一部份。

- TAG **與變數說明**

名稱	種類	說明

StartInt	TAG	用以控制開始與停止積分運算的 TAG
Tag1	TAG	瞬間值
Int_Tag1	TAG	積分值
$TIME	系統 TAG	其值每秒會自動加 1 的系統 TAG，可用作一秒計時器
PrevTime%	整數變數	用以儲存前一運算時刻的**$TIME** 值的變數
PrevValue	實數變數	用以儲存前一運算時刻的 **Tag1** 值的變數

■ **程式說明**

第 1 行： 設定 PrevTime%初值。

第 2 行： 設定 PrevValue 初值。

第 3 行： 將積分值 **Int_Tag1** 之值歸零。

第 4 行： 以 **StartInt**=1 為進入條件，開始一 While 迴圈。若 **StartInt**=0，則迴圈停止，程式也將結束。**StartInt** 亦在圖控面板中用作開始執行此段程式之 TAG。

第 5 行： 以梯形法計算 **Tag1** 的積分值。其中 PrevValue 為前一運算時刻的 Tag1 值，{**$TIME**}-PrevTime%則為前一運算時刻迄此次運算所經過之時間。

第 6 行： 更新 PrevTime%值。

第 7 行： 更新 PrevValue 值。

第 8 行： 迴圈末端。

第 9 行： 程式結束，***SmartScript*** 控制語言模組也隨之結束。

使用者按下「開始積分」按鈕，將 **StartInt** 設為 1 後，即開始執行本程式。程式開始後，會先將積分值歸零，然後持續計算 **Tag1** 的積分值 **Int_Tag1**，直到使用者按下「停止積分」按鈕，將 **StartInt** 設為 0 後，結束迴圈後程式停止。使用者若再按下「開始積分」按鈕，則重新執行程式，並將積分值歸零，再開始計算 **Tag1** 的積分值 **Int_Tag1**。

若須讓程式在圖控專案執行期間持續地執行，可將 While 迴圈的條件設為 1，如此即成為一無窮迴圈。無窮迴圈在圖控專案執行期間，均將持續執行。

事件觸發的執行方式

SmartScript 除了提供迴圈的順序執行方式外，也提供了事件的執行方式。在某些應用場合時，事件觸發的執行方式，會較迴圈的執行方式有效率。以前例中的積分程式而言，在每一迴圈運算中均會重覆計算積分值，即使時間秒數(**$TIME**)或 **Tag1** 值並無變化亦然。在下面的範例中，將前例中的迴圈計算改寫為事件觸發的形式，其畫面設定、操作方式、TAG 與變數的使用均與前例相同，將不再贅述。

■ **程式碼**

行號	程式碼
1	PrevTime%={$TIME}
2	PrevValue={Tag1}
3	{Int_Tag1}=0
4	WHILE({StartInt})
5	LOOP
6	END

行號	程式碼
7	{$TIME}:
8	{Int_Tag1}={Int_Tag1}+({Tag1}+PrevValue)*({$TIME}-PrevTime%)/2
9	PrevTime%={$TIME}
10	PrevValue={Tag1}
11	RETURN

註：上列行號部份的目的只是便於本節中的說明，並不屬於程式碼的一部份。

■ **程式說明**

第 1 行： 設定 PrevTime%初值。

第 2 行： 設定 PrevValue 初值。

第 3 行： 將積分值 **Int_Tag1** 之值歸零。

第 4 行： 以 **StartInt**=1 為進入條件，開始一 While 迴圈。若 **StartInt**=0，則迴圈停止，程式也將結束。**StartInt** 亦在圖控面板中用作開始執行此段程式之 TAG。

第 5 行： 迴圈末端。

第 6 行： 主程式結束，***SmartScript*** 控制語言模組也隨之結束。

第 7 行： 以**$TIME** 觸發的副程式之啟始標示(Label)。每當**$TIME** 值改變，即中斷主程式，而執行此標示以下的副程式。

第 8 行： 以梯形法計算 Tag1 的積分值。其中 PrevValue 為前一運算時刻的 Tag1 值，{**$TIME**}-PrevTime%則為前一運算時刻迄此次運算所經過之時間。

第 9 行： 更新 PrevTime%值。

第 10 行： 更新 PrevValue 值。

第 11 行： 副程式結束，返回原主程式中斷處繼續執行。

此範例同樣會在使用者按下「開始積分」按鈕，將 **StartInt** 設為 1 後開始執行。但程式開始並完成一

些初執的設定後，即開始執行一空迴圈，迴圈本身並不執行任何運算。只要**$TIME** 有任何變化(每秒鐘一次)，即觸發以**$TIME** 為 Label 的副程式的執行來計算積分值。由於本範例僅會每秒鐘執行一次積分運算，因此不會時常佔用 CPU 時間來處理積分運算，也可減低對 **Lab-LINK** 其他模組的影響。

實際應用時，可進一步藉由事件觸發 TAG 的選擇，更進一步控制事件副程式的執行。例如可以利用圖控面板中的計時器物件來產生不同時脈的觸發信號，以控制執行每次積分運算的時間間隔。

Tag 事件副程式應用時應注意的問題

Tag 事件副程式是 ***SmartScript*** 中相當好用的功能，但由於其事件觸發的特性，使用時程式設計者必須妥善的加以控制。一般而言， Tag 事件副程式係指一段以 Tag 名稱做**行標**的指令行開始，以 **RETURN** 敘述結尾的 SmartScript 程式碼。下面的程式碼即為一典型的範例：

```
…                    // 其他程式碼
{Tag1}:
…                    // Tag1 事件副程式的程式碼
RETURN
```

如下例所示，Tag 事件副程式可以有多個 Tag 行標，任何一個行標 Tag 的資料有變化，均將引發此副程式的執行。

```
…                    // 其他程式碼
{Tag1}:
{Tag2}:
{Tag3}:
…                    // Tag1, Tag2, Tag3 事件副程式的程式碼
RETURN
```

當做為副程式**行標**的 Tag 資料有變化時，***SmartScript*** 控制程序模組將立刻跳到此 Tag 行標，執行 Tag 行標以下的指令行，直到到達 **RETURN** 敘述後，程式流程才回到 Tag 事件發生前的指令行繼續往下執行。Tag 事件副程式係由行標 Tag 的資料變化觸發，這裡所指的 Tag 資料變化包括 Tag 的數值、訊息、日期、時間與狀態。

如前所述，Tag 事件副程式係由行標 Tag 的資料變化觸發，但若在某一 Tag 事件尚未完成之前，又有新的 Tag 事件觸發，往往會造成無法預期的結果。**TRAPOFF** 與 **TRAPON** 兩指令即用協助程式設計者掌控事件觸發的行為，前者將設定一 **TRAPOFF** 旗號來暫停 Tag 事件處理功能，後者則會清除 **TRAPOFF** 旗號以重新啟用 Tag 事件處理功能。當 Tag 事件處理功能被暫停時，控制程序模組將以一**事件佇列**來儲存暫停期間所觸發的 Tag 事件，以便在 Tag 事件處理功能在重新啟用後，可以執行這些被暫停的事件副程式。

為協助程式設計者瞭解如何利用 **TRAPOFF** 與 **TRAPON** 兩指令來配合 Tag 事件副程式的撰寫，以下

將說明 Tag **事件副程式的流程。**

- 在 ***SmartScript*** 執行期間，若有任何 Tag 事件觸發，且該 Tag 的事件處理副程式並不存在**事件佇列**中，則將該 Tag 事件加入**事件佇列**。
- 任一 ***SmartScript*** 指令執行後，將做以下檢查與處理：
  - 檢查 **TRAPOFF** 旗號，如果 **TRAPOFF** 旗號未被設定，檢查**事件佇列**中是否有未處理的 Tag 事件，如果有，則由**事件佇列**中取出一個事件，並將下一個將執行的指令指向該事件副程式。
    - 如果剛剛的指令是 **TRAPON** 敘述，則清除 **TRAPOFF** 旗號。
    - 繼續執行下一個指令。
- 基於前述的說明，如果想確保 Tag 事件處理副程式執行期間，不被新的 Tag 事件中斷，可如下例所示地加入 **TRAPOFF** 與 **TRAPON** 兩指令：

```
…                    // 其他程式碼
{Tag1}:
TRAPOFF
…                    // Tag1 事件副程式的程式碼
TRAPON
RETURN
```

  - 上例中，在 Tag1 事件觸發後，控制程序模組執行的第一個指令就是 **TRAPOFF** (行標本身不算是一個指令)，此指令會立刻設定 **TRAPOFF** 旗號；此時即使新的 Tag 事件觸發，該事件僅會被加入**事件佇列**而不會立刻執行。
  - 執行到 **TRAPON** 敘述後，由於控制模組會先檢查 **TRAPOFF** 旗號(此時尚未被清除)，並決定不執行下一個事件，接著才清除 **TRAPOFF** 旗號；因此程式流程並不會在執行 **TRAPON** 敘述後立即移轉至新事件，而是必須等到執行下一個 **RETURN** 敘述後，才因為發現 **TRAPOFF** 旗號已清除而將程式流程移轉至事件副程式，如此可以確保在本事件副程式完全處理完畢後才允許其他事件插入執行。
- 若未利用 **TRAPOFF** 與 **TRAPON** 兩指令來控制程式流程，可能發生以下的問題：
  - Tag 事件副程式執行期間，有可能因新的事件觸發而跳至其他 Tag 事件副程式執行，導致程式指令執行的順序難以掌控，以及變數值可能被其他副程式修改等問題。
  - Tag 副程式未執行完畢即跳至其他 Tag 副程式時，未執行完畢的副程式會被暫時記載於堆疊中，等新的 Tag 副程式執行完畢後，再返回原事件繼續執行。若遇大量 Tag 事件觸發，導致累積許多無法完成執行的 Tag 副程式，將可能引發「超出堆疊」的系統錯誤，導致 ***SmartScript*** 控制程序模組中止執行。
- 除前述說明外，使用 Tag 事件副程式亦須考慮以下因素：
  - Tag 事件副程式執行期間，任何 Tag 的資料均有可能因其他圖控模組的執行而引發改變，程式設計者不應假設在副程式處理的過程中，Tag 資料均與 Tag 事件觸發瞬間相同。為減輕此一特性的影響，可在 Tag 事件副程式開始時先將參與運算的 Tag

資料存到變數中，再就變數資料內容進行處理。但請注意在將 Tag 資料轉移至變數的指令執行過程中，Tag 資料還是有可能發生變化。

- **事件佇列**中對於同一段 Tag 事件副程式僅允許有一筆紀錄，因此在 **TRAPOFF** 期間，
  - 若有多個 Tag 行標共用同一段事件副程式，這些不同 Tag 的資料變化僅會在佇列中加入一筆事件副程式記錄，該事件副程式在佇列中的順序以第一次觸發時加入佇列的位置為準。
  - 同一個 Tag 的多次資料變化，亦僅會在佇列中加入一筆事件副程式記錄，該事件副程式在佇列中的順序以第一次觸發時加入佇列的位置為準。
- 由於 Tag **事件佇列**仍有容量的限制(512 個 Tag 事件)，因此 Tag 事件副程式的執行時間不宜過長，否則在 **TRAPOFF** 的情況下若有超過 512 個新事件被觸發，將因佇列已滿，新事件會被捨棄而不予處理。

跟隨面板畫面執行

在某些應用場合，可能需要隨著某一面板畫面開啟後，執行一段 ***SmartScript*** 程式，而當畫面被關閉後，即同時結束 ***SmartScript*** 程式。此種應用方式，只須結合前例中以圖控物件觸發 ***SmartScript*** 程式的執行，再加上適當的迴圈控制條件即可達成。

在下面的範例中，當使用者按下「開啟面板」按鈕後，即開啟一子面板，同時也觸發一段 ***SmartScript*** 程式，來累積計算此面板被開啟的時間。當面板被關閉，計算程式也隨之結束。此種作法可以改善 ***SmartScript*** 程式的使用效率，讓一些只跟畫面開啟後的顯示或操作有關的計算，只在畫面被開啟的時後執行，以避免浪費系統時間與資源。

■ ***SmartPanel*** 人機介面模組中的設定

本範例中使用了兩個面板：WKS1.PNL(初始面板)與 SUBPANEL.PNL(子面板)，其設定分別說明如下：

WKS1.PNL

物件	參數設定	TAG 設定	說明
按鈕	物件抬頭：開啟面板	**OpenPnl**	按下此按鈕會將名稱為 **OpenPnl** 的 TAG 值設為 1
面板視窗啟動器	X：0 Y：0 寬度：10000 高度：8500 面板檔名：~1\subpanel.pnl 物件風格：子視窗	**OpenPnl**	當 **OpenPnl** 為 1 時會開啟檔案名稱為 subpanel.pnl 的面板，**OpenPnl** 為 0 時即關閉

SUBPANEL.PNL

物件	參數設定	TAG 設定	說明
靜態文字顯示器	物件抬頭：累計開啟時間		顯示「累計開啟時間」文字
靜態文字顯示器	物件抬頭：時		顯示「時」文字
靜態文字顯示器	物件抬頭：分		顯示「分」文字
靜態文字顯示器	物件抬頭：秒		顯示「秒」文字
文字錶頭	小數點位數：0	**OpenHour**	顯示累計開啟時間的小時部份
文字錶頭	小數點位數：0	**OpenMin**	顯示累計開啟時間的分鐘部份
文字錶頭	小數點位數：0	**OpenSec**	顯示累計開啟時間的秒鐘部份
程式執行器	指令行： CONTROL.EXE ..\PROJECT\PROJ1\CSL\OPENTIME.CSL /R	**OpenPnl**	當 **OpenPnl** 值變為 1 即執行指定的 ***SmartScript*** 程式

註：控制程序檔通常應位於專案目錄下的 CSL 子目錄中，本例之專案名稱為 PROJ1，控制程序名稱則為「OPENTIME」。

■ TAG **與變數說明**

名稱	種類	說明
OpenPnl	TAG	用以控制 SUBPANEL.PNL 面板開啟的 TAG
OpenHour	TAG	面板累計開啟時間的小時部份
OpenMin	TAG	面板累計開啟時間的分鐘部份
OpenSec	TAG	面板累計開啟時間的秒鐘部份
$TIME	系統 TAG	其值每秒會自動加 1 的系統 TAG，可用作一秒計時器
OpenTime%	整數變數	用以儲存面板累計開啟時間總秒數的變數
PrevTime%	整數變數	用以儲存前一運算時刻的**$TIME** 值的變數

■ **程式碼**

行號	程式碼
1	OpenTime%={OpenHour}*3600+{OpenMin}*60+{OpenSec}
2	PrevTime%={$TIME}
3	WHILE({OpenPnl})
4	LOOP
5	END
6	{$TIME}:
7	OpenTime%=OpenTime%+{$TIME}-PrevTime%
8	{OpenHour}=OpenTime%/3600
9	{OpenMin}=(OpenTime%\3600)/60

行號	程式碼
10	{OpenSec}=(OpenTime%\3600)\60
11	PrevTime%={$TIME}
12	RETURN

註：上列行號部份的目的只是便於本節中的說明，並不屬於程式碼的一部份。

■ **程式說明**

第 1 行： 將前一次開啟的累積時間換算為秒數存到 OpenTime%。

第 2 行： 設定 PrevValue%初值。

第 3 行： 以 **OpenPnl**=1 為進入條件，開始一 While 迴圈。當 SUBPANEL.PNL 關閉導致 **OpenPnl**=0，則迴圈停止，程式也將結束。**OpenPnl** 亦在圖控面板中用作開始執行此段程式之 TAG。

第 4 行： 迴圈末端。

第 5 行： 主程式結束，***SmartScript*** 控制語言模組也隨之結束。

第 6 行： 以$TIME 觸發的副程式之啟始標示(Label)。每當$TIME 值改變，即中斷主程式，而執行此標示以下的副程式。

第 7 行： 累積計算面板開啟時間。其中 PrevValue%為前一運算時刻的 Tag1 值，{**$TIME**}-PrevTime%則為前一運算時刻迄此次運算所經過之時間，此敘述將此一差值加入累積時間。

第 8 行： 計算累積時間的小時部份。

第 9 行： 計算累積時間的分鐘部份。

第 10 行： 計算累積時間的秒鐘部份。

第 11 行： 更新 PrevTime%值。

第 12 行： 副程式結束，返回原主程式中斷處繼續執行。

通訊上的應用

SmartScript 程式也可以用來撰寫通訊驅動程式，以便與 Lab-LINK 沒有支援的 I/O 裝置進行通訊。以下的範例即被用來與一特殊儀器溝通，此儀器採用非常簡單的 ASCII 通訊協定，只要下達「#S00\r」命令，即傳回所需的資料。傳回的資料長度固定為 11 個字元，其中數值的部份在第 4~8 個字元。範例程式將每隔 1 秒鐘對儀器下一次命令，由傳回的資料中擷取所需的數值部份，並傳給名稱為 **Data-01** 的 TAG。此程式的執行方式，可仿效本章第一個範例隨圖控專案啟動執行，並隨圖控一起結束的執行方式，因此關於執行此程式的方式將不再贅述。

■ TAG 與變數說明

名稱	種類	說明
Data-01	TAG	用以代表儀器資料數值的 TAG
C$	字串變數	用以儲存讀取資料命令字串的變數
R$	字串變數	用以儲存儀器傳回的資料字串之變數
V$	字串變數	用以儲存儀器傳回的資料中數值部份之字串變數
TickBeg	變數	用以儲存前一運算時刻的 TICK 值的變數
CommLoop	Label	GOTO 迴圈的 Label

■ 程式碼

行號	程式碼
1	C$ = "#S00\r"
2	COMOPEN 1, "COM1:9600,N,8,1", 1024, 256
3	IF (! FCHECK(1))
4	MESSAGE " Test Program", "Can not open communication port"
5	STOP
6	ENDIF

行號	程式碼
7	CommLoop:
8	WRITE 1, C$
9	WHILE (FLEN(1) < 11)
10	LOOP
11	READ 1, R$, 11
12	V$ = MID$(R$, 4, 5)
13	{Data-01} = VAL(V$)
14	TickBeg = TICK()
15	WHILE (TICK()-TickBeg < 1000)
16	LOOP
17	GOTO CommLoop:
18	END

註：上列行號部份的目的只是便於本節中的說明，並不屬於程式碼的一部份。

■ **程式說明**

第 1 行： 將讀取資料的命令存到字串變數 C$。「\r」代表 Carriage Return。

第 2 行： 開啟通訊埠 COM1，通訊參數為 9600, N, 8,1，接收佇列(Interrupt-driven receive-queue)長度為 1024 個字元，而傳送佇列(Interrupt-driven transmit-queue)長度為 256 個字元。

第 3-6 行： 通訊埠開啟失敗的處理。程式會在畫面上顯示一錯誤訊息，經使用者確認後即結束程式

第 7-17 行： 處理通訊的 GOTO 迴圈。

第 8 行： 送出讀取資料的命令字串 C$。

第 9-10 行： 等待資料傳回，即等到資料接收佇列中至少有 11 個字元的資料。

第 11 行： 由接收佇列中讀取前 11 個字元，並存到 R$。

第 12 行： 擷取 R$中的第 4-8 個字元，即數值部份的字串，並存到 V$。

第 13 行： 利用 VAL 函數將 V$由字串轉為數值，並送給 **Data-01**。

第 14-16 行： 等待 1000 個 tick(1000 msec)，即 1 秒鐘。

第 17 行： 回到 CommLoop 的開頭，重新執行迴圈。

第 18 行： 程式結束。

第四章 基本語法

概要

SmartScript 模組提供完整的程式設計環境給使用者，提供與其他模組溝通的管道，以做為 **Lab-LINK** for Windows 內建的程式設計工具。***SmartScript*** 模組包含了類似 BASIC 或 C 語言的的大部分的指令與功能，而且也包含圖控所需的特殊功能。

SmartScript 模組提供檔案存取、非同步裝置的控制、以及播放音效的能力。也提供了一個具有除錯功能的編輯器，以簡化軟體的發展和程式碼的修改。

程式行格式

SmartScript Language 程式行格式如下(方括弧表可省略項目)：

[label:] statement [=> statement ……] <ENTER>

必要時，同一行中可以寫多個命令敘述，但是同一行的各命令敘述間必須以「=>」(then 符號)來分隔。

行標(Line Label)

程式行之前可以加行標。當程式分岐(branching)時行標被用做參考索引。行標可以由字母、數字及底線組成。行標的長度須為 1 至 16 個字元，而且第一個字元必須是字母。某些特殊字元也可以使用，如下所示：

行標可以是保留字或是包含保留字，但並不建議你這樣做。所謂保留字包括所有的指令、命令敘述、功能名稱和運算子(operator)。

當一個行標被大括弧括著時，這是一種特別的行標，叫做 TAG 行標。在這種情況下，括弧內的行標名稱一定是個 TAG 名稱。只要相應的 TAG 改變，原來執行中的程式將被中斷，此 TAG 行標的敘述區塊(statement block)將被執行。在敘述區塊執行後，原來的程式將會繼續執行。

字元集

SmartScript 模組所使用到的字元集包括字母(A-Z, a-z)、數字符號(0-9)以及 16 進位數字中的 A-F 或 a-f 以及其他的特別字元。有些字元在 ***SmartScript*** 模組有其特別的意義：

資料型態附加字元(Data-Type Suffixes)

% 整數

$ 字串

{} TAG

數學運算子

. 小數點

+ 加號

- 減號

* 乘號

/ 除號（斜線）

\ 整數除法餘數符號（反斜線）

^ 指數符號(up arrow or caret)

關係運算子

> 大於

< 小於

>= 大於或等於

<= 小於或等於

== 等於

!= 不等於

邏輯運算子

! 邏輯反相(NOT)

& 邏輯 AND

| 邏輯 OR

位元運算子

NOT 反向

<< 向左移位

>> 向右移位

^< 向左循環

>^ 向右循環

AND AND 運算

XOR XOR 運算

OR OR 運算

特殊運算子

= 設值

: 行標附加字元

@ 中斷點符號

, 參數分隔字元

=> Then 符號

// 註解行

常數

常數是 ***SmartScript*** 模組執行時的實際數值。有兩種類型的常數：**字串**和**數值**。

字串常數

一個字串常數是以雙引號括起來一連串的字元，字串常數的例子：

“$25,000.00”

"This is a test"

以雙引號包圍起來的字元可以是**字母**、**數字**、或**特別的字元**。控制碼(Escape sequences)是用一連串的字元來代表某些特殊字元。控制碼表列如下：

Sequence	Name
\n	New Line
\r	Carriage Return
\t	水平 Tab
\"	雙引號
\\	反斜線符號
\b*nnnnnnnn*	以二進位數字表示的 ASCII 字元
\B*nnnnnnnn*	
\o*nnn*	以八進位數字表示的 ASCII 字元
\O*nnn*	
\0*nnn*	
\d*nnn*	以十進位數字表示的 ASCII 字元
\D*nnn*	
nnn	
\x*nn*	以十六進位數字表示的 ASCII 字元
\X*nn*	
\h*nn*	
\H*nn*	

數值常數

數值常數是正數或負數，但不能含有逗號。數值常數有七種類型：

1. 整數常數

   介於-2147483648 和+2147483647 之間的整數，沒有小數點。

2. 實數常數

   正實數或負實數，即是含有小數點的數字。

3. 浮點實數常數

   以指數的形式來表示的正實數或負實數〈與科學記號類似〉。浮點實數常數由正負號，含有小數點的數字再加上字母 E 以及可帶有正負號的整數(即指數)組成。浮點實數常數所容許的範圍是從 10^{-308} 到 10^{+308}，以及可多達 16 位的有效數字。

   範例：

   476.983E-6 = 0.000476983

523E3 = 523000

特殊關鍵字"PI"代表一個浮點實數常數 － 圓週率 □ (3.141592653589793)。

4. 十進位常數(可帶有前置符號 0d 或 0D)

   範例：

   1024

   0d13

   0D27

5. 十六進位常數(可帶有前置符號 0x、0X、0h 或 0H)

   範例：

   0x41FF

   0X0D

   0hE6

   0H1A0

6. 八進位常數(可帶有前置符號 0o 或 0O)

   範例：

   0o10

   0O361

7. 二進位常數(可帶有前置符號 0b 或 0B)

   範例：

   0b1010

   0B11000011

變數

變數用於代表 ***SmartScript*** 模組程式中的某個的數值。變數的值可由程式設計者在程式中明確地設定，或者也可以由程式中的計算產生後設定。在給一個變數設定其值之前，它的值被假設為零。

變數名稱與宣告字元

在變數名稱中所允許使用的字元包括字母、數字和底線。請注意字母的部份區分大小寫，因此「VAR-1」與「var-1」會被視為兩個不同的變數。變數名稱的長度必須為 1 到 16 個字元，而且第一個字元一定是字母。您也可以使用特殊的宣告字元，宣告字元並不計入變數名稱的長度。茲就宣告字元的使用，說明如下。

變數名稱不可以是保留字，但是名稱中可以包含保留字。保留字包括所有命令、敘述句、函數名稱和運算子名稱。

變數可表示一個數值或字串，字串變數名稱是以$符號作為最後的字元。例如：A$ = “Hello the world”。$符號為變數型別宣告字元，換言之，所宣告的此一變數將用以代表一個字串。

數值變數名稱可被宣告為整數或是實數〈倍精確度實數〉, 對於這些變數的名字的型別宣告字元如下：

% 整數變數

數值變數名的預設型別是實數。

變數名稱的例子如下：

N% 宣告一個整數值

LIMIT 宣告一個實數值

MSG$ 宣告一個字串值

Tag 變數

在 ***SmartScript*** Language 中，有一種叫做 TAG 變數的特別變數是用來存取和操縱 TAG 資料的。TAG 變數應該命名如下：

{*tag-name*} *tag-name* 是 TAG 的名字，如果 TAG 不存在，它就將會被創造出來。{*tag-name*} 的用法與一般數值變數相同〈倍精確度實數〉, 但它代表的是 TAG 的數值欄位。任何對於 TAG 變數的修改也將會反應在 *tag-name* 的數值欄位內容。

{*tag-name*.$} *tag-name* 是 TAG 的名字, 如果 TAG 不存在, 它將會被創造出來。{*tag-name*.$} 的用法與一般字串變數相同，但它代表的是 TAG 的訊息欄位。任何對於 TAG 訊息變數的修改也將會反應在 *tag-name* 的訊息欄位內容。

{*tag-name*.t}或{*tag-name*.T} *tag-name* 是 TAG 的名字, 如果 TAG 不存在, 它將會被創造出來。{*tag-name*.t} 的用法與一般實數變數相同, 但它代表的是以實數表達的 TAG 之日期與時間欄位，其中整數部份代表日期, 小數部份代表以天為單位的時間。任何對於 TAG 日期時間變數的修改也將會反應在 *tag-name* 的日期與時間欄位內容。請注意對 TAG 的數值或訊息欄位的修改，均會使系統自動將 Tag 的日期與時間欄位更新為資料發生變動的日期與時間。

{*tag-name*.s}或{*tag-name*.S} *tag-name* 是 TAG 的名字, 如果 TAG 不存在, 它將會被創造出來。{*tag-name*.s} 的用法與一般整數變數相同，但它代表的是以 16 位元編碼表達的 TAG 之狀態欄位，各位元的定義如下表，使用者可利用位元運算子中的>>與 AND 來擷取對應位元之值以取得其狀態資料：

位元	分類	數值說明	Right Shift >>	AND MASK
0~2	輸入狀態	0: 未知 1: 未定 2: 離線 3: 連線	0	0x0007
3	輸出狀態	0: 輸出成功 1: 輸出失敗	3	0x0008
4~6	警報狀態	0: 正常 1: HI 警報 2: LO 警報 3: HH 警報 4: LL 警報	4	0x0070
7	確認狀態	0: 未確認 1: 已確認	7	0x0080
8~16	系統保留			

陣列變數

陣列是可以用同一變數名稱來代表的一組值。陣列中的每一個元素可以用陣列變數名稱加上一個做為索引數字的整數或整數運算式來表示。一個陣列變數名稱所帶的索引數字的數量與其維數相同。例如，V(6)代表一維陣列中的一個數值。M(2, 7)則代表二維陣列中的一個數值，以此類推。一個陣列所能宣告的最大維數是 3。宣告陣列中的元素數量最多為 8192 個。

需要的記憶空間

變數:	**陣列**	**字串**
整數 4 Bytes 實數 8 Bytes	整數 4(每一元素) Bytes 實數 8(每一元素) Bytes	3 bytes 加上字串中的字元數

型別轉換

在有需要的時候，***SmartScript*** 模組可以將數值常數從某種資料型別轉換成另一種資料型別。請注意接下來所敘述的規則與例子。

1. 如果某一型別的數值常數被設定給不同類型的數值變數，這個數值將會被轉為變數的型別。(如果字串變數被設定等於一個數值或相反的情況，將會發生"資料型別不符(Type mismatch)"的錯誤。)

範例：

N% = 32.64

在這個敘述句被執行後，N%的值是 32

2. 在計算運算式之值時，所有算數運算或關係運算的運算元均將被轉成相同的精度，也就是與精確度最高的運算元相同的精確度的那個。同樣地，算數運算的結果也將以此精確度傳回。

   範例：

   A = 15.0 / 2

   這個算數運算是以倍精確度實數運算來完成的，其結果也將以倍精確度實數的數值傳回。在這個敘述句執行後，A 的值是 7.5。

   A = 9 / 5

   這個算術是以整數精度的運算完成的，其結果也將以整數的數值傳回。在這個敘述句被執行後，由於運算結果被指定給 A，而 A 是倍精度實數變數，因此數值會再被轉成倍精度實數，而得到 A 的值為 1.0。

3. 邏輯運算子會先將將他們的運算元轉成整數，並且將傳回整數結果。

   範例：

   N% = 2.5 XOR 1

   在這個敘述句被執行後，N%的值是 3

4. 當浮點數被轉變成整數時，會直接將小數部份去掉。

   範例：

   N% = 6.67

   M% = -6.67

   在這個敘述句被執行後，N%的值是 6，而 M%的值是 -6。

運算式(Expressions) 與運算子(Operations)

運算式可以是一個字串或數值的常數，或是一個變數，它也可能是以運算子結合常數和變數來產生的單一數值。

運算子是用來進行數值的數學或邏輯運算，***SmartScript*** 模組提供的運算子，可以分成四個類別：

1. 算術運算子
2. 關係運算子
3. 邏輯運算子
4. 位元運算子

算術運算子

算術運算子依其計算優先順序排列如下:

運算子	運算種類	範例
-	取負值	-X
^	乘冪	X ^ Y
*	相乘	X * Y
/	相除	X / Y
\	取餘數	X \ Y
+	相加	X + Y
-	相減	X - Y

要改變運算的順序，可以用括號。括號內的運算會先予以執行。在括號當中，則依照正常的計算優先順序。

以下是一些代數運算式的範例，以及的 ***SmartScript*** 中的對應語法。

代數運算式	***SmartScript*** 模組運算式
$X + 2Y$	X + 2 * Y
$X - \frac{Y}{Z}$	X – Y / Z
$\frac{X \cdot Y}{Z}$	X * Y / Z
$\frac{X + Y}{Z}$	(X + Y) / Z
$(X^2)^Y$	X ^ 2 ^ Y
X^{Y^Z}	X ^ (Y ^ Z)
$X \cdot -Y$	X * -Y

整數的餘數運算

整數的餘數運算是用反斜線符號(\)來表示，在完成餘數運算之前，運算子的小數部份會被捨去成為整數，並且傳回兩個整數相除所得的整數餘數值。

範例：

N% = 11.9 \ 3

這個敘述句執行後，N%的值是 2。

除以 0

如果在運算式的計算中，遇到除以 0 的情況，會出現"除以零(Division by Zero) !!"的錯誤訊息並且停止執行。

關係運算子

關係運算子通常用於比較兩個值，比較的結果是“True” (1) 或 “False” (0)，然後依據此結果來決定程式流程。關係運算子依其運算優先順序列表如下：

運算子	運算種類	範例
>	大於	X > Y
<	小於	X < Y
>=	大於或等於	X >= Y
<=	小於或等於	X <= Y
==	等於	X == Y
!=	不等於	X != Y

當算術運算子與關係運算子同時出現在一個運算式時，總是先完成算術運算，例如以下的運算式：

X + Y < (Z + 5) / W

如果 X 的值加上 Y 的和小於 Z+5 除以 W 的商，則其結果為 true。以下是一些其他的例子：

B% = LIMIT > 95.0

IF (B%) => GOSUB HiAlarm:

IF (SIN(X) < 0) => GOTO Label1:

IF (N% \ 2 != 0) => L% = L% + 1

IF (A$ == “EXIT”) => STOP

邏輯運算子

邏輯運算子用於多重的關係或布林運算，邏輯運算子會傳回"True" (1)或 "False" (0)。邏輯運算子依其運算優先順序列表如下：

運算子	運算種類	範例
!	Logic NOT	! (X > Y)
&	Logic AND	X > 10.5 & Y < 21.3
|	Logic OR	N% < 0x30 | N% > 0x39

邏輯運算的結果如下表所顯示：

X	! X
True (非 0)	False (0)
False (0)	True (1)

& (Logical AND)		
X	Y	X & Y
True (非 0)	True (非 0)	True (1)
True (非 0)	False (0)	False (0)
False (0)	True (非 0)	False (0)
False (0)	False (0)	False (0)

| (Logical OR)		
X	Y	X | Y
True (非 0)	True (非 0)	True (1)
True (非 0)	False (0)	True (1)
False (0)	True (非 0)	True (1)
False (0)	False (0)	False (0)

就像關係運算子可用來決定程式流程，邏輯運算子則可以連結兩個或兩個以上的關係式，並傳回一個 true 或 false 的值，作為決定程式流程方向的依據〈參照 IF 敘述〉

範例：

```
IF (A >= 0) & (A <= 100) => GOTO Label1:
B% = (N% < 48) | (N% > 57)
IF ! B% => GOTO Label2:
```

位元運算子

位元運算子提供個別位元的運算操作功能，並傳回位元運算的結果。在進行位元運算之前，會先將運算元取整數再進行運算。運算後，它也將傳回了位元運算結果的整數值，位元運算子依其運算優先順序列表如下：

運算子	運算種類	範例
NOT	反相(Invert)	NOT N%
<<	向左移位(Shift left)	X << 3
>>	向右移位(Shift right)	X >> 8
^<	向左循環(Rotate left)	X ^< 1
>^	向右循環(Rotate right)	X >^ 2
AND	AND	X AND 0x00FF
XOR	XOR	X XOR 0b10101010
OR	OR	X OR Y

位元運算的結果如下表所顯示：

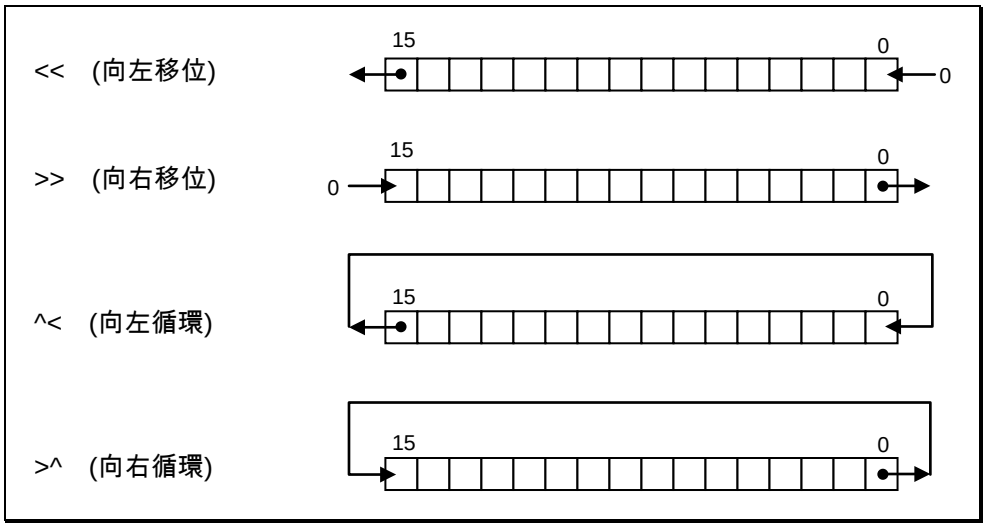
NOT	
X	NOT X
1	0
0	1

AND		
X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

XOR		
X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

OR		
X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

Shift and Rotate



字串運算

字串可使用"+"來連結

範例：

A$ = "ABC"

B$ = "DEF"

C$ = A$ + " " +B$

在這些敘述執行後，C$的值是"ABC DEF"

數值比較用的關係運算子，同樣可以用來做字串的比較：

> >= < <= == !=

字串的比較方式是由字串中一次取一個字元，逐一比較其 ASCII 碼。如果有任何一個字元的

ASCII 碼不一樣，字串就不相等。此時即停止比較，含有數值較低的 ACSII 碼的字串視為小於含有數值較高的 ACSII 碼的字串。如果在個別字元逐一比較期間，其中一個字串先到達其末端，較短的字串被視為較小。字串前後所包含的空白字元也將被列入比較。

範例：

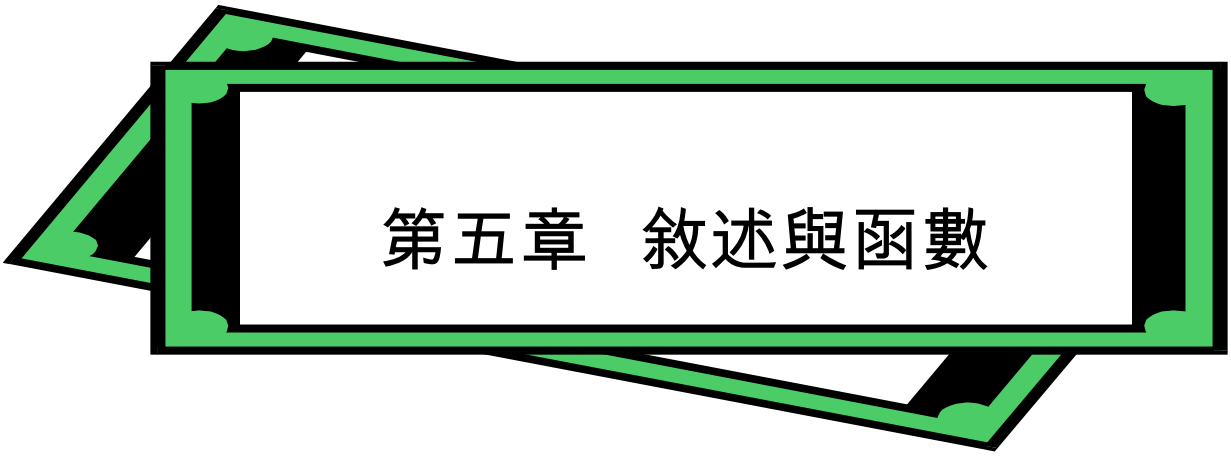
“IS” < “IT”

“OK” != “ok”

“This ” > “This”

“kg” > “KG”

“NOT” < “NOTE”



第五章　敘述與函數

概要

本章將詳細說明 ***SmartScript*** 模組所提供的各種敘述及函數。多數說明中均包含五個部分：**功能**、**版本**、**格式**、**註解**及**範例**：

功能	對敘述或函數的功能做簡短的解釋。
版本	指出何種 ***SmartScript*** 模組的版本支援此一敘述或函數。
格式	利用以下的慣用字體來描述此敘述或函數的語法： **大寫字體(CAPS)**　大寫字母代表關鍵字，使用時須完全正確地輸入 ***斜體字(Italics)***　斜體字代表由使用者輸入的變動資訊 [　]　方括弧表示選用參數 **…**　省略符號表示必要時可以重複一個項目許多次 |　直線符號表示二選一的選擇 ，　格式中的標點符號不應省略，除非是出現方括弧或省略符號之中。
註解	對於此敘述或函數的正確使用方式，做進一步的詳細說明。
範例	舉例說明使用此敘述或函數的各種使用方式，必要時會加強說明特殊的用法。

關鍵字：依程式功能需求分類

程式功能需求	關鍵字
程式流程控制	IF...ELSE...ELSEIF...ENDIF, CHOICE(), SWITCH...CASE...DEFAULT...ENDSW, FOR...LOOP, WHILE...LOOP, CONTINUE, EXIT, GOTO, GOSUB...RETURN, STOP, END IDLE SLEEP
數學運算	SIN(), COS(), TAN(), ASIN(), ACOS(), ATAN(), SINH(), COSH(), TANH(), EXP(), LN(), LOG(), SQRT(), ABS(), RAND(), FAC(), MIN(), MAX(), INT(), FMBCD(), FMFLT(), FMDBL()
字串處理	LEN(), VAL(), IVAL(), ASC(), STR$(), ISTR$(), CHR$(), STRING$(), LEFT$(), RIGHT$(), MID$(), LOWER$(), UPPER$(), LTRIM$(), RTRIM$(), SUM08(), XOR08(), CRC16(),CRC32(), TOKEN, FORMAT$(), DATETIME$(), RAWVAL(), VALRAW$()
系統時間取得	TIMER(), SECOND(), MINUTE(), HOUR(), DAY(), WEEKDAY(), MONTH(), YEAR(), TICK(), NOW(), NOW$(), DATETIME()
檔案輸出與輸入	CREATE, OPEN, COMOPEN, SEEK, READ, WRITE, FPRINT, CLOSE, FCHECK(), FLEN(), FPOS(), CD, MD, RD, COPY, MOVE, DEL, DIR$(), COMMODE, SHORTCUT, FILE$()
陣列的宣告	DIM

程式功能需求	關鍵字
TAG 事件的補捉	TRAPON, TRAPOFF
其他	PI, BEEP, PLAY, MESSAGE, MSGBOARD, EXEC, SETDIR, ALMTAG$(), ALARM(), TAG(), ALMGRP(), ALMPRI(), ERRID(), ERRORTAG, NERR(), RSTERR, PASS, SHUTDOWN, SYSINFO$(), TONE

敘述與函數

本章中將依英文字母排列的順序，列出並詳加說明所有的敘述及函數。

ABS() 函數

【功能】 ABS 傳回輸入引數的絕對值

【版本】 1.0(含)以上

【格式】 Y = **ABS(***numeric-expression***)**

【註解】 *numeric-expression* 可以是任何數值運算式。
此函數相等於代數運算式。

【範例】 Y = ABS(45.5 - 100) // Y 值為 54.5

ACOS() 函數

【功能】 ACOS 傳回輸入引數的反餘弦函數值。

【版本】 1.0(含)以上

【格式】 Y = **ACOS(***numeric*-expression**)**

【註解】 *numeric-expression* 可以是任何數值運算式。
這個函數相當於代數上的運算式 $y = \cos(x)^{-1}$。
ACOS 函數傳回值的單位為弧度(radians)，要將單位傳為度(degrees)，應將弧度值除以(PI /180)。

【範例】 Y = ACOS(0) / PI * 180 // Y 值為 90

ALARM() 函數

【功能】 ALARM 傳回指定 TAG 的警報狀態

【版本】 1.1(含)以上

【格式】 Y = **ALARM(***string*-expression**)**

【註解】 *string-expression* 必須是代表一 TAG 名稱的字串運算式。
這個函數傳回引數所指定的 TAG 當時的警報狀態。警報狀態值及代表意義如下：
0x00 無警報
0x01 High 警報(類比警報點)或警報(數位警報點)
0x02 Low 警報(類比警報點)
0x03 High-high 警報(類比警報點)
0x04 Low-low 警報(類比警報點)
0x81 經確認的 High 警報(類比警報點)或警報(數位警報點)
0x82 經確認的 Low 警報(類比警報點)
0x83 經確認的 High-high 警報(類比警報點)
0x84 經確認的 Low-low 警報(類比警報點)

【範例】 N% = ALARM(“AI-0001”)

ALMGRP() 函數

【功能】 ALMGRP 傳回指定 TAG 所屬的警報群組。

【版本】 1.2(含)以上

【格式】 *N%* = **ALMGRP(***string-expression***)**

【註解】 *string-expression* 必須是代表一 TAG 名稱的字串運算式。
這個函數傳回引數所指定的 TAG 所屬的警報群組。若該 *string-expression* 指定的 TAG 名稱未被定義為警報點，則傳回 0；若該 *string-expression* 為空字串，則傳回 -1。

【範例】 N%= ALMGRP(“Tag1”)

ALMPRI() 函數

【功能】 ALMPRI 傳回指定 TAG 的警報優先順序。

【版本】 1.2(含)以上

【格式】 *N%* = **ALMPRI(***string*-expression**)**

【註解】 *string-expression* 必須是代表一 TAG 名稱的字串運算式。
這個函數傳回引數所指定的 TAG 的警報優先順序。若該 *string-expression* 指定的 TAG 名稱未被定義為警報點，則傳回 0；若該 *string-expression* 為空字串，則傳回 -1。

【範例】 N%= ALMPRI(“Tag1”)

ALMTAG$() 函數

【功能】 ALMTAG$ 傳回當時系統中優先順序最高或最新的警報 TAG 的 TAG 名稱字串。

【版本】 1.1(含)以上

【格式】 *A$* = **ALMTAG$()**

【註解】 這個函數傳回一字串，此字串即為當時系統已發生的警報中最優先的警報之 TAG 名稱。所謂最優先的警報由以下邏輯決定：

(1) 同時發生多個警報，且全部或部份警報尚未被確認時，以未被確認的警報中，其「優先順序」設定值最高者為最優先；換言之，未被確認者優於已被確認者。
(2) 若所有警報均已被確認，則以被確認警報中，其「優先順序」設定值最高者為最優先。
(3) 當優先順序相同時，較晚發生的警報較優先。

最優先的警報亦等同於 ***SmartPanel*** 人機界面系統中「警報資料顯示板」物件上所顯示的警報訊息所對應的警報 TAG。

【範例】 A$ = ALMTAG$()

ASC() 函數

【功能】 ASC 傳回 *引數*字串中第一個字元的 ASCII 碼數值。

【版本】 1.0(含)以上

【格式】 *N%* = **ASC(***string-expression***)**

【註解】 *string-expression* 可以是任何字串運算式。

【範例】 N% = ASC("Q") // N% 值為 81

ASIN() 函數

【功能】 ASIN 傳回輸入引數的反正弦函數值。

【版本】 1.0(含)以上

【格式】 *Y* = ASIN(numeric-expression)

【註解】 numeric-expression 可以是任何數值運算式。
這個函數相當於代數上的運算式 $y = \sin(x)^{-1}$。
ASIN 函數傳回值的單位為弧度(radians)，要將單位傳為度(degrees)，應將弧度值除以(PI /180)。

【範例】 Y = ASIN(1) / PI * 180 // Y 值為 90

ATAN() 函數

【功能】ATAN 傳回輸入引數的反正切函數值。

【版本】1.0(含)以上

【格式】*Y* = **ATAN(***numeric- expression***)**

【註解】*numeric- expression* 可以是任何數值運算式。.

這個函數相當於代數上的運算式 $y = \tan(x)^{-1}$

ATAN 函數傳回值的單位為弧度(radians)，要將單位傳為度(degrees)，應將弧度值除以(PI /180)。

【範例】Y = ATAN(1) / PI * 180 // Y 值為 45

BEEP 敘述

【功能】BEEP 讓你電腦中的喇叭產生出 beep 的聲音

【版本】1.0 版(含)以上

【格式】**BEEP**

【註解】這個敘述從你電腦中的喇叭產生出 beep 的聲音

【範例】BEEP

CD 敘述

【功能】將目前路徑切換至引數所指定的路徑位置

【版本】4.0 版(含)以上

【格式】**CD** *directory-name*

【註解】*directory-name* 路徑名稱。必須是一個字串運算式，用以指定路徑。

【範例】CD “..\\Project\\Proj1\\Dat\” // 切換目前路徑至..\Project\Proj1\Dat\

CHOICE() 函數

【功能】CHOICE 根據條件運算式的結果，決定傳回 true 的運算式之值，或 false 運算式之值。

【版本】1.0 版(含)以上

【格式】*Y* = **CHOICE(***cond-expression, true-expression, false-expression***)**

【註解】*cond-expression* 可以是任何運算式。若其運算結果為 true (非 0)，則求出 *true-expression* 的值，並以其資料型別傳回了它的值，做為此函數的運算結果；如果它求出 false (0)的值，則計算 *false-expression* 的值，並以其資料型別傳回了它的值，做為此函數的運算結果。

true-expression 可以是任意數值運算式，它只有在當 *cond-expression* 之值為 true 時，才會被計算。

false-expression 可以是任意數值運算式，它只有在當 *cond-expression* 之值為 false 時，才會被計算。

【範例】Y = CHOICE(Y >= 0, Y + 1, Y - 1)

CHR$() 函數

【功能】CHR$將輸入引數當作 ASCII 碼之數值，並傳回該數值在 ASCII 碼中所對應的字元。

【版本】1.0 版(含)以上

【格式】*M$* = **CHR$(***numeric- expression***)**

【註解】*numeric- expression* 是一個*範圍*從 0 到 255 數值運算式。

【範例】M$ = CHR$(*13*) + CHR$(10)　　// M$值為“\r\n”

CLOSE 敘述

【功能】CLOSE 關閉一個已開啟的檔案或裝置(device)。

【版本】1.0 版(含)以上

【格式】**CLOSE [***file-number***]**

【註解】*file-number*　是一個用以標示某個已被開啟的檔案或裝置的數字。
沒有指定引數的 CLOSE 會關閉所有已開啟的檔案和裝置。

【範例】CREATE 1, "TEST.TXT"
CLOSE 1

COMMODE 敘述

【功能】COMMODE 用來改變已經開啟的通訊裝置檔之通訊參數

【版本】4.0 版(含)以上

【格式】**COMMODE** *file-number,comm-param*

【註解】*file-number* 是一個整數的運算式，其值應對應至一個已經開啟的檔案編號(file number)。
comm-param 是一個字串運算式，用以改變通訊裝置的控制資訊。該字串的格式必須與 DOS 中的 MODE 敘述之命令行參數相同。

【範例】COMOPEN 1, “COM2:9600,N,8,1”, 2048, 1024
COMMODE 1,”COM2:19200,E,7,1”

COMOPEN 敘述

【功能】COMOPEN 用以打開一個通訊裝置檔(communications file)。

【版本】1.0 版(含)以上

【格式】**COMOPEN** *file-number, comm-param, in-queue, out-queue*

【註解】*file-number* 是一個整數的運算式，其值應可對應至一個有效的檔案編號(file number)。只要通訊裝置一直開著，而且有被其他 I/O 通訊敘述使用，該數值即被關聯於此一通訊裝置。有效的檔案編號是 1 到 16。
comm-param 是一個字串運算式，用以指定通訊裝置的控制資訊。該字串的格式必須與 DOS 中的 MODE 敘述之命令行參數相同。
in-queue 是一個整數運算式，用以指定中斷接收佇列的大小。
out-queue 是一個整數運算式，用以指定中斷傳送佇列的大小。

【範例】COMOPEN 1, “COM2:9600,N,8,1”, 2048, 1024

CONTINUE 敘述

【功能】CONTINUE 將程式流程跳到其所在的 FOR 或 WHILE 迴圈的下一輪運算。

【版本】1.0 版(含)以上

【格式】**CONTINUE**

【註解】詳見 FOR … LOOP、WHILE … LOOP 及 EXIT 敘述。

【範例】FOR N%=0 TO 10

```
    .
    .
    IF (N% == 3) => CONTINUE
    .
    .
    .

LOOP
```

COPY 敘述

【功能】COPY 將指定的檔案或資料夾複製到指定的路徑位置。

【版本】4.0 版(含)以上

【格式】**COPY** *source-path,target-path*

【註解】*source-path* 是一個字串的運算式，用來描述來源檔案或資料夾的路徑，
target-path 是一個字串運算式，用以描述目的檔案或資料夾的路徑。
複製檔案時，若來源檔案不存在，目標路徑不存在，或因其他原因導致無法成功複製檔案，則 COPY 敘述失敗,但敘述失敗僅會被控制程序模組計為一次內部錯誤(可由 ERRID()得知錯誤碼)，程式流程則將繼續。
複製檔案時，若目標檔案已存在，COPY 敘述會用來源檔案直接覆蓋目標檔案。

【範例】COPY “C:\\Proj1\\Log.txt”,” C:\\Proj2\\LogFile.txt”

COS() 函數

【功能】COS 傳回輸入引數的餘弦函數值。

【版本】1.0 版(含)以上

【格式】*Y* = **COS(***numeric- expression***)**

【註解】*numeric- expression* 必須是一個以弧度(radians)為單位的角度值
這個函數相當於代數運算式 $y = \cos(x)$。
將度(degrees)還算為弧度(radians)，應將度乘以(PI / 180)。

【範例】Y = COS(180 * (PI / 180)) // Y 值為 –1

COSH() 函數

【功能】COSH 傳回輸入引數的雙曲線餘弦函數值。

【版本】1.0 版(含)以上

【格式】*Y* = **COSH(***numeric- expression***)**

【註解】*numeric- expression* 可以是任意數值運算式。
這個函數相當於代數上的運算式 $y = \cosh(x)$ 。

【範例】Y = COSH(1) // Y 值為 1.543081

CRC16() 函數

【功能】CRC16 傳回輸入引數經 CRC-16 演算法計算所得的結果值。CRC-16 是一種通常用於通訊資料錯誤檢測的演算法。

【版本】1.0 版(含)以上

【格式】*N%* = **CRC16(***string-expression***)**

【註解】*string-expression* 可以是任意字串運算式。

【範例】N% = CRC16(“\x02ABC\x03”) // N%值為 55704

CRC32() 函數

【功能】CRC32 傳回輸入引數經 CRC-32 演算法計算所得的結果值。CRC-32 是一種通常用於通訊資料錯誤檢測的演算法。

【版本】4.0 版(含)以上

【格式】*N%* = **CRC32(***string-expression***)**

【註解】*string-expression*　可以是任意字串運算式。

【範例】N% = CRC32(“\x02ABC\x03”)　　// N%值為 -1375105033

CREATE 敘述

【功能】CREATE 產生一個新檔案，並且將它打開。

【版本】1.0 版(含)以上

【格式】**CREATE** *file-number, file-name*

【註解】***file-number*** 是一個整數的運算式，其值應可對應至一個有效的檔案編號(file number)。只要檔案一直開著，而且有被其他檔案 I/O 敘述使用，該數值即被關聯於此一檔案。有效的檔案編號是 1 到 16。

file-name 是一個字串運算式，用以命名新產生的檔案。

如果檔案不存在，此敘述會建立一個新檔案，並且打開它以便寫入。如果檔案已經存在，此敘述會將檔案大小設為零，並且打開它以便讀取或寫入。此敘述打開檔案時，同時也會將檔案讀寫指標(read-write pointer)設到檔案的開頭。

【範例】CREATE 1, "TEST.TXT"

CLOSE 1

DATETIME() 函數

【功能】將內容為日期時間字串的輸入引數轉為一時間時數。

【版本】4.0.0.10 版(含)以上

【格式】*N%*= **DATETIME$(***string-expression***)**

【註解】*string-expression* 是一個字串的運算式，其格式應為一代表日期時間的字串，此函數的輸出是一個實數，其數值的意義相當於 Tag 變數表達的 TAG 日期時間欄位。輸入引數之日期時間字串格式為"*mm/dd/yyyy hh:ii:ss*"，其中 *yyyy* 為四位數的西元年份，*mm* 為兩位數的月份，*dd* 為兩位數的日期，*hh* 為為兩位數的小時，*ii* 為兩位數的分鐘，*ss* 為兩位數的秒鐘，整個字串的長度固定為 19 個字元。

【範例】N%= DATETIME$("01/01/2018 12:00:00")　　// N%為以輸入時間的實數表示

DATETIME$() 函數

【功能】將輸入引數轉為一日期時間字串。

【版本】4.0.0.10 版(含)以上

【格式】*M*$= **DATETIME$(** *numeric- expression***)**

【註解】*numeric-expression* 是一個實數的運算式，其值應為一代表日期時間的實數，例如以 Tag 變數表達的 TAG 日期時間欄位。輸出之日期時間字串格式為"*mm/dd/yyyy hh:ii:ss*"，其中 *yyyy* 為四位數的西元年份，*mm* 為兩位數的月份，*dd* 為兩位數的日期，*hh* 為為兩位數的小時，*ii* 為兩位數的分鐘，*ss* 為兩位數的秒鐘。

【範例】{Tag1.t}= DATETIME$("01/01/2018 12:00:00") // 將 Tag1 之日期時間設為 01/01/2018 12:00:00

DAY() 函數

【功能】DAY 傳回日期中日的部份。

【版本】1.0 版(含)以上

【格式】*N*% = **DAY()**

【註解】The 函數傳回日期中日的部份 (1~31) 。

【範例】N% = DAY()

DEL 敘述

【功能】DEL 可用於檔案或資料夾的刪除

【版本】4.0 版(含)以上

【格式】**DEL** *path*

【註解】*path* 是一個字串的運算式，用來描述檔案或資料夾路徑

若要刪除的檔案不存在，或因為檔案存取權限的限制導致檔案無法被刪除，則 DEL 敘述會失敗,但敘述失敗僅會被控制程序模組計為一次內部錯誤(可由 ERRID()得知錯誤碼,詳 ERRID()及 NERR())，程式流程則將繼續。

刪除檔案時，即使目標檔案被設為唯讀，DEL 敘述仍可成功刪除檔案。

【範例】DEL “C:\\Lablink\\Project\\Proj1\\Txt\\Log.txt”

DIM 敘述

【功能】DIM 指定陣列變數之索引數字的最大值，並據以分配所需的記憶體。

【版本】1.0 版(含)以上

【格式】**DIM** *variable***(***numeric-const* **[,** *numeric-const* **[,** *numeric-const***]])**

【註解】*variable* 可以是任何陣列變數的名稱。

numeric-const 指定陣列索引數字的最大值，必須是整數常數

DIM 敘述會將陣列中所有變數的初值設為 0。陣列的維數最大為 3，而每一個陣列所能容納的元素最多為 8192 個，至於索引數字的最小值則為 1。

此敘述是個不可執行的敘述。

【範例】DIM A(10)

A(1) = 6.5

DIR$() 函數

【功能】DIR$()可以查詢目前路徑、Windows 及系統路徑。

【版本】4.0 版(含)以上

【格式】M$ = **DIR$(***"NOW"*|*"WIN"*|*"SYS"***)**

【註解】*NOW* - 查詢目前所在路徑。
WIN - 查詢 Windows 所在路徑。
SYS - 查詢 Windows 系統所在路徑。

【範例】M1$ = DIR$("NOW") // M1$ = "C:\Lablink\System4"
M2$ = DIR$("WIN") // M2$ = "C:\Windows"
M3$ = DIR$("SYS") // M3$ = "C:\Windows\System32"

END 敘述

【功能】END 使程式停止執行，並關閉所有檔案。

【版本】1.0 版(含)以上

【格式】**END**

【註解】END 敘述可以置於程式的任何地方，以使程式執行停止。END 敘述不見得一定要放在程式的最後。

【範例】IF (A > 0) => END

ERRID() 函數

【功能】ERRID()可以得知最新一次錯誤的錯誤碼代號

【版本】4.0 版(含)以上

【格式】N% = **ERRID()**

【註解】ERRID()可以取得最新一次的錯誤代碼(可由附錄查詢)，0 表示無錯誤。
請注意由於只要有任一敘述或函數能正確執行，均會清除本函數所傳回的錯誤碼，因此本函數的敘述句須緊接在發生錯誤的敘述句之後方有效。

【範例】N% = ERRID()　　// N% = 0~37

ERRORTAG 敘述

【功能】ERRORTAG 可以指定一個 Tag 名稱，當 SmartScript 因程式內容發生錯誤，導致無法繼續執行而須終止時，將錯誤碼代號與錯誤訊息分別寫入此 Tag 的數值與訊息欄位。

【版本】4.0.1.1 版(含)以上

【格式】**ERRORTAG** *string- expression*

【註解】*string- expression*　　字串內容必須是一個合法的 TAG 名稱

若 *string-expression* 對應的 TAG 名稱不存在，該 TAG 將自動被創造出來，因此其內容必須符合 TAG 命名的原則。

當程式異常結束時，指定的 Tag 之數值會被設為錯誤碼，其訊息欄位則會顯示錯誤訊息內容、發生錯誤的 SmartScript 檔名與行號。

若本敘述在 Script 中被執行多次，則僅程式異常結束前最後一個被執行的敘述有效。若執行多份 SmartScript，為區分不同 Script 的錯誤，每一份 SnmartScript 應指定不同的 Tag 名稱做為參數，並建議將此敘述放在 Script 的開頭執行。

【範例】ERRORTAG=”ETag”　　// 指定 ETag 為用來存放錯誤代碼與訊息的 Tag

a=1/0

若以上範例的 SmartScript 檔名為 CSLPRG1,csl，則在執行以上敘述後，SmartScript 將因變數被除以零而異常終止，此時 ETag 的數將為 18，其訊息則為「ERROR 18:除以零!! @ CSLPRG1.csl Line:2」

EXEC 敘述

【功能】EXEC 用於打開指定的可執行檔或文件檔

【版本】1.0 版(含)以上

【格式】**EXEC "***file-name***" , "***parameter***" , "***show-mode***"**

【註解】*file-name* **檔案路徑及名稱**。必須是一個字串運算式，用以指定要開啟的檔案或資料夾。此敘述可開啟可執行檔或文件檔。

parameter **參數**。必須是一個字串運算式，用以指定要傳給應用程式的參數(如果 *file-name* 指定的是可執行檔)。 如果 *file-name* 指定的是文件檔或不需要參數的執行檔，*parameter* 應為空字串。

show-mode **顯示模式**。指定開啟檔案的應用程式之顯示模式。這個參數可以是以下列出的字串之一：

"MIN" 使視窗最小化，且為非使用中。

"MAX" 使視窗最大化。

"HIDE" 隱藏此視窗。

"ICON" 將使視窗最小化，但仍為使用中。

"NORMAL" 以該視窗的正常大小與位置顯示，且為使用中。

【範例】

1. 執行「筆記本」程式
   EXEC "NOTEPAD.EXE",“”, “NORMAL”
2. 執行「DBSaver」程式
   EXEC "C:\\LABLINK4\\SYSTEM4\\DBSAVER.EXE","..\\Project\\專案名稱\\DEMO.cfg","NORMAL"
3. 執行「報表」程式
   EXEC
   "C:\\LABLINK4\\SYSTEM4\\REPORT.EXE","..\\Project\\Ten-SE\\cfg\\Wks1\\report.cat","NORMAL"

EXIT 敘述

【功能】EXIT 終止包含其所在位置的最小 FOR 或 WHILE 迴圈的執行。

【版本】1.0 版(含)以上

【格式】**EXIT**

【註解】詳見 FOR … LOOP、WHILE … LOOP 及 CONTINUE 敘述。

【範例】FOR N%=0 TO 10

```
    IF (A$ == “QUIT”) => EXIT
LOOP
```

EXP() 函數

【功能】EXP 傳回輸入引數的自然指數值

【版本】1.0 版(含)以上

【格式】*Y* = **EXP(***numeric- expression***)**

【註解】*numeric- expression* 可以是任意數值的運算式。

此函數是相等於代數上的運算式 $y = e^x$。

【範例】Y = EXP(1) // Y 值為 2.718282

FAC() 函數

【功能】FAC 傳回輸入引數的階乘值。

【版本】1.0 版(含)以上

【格式】*Y* = **FAC(***numeric- expression***)**

【註解】*numeric- expression*　　必須是一個整數運算式。

此函數相當於代數上的運算式 $y = x!$。

【範例】Y = FAC(5)　　// Y 值為 120

FCHECK() 函數

【功能】FCHECK 決定指定的檔案號碼是否有效。

【版本】1.0 版(含)以上

【格式】*N%* = **FCHECK(***file-number***)**

【註解】*file-number*　　是一個整數的運算式，其值為檔案編號(file number)。

如果此檔案編號關聯到的檔案已被成功地開啟，將傳回 true (1)，否則會傳回 false (0)。

【範例】OPEN 1， “TEST.TXT”

IF (! FCHECK(1)) => GOTO Error:

FILE$() 函數

【功能】FILE$()可以將一份檔案的全部內容讀入，並轉為一個字串

【版本】4.0 版(含)以上

【格式】M$ = **FILE$(***filename***)**

【註解】*filename* - 為一字串運算式，表示要讀入的檔案路徑。檔案內的所有字元(包括換行字元的控制字元)均會被填入字串。

【範例】M$ = FILE$("C:\\PROJ1\\Log.txt")

FLEN() 函數

【功能】FLEN 傳回輸入引數所指定的檔案之大小。

【版本】1.0 版(含)以上

【格式】*N%* = **FLEN(***file-number***)**

【註解】*file-number*　是一個整數運算式，其值為一檔案編號。
如果此檔案編號關聯到的檔案已被成功地開啟，此函數會傳回該檔案的長度大小。如果該檔案是一個通訊裝置檔，此函數會傳回目前接收佇列(receive-queue)中的字元數。

【範例】OPEN 1, "TEST.TXT"

```
N% = FLEN(1)
CLOSE 1
```

FMBCD () 函數

【功能】FMBCD 將 BCD 編碼的輸入引數轉為整數後傳回。

【版本】2.0 版(含)以上

【格式】*N%* = **FMBCD(***numeric-expression***)**

【註解】*numeric-expression* 是一個整數運算式，其值為一 BCD 編碼的整數。
此函數主要應用於輸入引數為一 BCD 編碼的整數,例如來自某 I/O 裝置驅動程式掃瞄自 PLC 暫存器得來的 Tag 資料，為便於圖控的顯示或計算，必須利用本函數將之轉為標準的整數型態。

【範例】Y = FMBCD(0x9876)　　// Y 值為 987

FMDBL () 函數

【功能】FMDBL 將兩個分別代表一雙精度浮點數的 4 個 word 的輸入引數轉為一個雙精度浮點數後傳回。

【版本】2.0 版(含)以上

【格式】*Y* = **FMDBL(***num-expression1, num-expression2, num-expression3,num-expression4***)**

【註解】*num-expression1* 是一個整數運算式，其值代表一雙精度浮點數的第一個 word 資料。
num-expression2 是一個整數運算式，其值代表一雙精度浮點數的第二個 word 資料。
num-expression3 是一個整數運算式，其值代表一雙精度浮點數的第三個 word 資料。
num-expression4 是一個整數運算式，其值代表一雙精度浮點數的第四個 word 資料。
此函數主要應用於輸入引數為四個分別代表一雙精度浮點數的四個 word 的整數，例如由 I/O 裝置驅動程式掃瞄自 PLC 暫存器得來的 Tag 資料，為便於圖控的顯示或計算，必須利用本函數將之轉為標準的雙精度浮點數型態(8 個位元組)。

【範例】Y = FMDBL((0x0000, 0x0000, 0x8000,0x4024))　　// Y 值為 10.25

FMFLT () 函數

【功能】FMFLT 將兩個分別代表一浮點數的 lower word 與 higher word 的輸入引數轉為一個浮點數後傳回。

【版本】2.0 版(含)以上

【格式】*Y* = **FMFLT(***numeric-expression1, numeric-expression2***)**

【註解】*numeric-expression1* 是一個整數運算式，其值代表一浮點數的 lower word 資料。
numeric-expression2 是一個整數運算式，其值代表一浮點數的 higher word 資料。
此函數主要應用於輸入引數為兩個分別代表一浮點數的 lower word 與 higher word 的整數，例如由 I/O 裝置驅動程式掃瞄自 PLC 暫存器得來的 Tag 資料，為便於圖控的顯示或計算，必須利用本函數將之轉為標準的浮點數型態(4 個位元組)。

【範例】Y = FMFLT(0x0000, 0x4124) // Y 值為 10.25

FOR … LOOP 敘述

【功能】FOR 迴圈內包含一連串的指令，此敘述可依指定的次數重覆地執行這些指令。

【版本】1.0 版(含)以上

【格式】**FOR** *variable* **=** *init-expression* **TO** *final-expression* **[STEP** *inc-expression***]**

…

LOOP

【註解】*variable* 用作計數器的變數

init-expression 是一個數值運算式，它是計數器的初始值。

final-expression 是一個數值運算式，它是計數器最後的數值。

inc-expression 用做計數器增加值的運算式（預設值是 1）。

如果計數器的值小於或等於 *final-expression* 的值，那麼 ***SmartScript*** 模組將會繼續執行 FOR 敘述之後的指令。

程式進入迴圈後，將執行 FOR 之後的程式敘述，當到達 LOOP 時，計數器將增加一個 STEP 值(*inc-expression*)。如果此時計數器的值仍然小於等於 *final-expression* 之值，***SmartScript*** 模組的程式流程將再回到 FOR 敘述， 上述的程序加會再度重覆。

如果計數器的值已大於 *final-expression* 之值，那麼 ***SmartScript*** 模組的程式流程將跳到 LOOP 敘述之後的下一個敘述。

如果 *inc-expression* 的值是負的，則做相反的測試。計數器會隨著每次迴圈的執行減少，直到計數器小於 *final-expression* 之值。

巢狀迴圈

FOR … LOOP 可以是巢狀迴圈，換句話說，FOR … LOOP 可以被放在另一個 FOR … LOOP 之中。使用巢狀迴圈時， 每個迴圈必須有獨立的計數器變數。每一個 LOOP 則是對應到離它最近的 FOR。

【範例】
```
DIM A(3， 4)
  FOR I = 1 TO 3
    FOR J = 1 TO 4
      A(I， J) = (I – 1) * 4 + (J – 1)
    LOOP
  LOOP
```

Format$() 函數

【功能】Format 將輸入引數中的數值，依輸入引數中的格式字串要求進行格式化，並將格式化後的字串傳回。

【版本】2.0 版(含)以上

【格式】*N$* = **FORMAT$(***string-expression, numeric-expression***)**

【註解】*string- expression* 字串內容必須是一個合法的格式定義字串，詳下說明

numeric-expression 可以是任何數值運算式。

格式定義字串由以下欄位組成，其中方括弧中為選用欄位，可予省略：

%[flags] [width] [**.**precision] type

格式定義字串中的每一個欄位為一個單一字元或數字，用以指定某一特定的格式選項。最簡單的格式定義字串只包含一個百分號(%)與一個 *type* (資料型態)字元，例如%d。出現在 *type* 字元之前的選用欄位，則用以控制其他格式項目，茲說明如下：

type

type 字元是**格式定義字串中**唯一的必要欄位，它出現在所有選用格式字元之後。*type* 字元指定輸入引數的資料型態，其內容說明如下：

字元	資料型態	輸出格式
d	**int(整數)**	十進位有號整數
i	**int (整數)**	十進位有號整數
o	**int (整數)**	八進位無號整數
u	**int (整數)**	十進位無號整數
x	**int(整數)**	十六進位無號整數,使用“abcdef.”表示
X	**int(整數)**	十六進位無號整數,使用“ABCDEF.” 表示
e	**double(雙精度浮點數)**	以[–]*d.dddd* **e** [*sign*]*ddd* 格式表式的有號數，其中 *d* 為一位數的十進位數字，*dddd* 為一位或多位數的十進位數字，*ddd* 為剛好三位數的十進位數字 *sign* 則為 + 或 –
E	**double(雙精度浮點數)**	除了用大寫 E 取代小寫 **e** 來代表指數外，與 e 格式相同

字元	資料型態	輸出格式
f	**double(雙精度浮點數)**	以[–]*dddd.dddd* 格式表式的有號數，其中 *dddd* 為一位或多位數的十進位數字，小數點前的位數視該數值的大小而定，小數點後的位數則視所需的精度而定
g	**double(雙精度浮點數)**	以 **f** 或 **e** 格式輸出的有號數，取何種格式輸出則視該數值在要求的精度下以何者表示較為精簡。**e** 格式僅在該數值的指數部份小於-4，或大於等於精度指定的位數時採使用。數字後多餘的 0 將被捨去，小數點則只有在有小數部份的數字時才會輸出。
G	**double(雙精度浮點數)**	除了用大寫 E 取代小寫 **e** 來代表指數外，與 g 格式相同

flags

flag (旗號)字元是**格式**定義**字串中**的第一個選用欄位。flag 指令是用來指定輸出符號、空白、小數點以及八進位與十六進位字首的對齊方式的單一字元。**格式定義字串中可包含一個以上的** flag 指令。

Flag	意義	預設值
–	在指定欄寬中將運算結果向左對齊	向右對齊
+	如果輸出值屬於有號的資料型態，在輸出值前加上正負符號(+ or –)	只在負數前顯示負號(–).
0	如果欄寬設定前加 0 旗號做為字首，則在小數數字後補 0 直至達到最小欄寬。如同同時設定 0 旗號和 – 旗號，則 **0** 旗號將被忽略；如果 **0** 旗號與整數格式(**i**, **u**, **x**, **X**, **o**, **d**)同時指定，則 **0** 旗號將被忽略。	不補 0.
空白 (' ')	若輸出值為有號正數，在輸出值前補空白。如果同時設定空白旗號與 + 旗號令，則空白旗號將被忽略。	不補空白
#	當與 **o**、 **x**、或 X 格式同時使用時，#旗號則依資料型態決定在非為 0 的輸出值前加 0、0x 或 0X 字首	不補空白
	當與 **e**、**E** 或 **f** 格式同時使用時，#旗號強制輸出值永遠包含小數點	僅當有小數位數時才顯示小數點
	當與 **g** 或 **G** 格式同時使用時，#旗號強制輸出值永遠包含小數點，並防止小數數字後多餘的 0 被捨去 當與 **d**、 **I** 或 **u** 格式同時使用時，忽略此旗號	僅當有小數位數時才顯示小數點，並刪除小數數字後多餘的 0

width

width(欄寬)是**格式定義字串中**的第二個選用欄位。*width* 是一個不可為負值的十進位整數，用來控制輸出的最小字元數。如果輸出值的字元數小於指定的欄寬，則依據 - 旗號(表向左對齊)之指定與否，在數字前或後補空白以達到指定的最小欄寬。如果 *width* 前加字首 0，則補

0 直到達到最小欄寬(指定向左對齊時無效)。

欄寬設定不會導致任何數字被捨去。如果輸出值的字元數大於指定的欄寬，則欄寬的設定將被忽略，該數值的所有字元均將被輸出(仍須視精度的定義而定)。

precision

precision (精度)是**格式定義字串中**的第三個選用欄位。它是一個不可為負值的十進位整數，前面有一句點(**.**) ，用來指定輸出的字元數、小數點位數以及有效位數(見下表)。與欄寬不同的是精度設定會導致輸出值部份位數的捨去或實數的四捨五入。如果 *precision* 被設為 0，且輸出值亦為 0，則將不輸出任何字元：

【範例】Y$ = FORMAT$("%.0d", 0)　　// Y **值**無輸出字元

資料型態(*type*)的設定將決定 *precision* 的詮釋方式，以及省略 *precision* 時的預設值，詳下表：

資料型態(*type*)	**意義**	**預設值**
d, i, u, o, x, X	精度指定輸出的最小位數，如果輸入引數的位數小於 *precision*，輸出值將在左邊補 0。如果數值的位數超過 *precision* ，多出的位數不會被捨去。	精度預設值為 1.
e, E	精度指定小數點以後的位數，最後一位小數將被四捨五入	精度預設值為 6;如果精度值設為 0 或句點後沒有數字，則不輸出小數點及以下的小數數字
f	精度指定小數點以後的位數，如果含有小數點，其前至少有一位數，數值則依指定小數位數做四捨五入	精度預設值為 6;如果精度值設為 0 或句點後沒有數字，則不輸出小數點及以下的小數數字
g, G	精度指定輸出最大的有效位數	6 位有效數字，並捨去後面任何多餘的 0

FPOS() 函數

【功能】FPOS 傳回指定檔案之讀寫指標(read-write pointer)所在的位置。

【版本】1.0 版(含)以上

【格式】*N%* = **FPOS(***file-number***)**

【註解】*file-number* 是一個整數的運算式，其值為一檔案編號(file number)
若此檔案編號所關聯的檔案已被成功地打開，此函數將傳回該檔案目前讀寫指標(read-write pointer)所在的位置。如果這個檔案是一個通訊裝置檔，此函數則傳回傳送佇列(transmit-queue)中的字元數。

【範例】OPEN 1, “TEST.TXT”
N% = FPOS(1) // N% 值為 0
CLOSE 1

FPRINT 敘述

【功能】FPRINT 將資料寫入指定的檔案

【版本】1.0 版(含)以上

【格式】**FPRINT** *file-number*，*expression*， …，*expression*

【註解】*file-number* 是一個整數運算式，其值為一檔案編號。
expression 可以是任意數值或字串運算式，它會被寫到檔案中。
所有的數值均會先依自由格式(free format)的方式轉為 ASCII 字串，然後再寫入檔案中，各運算式間應該以逗號來做分隔，而運算式間不得加入任何空格。

【範例】A = PI
CREATE 1, “TEST.TXT”
FPRINT 1, “A = “, A, “\r\n”
CLOSE 1

GOSUB 敘述

【功能】GOSUB 會將程式流程導至指定的副程式(Subroutine)去執行。

【版本】1.0 版(含)以上

【格式】**GOSUB** *line-label*

【註解】*line-label*　　副程式的第一行。

副程式必須以 RETURN 敘述來結束，結述後 ***SmartScript*** 模組會將程式流程重新回到最靠進 GOSUB 敘述的下一個敘述去繼續執行。程式中，一個副程式可以被呼叫任意多次，而且它也可以被其副程式呼叫。這種子程式的巢狀呼叫只受到可用記憶體大小的限制。

副程式可以出現在程式中的任何一個地方，但是建議最好讓副程式與主程式有明顯的區分。為了防止不小心進入副程式，通常可以在副程式之前加 STOP、END 或 GOTO 敘述，讓程式在執行時能繞過副程式。

【範例】GOSUB ShowMsg:

```
MESSAGE "Test Program", "In main program"
END
ShowMsg:
  MESSAGE "Test Program", "In subroutine"
  RETURN
```

GOTO 敘述

【功能】GOTO 會將程式流程無條件地導引至指定的行標。

【版本】1.0 版(含)以上

【格式】**GOTO** *line-label*

【註解】*line-label*　　程式中的有效行標。

如果行標所在的敘述是一個可執行的敘述，該敘述以及接續在它之後的敘述將被執行。如果它是個不可執行的敘述，則由位於行標之後的第一個可執行敘述繼續往下執行。

【範例】GOTO ErrHandle:

HOUR() 函數

【功能】HOUR 傳回目前系統時間的小時部份。

【版本】1.0 版(含)以上

【格式】*N%* = **HOUR()**

【註解】函數傳回目前系統時間中，由午夜起算的小時部份(範圍為 0~23 的整數)。

【範例】N% = HOUR()

IDLE 敘述

【功能】IDLE 會使整個程式處於休眠的狀態，但是能處理事件的變化

【版本】4.0 版(含)以上

【格式】**IDLE**

【註解】IDLE 敘述通常用於等待事件發生，相當於執行一空的無窮迴圈(例如 WHILE(1)……..LOOP)，但不同於空迴圈，IDLE 不會消耗 CPU 資源。

【範例】IDLE

```
{EVENT}:
{Value} = {Value} + 1              //每當{EVENT}變化時{Value}會增加 1。
RETURN
```

IF … ELSEIF … ELSE … ENDIF 敘述

【功能】IF 依據某個運算式來決定程式流程的方向。

【版本】1.0 版(含)以上

【格式】**IF (***cond-expression***) =>** *statement*

IF (*cond-expression***)**

…

[ELSEIF (*cond-expression***)]**

…

[ELSE]

…

ENDIF

【註解】*statement* 可以是任何一個敘述

cond-expression 可以是任何一個敘述

如果 *cond-expression* 的結果是 true (非 0)，則執行位於 “then”符號之後或緊接在 IF 之後的敘述。如果 *cond-expression* 結果是 false (0)，在 IF 後面的敘述會被忽略，而將執行 ELSEIF 敘述(如果存在的話)。與 IF 敘述相同，如果 *cond-expression* 的結果是 true，則位在 ELSEIF 之後的敘述會被執行；否則，下一個 ELSEIF 敘述(如果存在的話)將被執行，以此類推。最後，如果沒有任何一個 ELSEIF 是 true，則執行 ELSE 之後的第一個敘述(如果存在的話)。 如果沒有 ELSE，則執行 ENDIF 之後的敘述。

IF … ENDIF 敘述也可以是巢狀的，每個 ENDIF 均會對應到離它最近的 IF。

【範例】
```
IF ((A >= 0) & (A <= 10))
   M$ = “OK”
ELSEIF (A > 10)
   M$ = “> 10”
ELSE
   M$ = “< 0”
ENDIF
IF (A < 0) => GOTO TheEnd:
```

INT() 函數

【功能】INT 會將輸入引數的小數部份捨去，只傳回其整數部份。

【版本】1.0 版(含)以上

【格式】*N%* = **INT(***numeric- expression***)**

【註解】*numeric-expression* 可以是任意數值運算式。

【範例】N% = INT(99.9) // N%值為 99
N% = INT(-4.8) // N%值為-4

ISTR$() 函數

【功能】ISTR$依指定的進位法將一個數值轉換成一個字串。

【版本】1.0 版(含)以上

【格式】*M$* = **ISTR$(***numeric-expression* **，** *radix-expression***)**

【註解】*numeric- expression* 可以是任何數值運算式。
radix-expression 是一個整數運算式，其值用以指定進位法。有效的進位法是 2 進位法到 36 進位法。
此函數先將 *numeric-expression* 的結果轉變為一個無號長整數，然後以指定的進位法將它轉變為一個字串。

【範例】M$ = ISTR$(15, 2) // M$值為 “1111”，即以二進位法來表示 15 所構成的字串

IVAL() 函數

【功能】IVAL 用指定的進位法將一個字串轉變為一數值。

【版本】1.0 版(含)以上

【格式】*Y* = **IVAL(***string-expression* **,** *radix-expression***)**

【註解】*string-expression* 可以是任何字串運算式

radix-expression 是一個整數運算式，其值代表一種進位法，有效的進位法是 2 進位到 36 進位。

此函數以指定的進位法將 *string-expression* 的結果轉為一個整數，如果無法轉換該字串的話，則 IVAL 將傳回 0。

【範例】A = IVAL(“FFFF”,16) // A 值為 65535

LEFT$() 函數

【功能】LEFT$傳回一個由輸入字串的前 N 個字母組成的新字串。

【版本】1.0 版(含)以上

【格式】*M$* = **LEFT$(***string-expression* **,** *count-expression***)**

【註解】*string-expression* 可以是任何字串運算式。

count- expression 是一個整數運算式，其值代表要擷取出的字元數目。有效的數目由 0 至原字串的最大長度。

【範例】M$ = LEFT$(“ABCDEFG”, 3) // M$值為“ABC”

LEN() 函數

【功能】LEN 傳回輸入字串所包含的字元數。

【版本】1.0 版(含)以上

【格式】*N%* = **LEN(***string-expression***)**

【註解】*string-expression* 可以是任何字串運算式。

【範例】N% = LEN(“ABCDEFG”) // N%值為 7

LN() 函數

【功能】LN 傳回輸入數值的自然對數值。

【版本】1.0 版(含)以上

【格式】*Y* = **LN(***numeric- expression***)**

【註解】*numeric- expression* 是一個數值運算式，而且必須大於零。

此函數相等於代數上的運算式 $y = \ln x = \log_e x$。自然對數即是以 *e* 為底的對數。

【範例】Y = LN(2) // Y 值為 0.693147

LOG() 函數

【功能】LOG 傳回輸入數值的的對數值。

【版本】1.0 版(含)以上

【格式】*Y* = **LOG(***numeric- expression***)**

【註解】*numeric- expression* 是一個數值運算式，而且必須大於零。

此函數相等於代數上的運算式 $y = \log x = \log_{10} x$ 。此對數是以 10 為底的對數。

【範例】Y = LOG(10) // Y 值為 1

LOWER$() 函數

【功能】LOWER$傳回與輸入字串相同的字串，但是所有的字母都會被轉為小寫。

【版本】1.0 版(含)以上

【格式】*M$* = **LOWER$(***string-expression***)**

【註解】*string-expression* 可以是任何字串運算式。

【範例】M$ = LOWER$("ABCDefg") // M$ 值為 “abcdefg”

LTRIM$() 函數

【功能】LTRIM$傳回與輸入字串相同的字串，但移除所有的前置空白(Space，ASCII 碼=32)、跳格(Tab，ASCII 碼=9)、Carriage Return(ASCII 碼=13)及 Line Feed(ASCII 碼=10)等控制字元。

【版本】1.0 版(含)以上

【格式】*M$* = **LTRIM$(***string-expression***)**

【註解】*string-expression* 可以是任何字串運算式。

【範例】M$ = LTRIM$(" ABCDEFG") // M$值為“ABCDEFG”

MAX() 函數

【功能】MAX 比較兩個值，並傳回較大的一個。

【版本】1.0 版(含)以上

【格式】*Y* = **MAX(***numeric- expression* **,** *numeric- expression***)**

【註解】*numeric- expression* 可以是任何數值運算式。

【範例】Y = MAX(2.15,4.3) // Y 值為 4.3

MD 敘述

【功能】MD 用以建立一個新資料夾。

【版本】4.0 版(含)以上

【格式】**MD** *directory-name*

【註解】*directory-name* 是一個字串的運算式，用來描述所要建立的資料夾路徑

directory-name 可以包含多層的子資料夾，若中間各層的子資料夾不存在，MD 敘述會一併建立指定路徑中所需的各層子資料夾。

若因檔案存取權限等因素造成資料夾無法建立，則 MD 敘述將失敗。敘述失敗僅會導致控制程序模組將之計為一次內部錯誤(可由 ERRID()得知錯誤碼，詳 ERRID()及 NERR())，但程式流程則將繼續。

【範例】MD “C:\\NewFolder”

MESSAGE 敘述

【功能】MESSAGE 畫面上顯示一個訊息方塊。

【版本】1.0 版(含)以上

【格式】**MESSAGE** *title-expression, text-expression*

【註解】*title- expression* 是一個字串運算式，用作對話方塊的標題。

text- expression 是一個字串運算式，即訊息方塊中所要顯示的訊息。

訊息方塊包括一個代表應用目的的標題和以及訊息本身， 再加上一個標示小寫字母「*i* 」的圓形圖示以及一個「確定」按鈕。Windows 系統並不自動將訊息拆行來容納完整的訊息，所以訊息字串中必須包含換行字元，好讓較長的訊息能在適當位置換行。

【範例】MESSAGE “測試程式”, “這是第一行\n 這是第二行”

MID$() 函數

【功能】MID$傳回輸入字串的指定部份。

【版本】1.0 版(含)以上

【格式】*M$* = **MID$(***string-expression* **,** *index-expression* **,** *count-expression***)**

【註解】*string-expression* 可以是任何字串運算式。

index-運算式 是一個整數運算式，其值應為一有效的索引值。有效的索引值由 1 到字串的最大長度。

count-運算式 是一個整數運算式，其值應為一有效的索引值。有效的索引值由 0 到字串的最大長度。

【範例】M$ = MID$(“ABCDEFG”, 3, 2) // M$ 值為“CD”

MIN() 函數

【功能】MIN 比較兩個值，並且傳回較小的值。

【版本】1.0 版(含)以上

【格式】*Y* = **MIN(***numeric-expression* **,** *numeric-expression***)**

【註解】*numeric-expression* 可以是任何數值運算式

【範例】Y = MIN(2.15 , 4.3) // Y 值為 2.15

MINUTE() 函數

【功能】MINUTE 傳回目前系統時間的分鐘部份。

【版本】1.0 版(含)以上

【格式】*N%* = **MINUTE()**

【註解】此函數傳回了目前系統時間的分鐘部份(0 ~ 59) 。

【範例】N% = MINUTE()

MONTH() 函數

【功能】MONTH 傳回目前的月份。

【版本】1.0 版(含)以上

【格式】*N%* = **MONTH()**

【註解】此函數傳回目前的月份(1~12) 。

【範例】N% = MONTH()

MOVE 敘述

【功能】MOVE 可移動一檔案或資料夾。

【版本】4.0 版(含)以上

【格式】**MOVE** *source-path,target-path*

【註解】*source-path* 是一個字串的運算式，用來描述來源檔案或資料夾路徑

target-path 是一個字串運算式，用以描述目的檔案或資料夾路徑。

移動檔案或資料夾時，若來源檔案或資料夾不存在，目標路徑不存在，或因其他原因導致無法成成功移動檔案，則 MOVE 敘述失敗，但敘述失敗僅會被控制程序模組計為一次內部錯誤(可由 ERRID()得知錯誤碼，詳 ERRID()與 NERR())，程式流程則將繼續。

移動檔案時，若目標檔案已存在，MOVE 敘述將會失敗。請注意此特性與 COPY 敘述不同。

【範例】MOVE “C:\\Folder1\\Log1.txt”,” C:\\ Folder1\\Log2.txt”

MSGBOARD 敘述

【功能】在畫面上開啟一「重要訊息」視窗並顯示一行訊息文字。

【版本】4.0 版(含)以上

【格式】**MSGBOARD** *text-expression*

【註解】*text- expression* 是一個字串運算式，即要顯示的一行訊息。

執行本指令將開啟「重要訊息」視窗以顯示一行指定的文字訊息，訊息前會自動加註顯示訊息的日期時間與[CslMan32]的分類標簽。顯示訊息後，SmartScript 並不會等待使用者關閉「重要訊息」視窗即繼續執行後續的指令。若執行本指令時，「重要訊息」視窗已開啟而未被使用者關閉，新的訊息會被附加在前一個訊息的下方。

「重要訊息」視窗下方有兩個按鈕，按下「儲存且關閉」按鈕，會將視窗中所顯示的所有訊息存入系統程式資料夾(預設為 c:\lablink\system4)中的文字檔 MsgBoard.log 後，將訊息予以清除並關閉視窗；若按下「不存即關閉」按鈕，將直接關閉「重要訊息」視窗，所有顯示內容亦將被清除。每一次執行本指令僅顯示一行訊息，系統將自動濾除換行字元。

【範例】MSGBOARD “這是測試訊息”

NERR() 函數

【功能】NERR()可以取得目前累積的錯誤次數

【版本】4.0 版(含)以上

【格式】N% = **NERR()**

【註解】NERR()將回傳目前累積的錯誤次數，控制程序模組的敘述或函數執行時若發生錯誤，將使本函數的回傳值加 1，RSTERR 敘述可清除本函數回傳的累積的錯誤次數。

【範例】N% = NERR() //N%為正整數

NOW() 函數

【功能】NOW()可以取得以一實數表達之目前的系統日期與時間

【版本】4.0.0.10 版(含)以上

【格式】N = **NOW()**

【註解】NOW()將回傳目前的系統日期與時間。此日期與時間將以一實數來表達，其中整數部份代表日期，小數部份代表以天為單位的時間。此函數所回傳的數值可以直接設定給 TAG 變數來改變該 TAG 的日期與時間欄位。

【範例】{Tag1.t} = NOW()　　//將 Tag1 的日期時間設定為目前的系統日期時間

NOW$() 函數

【功能】NOW$()可以取得以一字串表達之目前的系統日期與時間

【版本】4.0.0.10 版(含)以上

【格式】M$ = **NOW$()**

【註解】NOW$()將回傳目前的系統日期與時間。此日期與時間將以一字串來表達，其格式為"*yyyy/mm/dd hh:ii:ss.sss*"，其中 *yyyy* 為四位數的西元年份，*mm* 為兩位數的月份，*dd* 為兩位數的日期，*hh* 為為兩位數的小時，*ii* 為兩位數的分鐘，*ss.sss* 的整數部份為兩位數的秒鐘，小數部份則為三位數的毫秒，整個字串的長度固定為 23 個字元。

【範例】M$ = NOW$()　　//M$之內容為目前的系統日期時間，
　　//例如"2010/01/01 12:00.00.000"

OPEN 敘述

【功能】OPEN 打開指定的檔案。

【版本】1.0 版(含)以上

【格式】**OPEN** *file-number* **,** *file-name* **,** *access-type*

【註解】*file-number*　是一個整數運算式，其值應為一有效的檔案編號。只要檔案未被關閉，而且有其他的檔案 I/O 敘述在使用他，此檔案編號即被關聯到 *file-name* 所指定的檔案。有效的檔案編號是 1 到 16。

file-name 是一個字串運算式，用以指定將被打開的檔案名稱。

access-type　是一個字串運算式，用以指定對檔案的存取模式。此參數可以是下列關鍵字之一：

"R"　對檔案有唯讀存取權，只能將資料由檔案中讀出。

"W" 對檔案有唯寫存取權，只能將資料寫入檔案中。

"RW"對檔案有讀寫存取權，可以從檔案中將資料讀出，也可以寫入。

當此敘述打開檔案後，讀寫指標(read-write pointer)會被設到檔案的開頭。

【範例】OPEN 1 , “TEST.TXT” , “R”

PASS 敘述

【功能】PASS 用於打開指定的可執行檔或文件檔，與 EXEC 相同，但使用 PASS 敘述時，控制程序模組會等待 PASS 敘述所引發執行的程式結束，才繼續執行後續的程式敘述。

【版本】4.0 版(含)以上

【格式】**PASS "***filename***","***parameter***","***showmode***"**

【註解】 *file-name* **檔案路徑及名稱。**必須是一個字串運算式，用以指定要開啟的檔案或資料夾。此敘述可開啟可執行檔或文件檔。

parameter **參數。**必須是一個字串運算式， 用以指定要傳給應用程式的參數(如果 *file-name* 指定的是可執行檔)。 如果 *file-name* 指定的是文件檔或不需要參數的執行檔，*parameter* 應為空字串。

show-mode **顯示模式。**指定開啟檔案的應用程式之顯示模式。這個參數可以是以下列出的字串之一：

"MIN" 使視窗最小化，且為非使用中。

"MAX" 使視窗最大化。

"HIDE" 隱藏此視窗。

"ICON"將使視窗最小化，但仍為使用中。

"NORMAL" 以該視窗的正常大小與位置顯示，且為使用中。

【範例】PASS "NOTEPAD.EXE","", "NORMAL"

MESSAGE "","NOTEPAD CLOSED"

上述 PASS 敘述將執行「記筆本」程式，但須關閉「記筆本」後，MESSAGE 敘述才會被執行。

PLAY 敘述

【功能】PLAY 依指定檔名播放聲音檔。

【版本】1.0 版(含)以上

【格式】**PLAY** *file-name* **,** *play-mode*

【註解】*file-name*　　是一個字串運算式，用以指定要播放的 wave 檔的檔名

play-mode　　是一個字串運算式，用以指定播放 wave 檔的模式。此參數可以是下列關鍵字之一：

""　停止目前進行中的聲音檔播放，並且播放 *file-name* 指定的聲音檔。

"LOOP"　停止目前進行中的聲音檔播放，並且播放 *file-name* 指定的聲音檔。

"NOSTOP"　播放 *file-name* 指定的聲音檔，但是不停止目前進行中的聲音檔播放。如果由於產生聲音所需的系統資源正忙於播放其他的聲音，此敘述將立即中止，回到原先的程式流程而不播放任何聲音。

聲音並非同步播放，同時 PLAY 在開始聲音播送後即立刻返回原先的程式流程。要終止一個進行中的 wave 聲音播放，呼叫 PLAY，並將 *file-name* 設為空字串。

The sound specified by *file-name* 所指定的聲音檔長度，必須小於可用的實體記憶體容量，同時須先安裝 waveform-audio devicedriver 方可播放。

【範例】PLAY “Siren-01.wav” , “LOOP”

RAND() 函數

【功能】傳回一個亂數。

【版本】1.0 版(含)以上

【格式】*Y* = **RAND(***numeric-expression***)**

【註解】*numeric- expression* 可以是任何數值運算式。

此函數傳回一隨機數字，其值介於 0 與 *numeric- expression* 的結果之間。隨機數值產生器是 auto re-seeded，所以每次執行程式均會產生不同亂數。

【範例】Y = RAND(5.5)

RAWVAL() 函數

【功能】將字串依指定編碼方式轉換為數值。

【版本】4.0.2.3 版(含)以上

【格式】*Y* = **RAWVAL(***string-expression, string-constant***)**

【註解】*string-expression* 代表一以特定編碼方式表達的數值字串。

string-constant 用來指定編碼方式的字串常數，必須是以下字串之一：

"INT16"	表字串為 16 位元的二進制編碼整數
"BCD16"	表字串為 16 位元的 BCD 編碼無號整數
"SBCD16"	表字串為 16 位元的 BCD 編碼有號整數
"INT32"	表字串為 32 位元的二進制編碼整數
"BCD32"	表字串為 32 位元的 BCD 編碼無號整數
"SBCD32"	表字串為 32 位元的 BCD 編碼有號整數
"FLOAT"	表字串為二進制編碼的單精準度浮點數
"DOUBLE"	表字串為二進制編碼的雙精準度浮點數

此函數傳回一整數或浮點數，其值為將 *string-expression* 依 *string-constant* 指定的編碼方式解碼後所得到的數值。

【範例】N% = RAWVAL("\x10\x00\x00\x00", "INT32")

N%=16

RD 敘述

【功能】RD 用以刪除一個空資料夾。

【版本】4.0 版(含)以上

【格式】**RD** *directory-name*

【註解】*directory-name* 是一個字串的運算式，用來描述所要刪除的資料夾路徑

若指定的資料夾不存在，或雖存在但內仍有檔案或子資料夾，則 RD 敘述失敗。但敘述失敗僅會被控制程序模組計為一次內部錯誤(可由 ERRID()得知錯誤碼，詳 ERRID()與 NERR())，程式流程則將繼續。

【範例】RD “C:\\EmptyFolder”

READ 敘述

【功能】READ 從指定檔案中讀取數據。

【版本】1.0 版(含)以上

【格式】**READ** *file-number*, *string-variable*, *count-expression*

【註解】*file-number* 是一個整數運算式，其值為一檔案編號。

string-variable 一個用來從檔案中接收數據資料的字串變數。

count-expression 是一個整數運算式，其值為一計數值，用來指定將由檔案中讀出的位元組數目。

如果與指定的檔案編號關聯到的檔案已被成功地打開，此敘述會從檔中讀取數據資料。如果指定的檔案是一個通訊裝置檔，此敘述則將從接收佇列(receive-queue)中讀取數據資料。

【範例】READ 1 , A$, 80

RETURN 敘述

【功能】RETURN 從副程式返回原先的程式流程。

【版本】1.0 版(含)以上

【格式】**RETURN**

【註解】副程式中的 RETURN 敘述將使 ***SmartScript*** 模組的程式流程跳回此次 GOSUB 敘述之後的下一個敘述。如果程式邏輯需要在副程式的不同部份均需有 RETURN 的動作，副程式中可以包含一個以上的 RETURN 敘述。副程式可以出現在程式中的任何一個地方，但是建議最好讓副程式與主程式有明確的區分。為避免不小心進入副程式，可利用 STOP，END 或 GOTO 敘述來讓主程式的執行流程繞過副程式。

【範例】GOSUB ShowMsg:

```
MESSAGE "Test Program" , "In main program"
END

ShowMsg:
  MESSAGE "Test Program" , "In subroutine"
  RETURN
```

RIGHT$() 函數

【功能】RIGHT$傳回輸入字串最右邊 N 個字元所組成的字串。

【版本】1.0 版(含)以上

【格式】*M$* = **RIGHT$(**string-expression , count-expression**)**

【註解】*string-expression*　可以是任何字串運算式

count-expression　是一個整數運算式，其值被用做一個計數器。有效的計數器值是 0 到字串的最大長度。

【範例】M$ = RIGHT$(“ABCDEFG” , 4)　　// M$值為“DEFG”

RSTERR 敘述

【功能】RSTERR 用以清除控制程序模組的內部錯誤計數器

【版本】4.0 版(含)以上

【格式】**RSTERR**

【註解】控制程序模組內部有一錯誤計數器用來累積錯誤次數 , NERR()即可傳回目前錯誤計數器之累計次數值。執行本敘述將會將錯誤計數器之累計值歸零 , 因此執行 RSTERR 敘述後 , NERR()將傳回 0。

【範例】RSTERR　　// NERR() = 0。

RTRIM$() 函數

【功能】RTRIM$傳回與輸入字串相同的字串，但是移除了字串末端的所有空白(Space，ASCII 碼=32)、跳格(Tab，ASCII 碼=9)、Carriage Return(ASCII 碼=13)及 Line Feed(ASCII 碼=10)等控制字元。

【版本】1.0 版(含)以上

【格式】*M$* = **RTRIM$(***string-expression***)**

【註解】*string-expression*　可以是任何字串運算式。

【範例】M$ = RTRIM$("ABCDEFG　　")　　// M$值為“ABCDEFG”

SECOND() 函數

【功能】SECOND 傳回目前系統時的秒數部份。

【版本】1.0 版(含)以上

【格式】*N%* = **SECOND()**

【註解】此函數傳回目前系統時的秒數部份 (0~59)。

【範例】N% = SECOND()

SEEK 敘述

【功能】SEEK 移動指定檔案的讀寫指標(read-write pointer)。

【版本】1.0 版(含)以上

【格式】**SEEK** *file-number* **,** *count-expression* **,** *seek-mode*

【註解】*file-number*　是一個整數運算式，其值為一檔案編號。

count-expression　是一個整數運算式，代表一個計數值。該計數值代表讀寫指標所要移動的 byte 數。

seek-mode　指定讀寫指標的啟始位置與移動方向。此變數可以是下列值之一：

"B" 從檔案的開頭，移動讀寫指標 *count-expression* 個 bytes。

"C" 從目前所在的位置，移動讀寫指標 *count-expression* 個 bytes。

"E" 從檔案的結尾，移動讀寫指標 *count-expression* 個 bytes。

如果檔案編號所連結的檔案已被成功地打開，此敘述會將檔案的讀寫指標移動至指定的位置。如果檔案是一個通訊裝置檔，此敘述會清除接收佇列(若 *seek-mode* = "C")或傳送佇列(若 *seek-mode* = "B")，或兩者皆清除(若 *seek-mode* = "E")。

當檔案剛被打開時，讀寫指標會被設到檔案的開頭，SEEK 敘述僅會移動讀寫指標，而不讀取檔案中的資料，如此即可達到隨機存取檔案內容的目的。

【範例】SEEK 1 , 0 , “E”

SETDIR 敘述

【功能】SETDIR 可在 Lab-LINK 執行期間設定參考目錄(~0，…，~9)所代表的目錄路徑。

【版本】1.1 版(含)以上

【格式】**SETDIR** *numeric-expression, string-expression*

【註解】*numeric - expression*　是一個數值運算式，其值須在 0~9 之間，用以指定要修改的參考目錄序號。

string - expression　是一個字串運算式，該字串應為一目錄之路徑，用以指定參考目錄變數所代表的目錄所在路徑。

參考目錄係 Lab-LINK 用以代表專案中某些預設目錄的路徑，其目錄由 10 個變數~0，…，~9 來代表。Lab-LINK 在啟動時在已預設其中 8 個變數(~0，…，~7)的，路徑如下表：

變數	預設路徑	說明
~0	..\project\專案名稱\cfg\工作站名稱	專案工作站的執行設定檔
~1	..\project\專案名稱\pnl	存放專案所使用的面板檔
~2	..\project\專案名稱\bmp	存放專案面板所使用的 bmp 圖檔
~3	..\project\專案名稱\wmf	存放專案面板所使用的 wmf 圖檔
~4	..\project\專案名稱\wav	存放專案面板所使用的 wav 聲音檔
~5	..\project\專案名稱\txt	存放專案面板所使用的文字檔
~6	..\project\專案名稱\dat	專案資料檔的預設儲存目錄
~7	..\project\專案名稱\csl	存放專案所使用的控制程序檔
~8	未定義	保留給使用者定義
~9	未定義	保留給使用者定義

SETDIR 可在 Lab-LINK 執行期間改變任一參考目錄變數所代表的目錄路徑。但應注意某些模組在啟動時即已載入目錄變數所代表的路徑，啟動後再以 SETDIR 修改變數所代表的路徑對其將無效，例如存檔模組若用預設存檔路徑時，將採用~6 做為存檔的路徑，系統啟動後即使改變~6 所代表的路徑，亦不會改變資料檔儲存位置。反之若「歷史資料趨勢圖」物件中採用

~6 作為資料檔路徑，則修改~6 之值後，只要觸發「歷史資料趨勢圖」更新資料，即可讓它由新的路徑讀取資料。

設定路徑時建議採用相對路徑，請注意相對路徑的基準為 Lab-LINK 系統目錄(預設安裝目錄為 C:\Lablink4\System4)。

【範例】//將~8 路徑設為 ..\project\proj1\dat\01

SETDIR 8, "..\\PROJECT\\PROJ1\\DAT\\01"

SHORTCUT 敘述

【功能】SHORTCUT 可用來建立一應用程式或文件檔之執行捷徑

【版本】4.0 版(含)以上

【格式】**SHORTCUT** *link-file,filename,parameter,work-directory*

【註解】*link-file* 是一個字串運算式，表示捷徑的檔名。

filename 是一個字串運算式，表示欲產生捷徑的來源檔案，路徑必須完整。

parameter 可以指定捷徑所需要的參數。

work-directory 可以指定來源檔案的工作目錄。

若 *link-file* 所包含的路徑不存在，則 SHORTCUT 敘述將失敗。但敘述失敗僅會被控制程序模組計為一次內部錯誤(可由 ERRID()得知錯誤碼，詳 ERRID()與 NERR())，程式流程則將繼續。

若 *link-file* 所指定的的捷徑已存在，SHORTCUT 敘述將建立新的捷徑來取代同名的舊捷徑。

SHORTCUT 敘述並不會檢查 *filename*、*parameter* 與 *work-directory* 等建立捷徑的參數是否正確，因此雖然 SHORTCUT 敘述已成功建立捷徑，並不保證該捷徑可否正確執行，使用者應自行確認這些參數正確性。

【範例】SHORTCUT “Test.lnk”,”DBSaver.exe”,”PROJ1 WKS1”,” C:\\LabLink\\System4\\”

SHUTDOWN 敘述

【功能】SHUTDOWN 可用來結束圖控系統的執行

【版本】4.0 版(含)以上

【格式】**SHUTDOWN**

【註解】若需由控制程序中結束 Lab-LINK 圖控系統的執行，應執行本敘述。若在控制程序中想利用將系統 Tag「$EXIT」設為 1 的方式來結束圖控，有可能造成部份模組無法結束，因此仍建議以執行本敘述來結束圖控為宜。

【範例】SHUTDOWN　　// 結束圖控系統

SIN() 函數

【功能】SIN 傳回輸入引數的正弦函數值。

【版本】1.0 版(含)以上

【格式】*Y* = **SIN(***numeric- expression***)**

【註解】*numeric- expression*　　必須是一個單位為弧度(radian)的角度值。
此函數等於代數上的運算式 $y = \sin(x)$，若須將角度單位由度(degree)轉為弧(radians)，可將度值乘以(PI / 180)。

【範例】Y = SIN(90 * (PI / 180))　　// Y 值為 1

SINH() 函數

【功能】SINH 傳回輸入引數的反曲正弦函數值。

【版本】1.0 版(含)以上

【格式】*Y* = **SINH(***numeric- expression***)**

【註解】*numeric- expression* 可以是任何數值運算式。

此函數等於代數上的運算式 $y = \sinh(x)$。

【範例】Y = SINH(1) // Y 值為 1.175201

SLEEP 敘述

【功能】SLEEP 可用來讓程式的執行暫停一段指定的時間

【版本】4.0 版(含)以上

【格式】**SLEEP** *numeric expression*

【註解】*numeric expression* 是一個數值運算式，而且必須大於等於零，單位為秒。

【範例】SLEEP 1.5 // 程式暫停 1.5 秒

SQRT() 函數

【功能】SQRT 傳回輸入引數的平方根值。

【版本】1.0 版(含)以上

【格式】*Y* = **SQRT(***numeric- expression***)**

【註解】*numeric- expression* 是一個數值運算式，而且必須大於等於零。

此函數等於代數上的運算式 $y = \sqrt{x}$ 。

【範例】Y = SQRT(2) // Y 值為 1.414214

STOP 敘述

【功能】STOP 終止程式的的執行，並關閉所有的檔案。

【版本】1.0 版(含)以上

【格式】**STOP**

【註解】STOP 敘述可置於程式中的任何位置，以終止程式的執行。

【範例】IF (A > 0) => STOP

STR$() 函數

【功能】STR$傳回所輸入數值轉換成的字串。

【版本】2.0 版(含)以上

【格式】*M$* = **STR$(***numeric- expression***)**

【註解】*numeric- expression* 可以是任何數值運算式。

【範例】M$ = STR$(5.3124) // M$值為“5.3124”

STRING$() 函數

【功能】STRING$會將輸入字串的第一個字元取出，重覆指定次數以產生一個新字串，並將此新字串傳回。

【版本】1.0 版(含)以上

【格式】*M$* = **STRING$(***string-expression***,** *count-expression***)**

【註解】*string-expression* 可以是任何字串運算式，字串運算式中的第一個字元將會被重複。
count-expression 是一個整數運算式，它計算出一個計數值，此計數值用以指定字元重覆的次數。

【範例】M$ = STRING$(“*”, 5) // M$值為“*****”

SUM08() 函數

【功能】SUM08 傳回輸入字串的 8 位元 checksum 值。

【版本】1.0 版(含)以上

【格式】*N%* = **SUM08(***string-expression***)**

【註解】*string-expression*　可以是任何字串運算式。

【範例】N% = SUM08(“\x02ABC\x03”)　　// N%值為 203

SWITCH … CASE … DEFAULT … ENDSW 敘述

【功能】SWITCH 會計算一個運算式的值，以其運算結果與各 case-expression 之值做比較，並執行與相符的 case-expression 關聯的敘述。

【版本】1.0 版(含)以上

【格式】**SWITCH (***switch-expression***)**
[CASE *case- expression***]**
。
。
。
[CASE *case- expression***]**
。
。
。
[DEFAULT]
。
。
。
ENDSW

【註解】*switch- expression* 可以是任何數值運算式。
case- expression 可以是任何數值運算式。
SWITCH 與 CASE 關鍵字會計算 *switch- expression* 的值,將計算結果與各 case-expression 之值做比較，並執行與相符的 case-expression 關連的敘述。
如果無法 *case- expression* 配對，則與 DEFAULT 關鍵字關連的敘述就會被執行；如果未使用 DEFAULT 關鍵字，程式會跳到 ENDSW 之後的敘述繼續執行。

【範例】SWITCH (N%)

```
    CASE 0
      Y = SIN(X)
    CASE 1
      Y = COS(X)
    DEFAULT
      Y = 0
  ENDSW
```

SYSINFO$() 函數

【功能】SYSINFO$傳回目前電腦的系統資訊。

【版本】4.0 版(含)以上

【格式】*S%* = **SYSINFO(***info-name***)**

【註解】*Iinfo-name* 一字串運算式，其值必須是以下關鍵字之一，用以指定要傳回的系統資訊種類

HOSTNAME 傳回電腦名稱

HOSTADDR 傳回電腦的 IP 位址，若 IP 位址多於一組，則回傳的多組 IP 位間以空格字元隔開。

【範例】S$ = SYSINFO$(HOSTNAME”) // N%值為”COMPUTER1”

T$ = SYSINFO$(HOSTADDR”) // N%值為”192.168.100.1”

TAG() 函數

【功能】**TAG** 以輸入引數字串作為 TAG 名稱，可傳回或設定該名稱對應的 TAG 之數值或訊息資料。

【版本】1.0 版(含)以上

【格式】*Y* = **TAG(***string- expression***)** 或

TAG(*string- expression***) =** *expression*

【註解】*string- expression* 字串內容必須是一個合法的 TAG 名稱或 TAG 名稱.$

expression 可以是任何數值或字串運算式，但其資料型別應與。**TAG** 所指定的 TAG 資料欄位相同。

若 *string-expression* 對應的 TAG 名稱不存在，該 TAG 將自動被創造出來，因此其內容必須符合 TAG 命名的原則，字串最後兩個字元若為「.$」, 則代表 TAG 的訊息欄位，而字串最後兩個字元若為「.t」或「.T」, 則代表 TAG 的日期時間欄位。若 *expression* 為一字串運算式，請注意 TAG 的訊息欄位長度有 80 個字元的限制

當此函數位於等號右方時，將傳回字串變數所對應的 TAG 名稱之數值或訊息。

當此函數位於等號左方時, 則會將等號後右方的運算結果, 設給對應 TAG 的數值或訊息欄位。

【範例】TAG("Tag"+"1") = 3.9 // 將名為「Tag1」的 Tag 之數值設為 3.9

TAG("Tag"+"1"+".$") = "OK" // 將名為「Tag1」的 Tag 之訊息設為 OK

{Tag2} = TAG("Tag1") // 將名為「Tag2」的 Tag 之數值設為名為

//「Tag2」的數值等於「Tag1」的 Tag 之數值

TAN() 函數

【功能】TAN 傳回輸入引數的正切函數值。

【版本】1.1 版(含)以上

【格式】*Y* = **TAN(***numeric- expression***)**

【註解】*numeric- expression* 必須是一個以弧度為單位的角度值。
此函數相當於代數上的運算式 $y = \tan(x)$ ，若須將角度單位由度(degree)轉為弧度(radians)，可將度值乘以(PI / 180)。

【範例】Y = TAN(45 * (PI / 180)) // Y 值為 1.0

TANH() 函數

【功能】TANH 傳回輸入引數的的雙曲線正切函數值。

【版本】1.0 版(含)以上

【格式】*Y* = **TANH(***numeric- expression***)**

【註解】*numeric- expression* 可以是任何數值運算式。
此函數相當於代數的運算式 $y = \tanh(x)$

【範例】Y = TANH(1) // Y 值為 0.761594

TICK() 函數

【功能】TICK 傳回自從 Windows 系統啟始後開始計算的毫秒值。

【版本】1.0 版(含)以上

【格式】*Y* = **TICK()**

【註解】傳回的值是自從 Windows 系統啟始後開始計算的毫秒數值，此時間計數是以一個 32 位元的數值來儲存，這表示 Windows 無法記錄超過 2^{32} 毫秒，否則該計數值將會因溢位而歸零。這是大概 49.7 天。如果你使用此計時方式，應該在比較時間時先檢查是否有溢位的狀況。

【範例】Y = TICK()

TIMER() 函數

【功能】TIMER 傳回目前的系統時間，其表達方式為午夜之後所經過的的秒數。

【版本】1.0 版(含)以上

【格式】*N%* = **TIMER()**

【註解】傳回的值是由午夜起算所經過的秒數。

【範例】N% = TIMER()

TOKEN 敘述

【功能】TOKEN 取出輸入字串中分隔字元之前的部份字串做為 token。

【版本】1.0 版(含)以上

【格式】**TOKEN** *token-variable* **,** *source-variable* **,** *delimiters*

【註解】*token-variable*　　一個將會收到 token 的字串變數。

source-variable　　可以是任何字串變數。

delimiters　　是一個含有所有分隔字元的字串運算式。

此敘述尋找來源字串中的第一個分隔字元(由 *delimiters* 指定) , 並據此將來源字串分為兩個字串。前半部字串被分配給 *token-variable*，後半部字串則被分配回 *source-variable*。

【範例】A$ = “Color = 255, 192, 128”

```
TOKEN M$, A$, “=“      // M$值為"Color "      A$值為" 255, 192, 128"
TOKEN M$, A$, “,”      // M$值為" 255"        A$值為" 192, 128"
TOKEN M$, A$, “,”      // M$值為" 192"        A$值為" 128"
TOKEN M$, A$, “,”      // M$值為" 128"        A$值為""
```

TONE 敘述

【功能】TONE 用於驅動 PC 內建喇叭播放一指定頻率與長度的聲音

【版本】4.0 版(含)以上

【格式】**TONE** *frequency,duration*

【註解】*frequency*　　為一數值運算式，用以指定聲音頻率，單位為 Hz。

duration　　為一數值運算式，用以指定播放長度，以秒為單位。

控制程序模組執行 TONE 敘述後，須待聲音播放結束後方會繼續執行後續的程式敘述

【範例】TONE 440,3 // 撥放 440Hz 聲音 3 秒鐘

TRAPOFF 敘述

【功能】TRAPOFF 將停止 TAG 事件的補捉。

【版本】1.0 版(含)以上

【格式】**TRAPOFF**

【註解】此敘述會使 TAG 事件的補捉功能失去作用，而所有 TAG 事件均會被忽略。
此敘述被執行後，系統雖停止執行其他 TAG 事件，但後續發生的 TAG 事件仍會被記錄於一事件佇列中，等 TAG 事件補捉恢復(執行 TRAPON 敘述)後，系統會從事件佇列捉逐一取出未執行的事件來執行。由於佇列有容量限制，請注意 TRAPOFF 的時間不宜太長，否則若積存的 TAG 事件數量大於佇列容量，將造成系統的錯誤。
關於 TAG 事件的使用，請參閱第一章中有關“行標”的說明。

【範例】TRAPOFF

TRAPON 敘述

【功能】TRAPON 啟動 TAG 事件捕捉功能。

【版本】1.0 版(含)以上

【格式】**TRAPON**

【註解】此敘述給予啟動 TAG 事件補捉功能，而所有的 TAG 事件將會被處理。請參閱第一章中“行標”的相關說明。

【範例】TRAPON

UPPER$() 函數

【功能】UPPER$傳回與輸入字串相同的字串，但是所有的字元被轉為大寫。

【版本】1.0 版(含)以上

【格式】*M$* = **UPPER$(***string-expression***)**

【註解】*string-expression*　可以是任何字串運算式。

【範例】M$ = UPPER$("ABCDefg")　　// M$值為“ABCDEFG”

VAL() 函數

【功能】VAL 傳回指定字串所表示的數值。

【版本】1.0 版(含)以上

【格式】*Y* = **VAL(***string-expression***)**

【註解】*string-expression*　可以是任何字串運算式。
如果字串運算式的第一個字元不是數字，那麼 VAL 將傳回 0。VAL 也會去除指定字串中的帶頭空白、 跳格(tab)、carriage-return 及換行(line-feeds) 等字元。

【範例】Y = VAL("　-3.84 ")　　// Y 值為–3.84

VALRAW() 函數

【功能】將數值依指定編碼方式轉換為字串。

【版本】4.0.2.3 版(含)以上

【格式】*M$* = **VALRAW$(***numeric-expression, string-constant***)**

【註解】*numeric-expression* 將被轉換為字串的數值。

string-constant 用來指定編碼方式的字串常數，必須是以下字串之一：

"INT16"	表字串為 16 位元的二進制編碼整數
"BCD16"	表字串為 16 位元的 BCD 編碼無號整數
"SBCD16"	表字串為 16 位元的 BCD 編碼有號整數
"INT32"	表字串為 32 位元的二進制編碼整數
"BCD32"	表字串為 32 位元的 BCD 編碼無號整數
"SBCD32"	表字串為 32 位元的 BCD 編碼有號整數
"FLOAT"	表字串為二進制編碼的單精準度浮點數
"DOUBLE”	表字串為二進制編碼的雙精準度浮點數

此函數傳回字串，其內容為將 *numeric-expression* 依 *string-constant* 指定的編碼方式加以編碼後的結果。

【範例】A$ = VALRAW$(16, "INT32")

A$ = "\x10\x00\x00\x00"

WEEKDAY() 函數

【功能】WEEKDAY 傳回目前系統日期為一週中的星期幾。

【版本】1.0 版(含)以上

【格式】*N%* = **WEEKDAY()**

【註解】此敘述傳回目前系統日期為一週中的星期幾(0 ~ 6， 週日= 0)。

【範例】N% =WEEKDAY()

WHILE … LOOP 敘述

【功能】只要給定的條件式之值為 true，WHILE 就會持續地重覆執行迴圈中的一連串敘述。

【版本】1.0 版(含)以上

【格式】**WHILE (***cond-expression***)**

。

。

。

LOOP

【註解】*cond-expression* 可以是任意運算式。

如果 *cond-expression* 是 true(非 0)，迴圈中的敘述會持續地執行，直到遇到 LOOP 敘述為止。接著 ***SmartScript*** 模組會回到 WHILE 敘述，並且再次檢查 *cond-expression*。如果它仍然是 true，就會重複此過程。如果不是 true，則跳到 LOOP 敘述之後的敘述繼續執行。

WHILE … LOOP 允許以多層巢狀的方式使用。 每一個 LOOP 均對應至離它最進的 WHILE。

【範例】WHILE(N == 1) //若 N 之值等於 1 則重複累加 X 之值，直到 N 之值不等於 1 為止

X = X + 1

LOOP

WRITE 敘述

【功能】WRITE 可以將資料寫入指定的檔案。

【版本】1.0 版(含)以上

【格式】**WRITE** *file-number* **,** *string-expression*

【註解】*file-number*　是一個整數運算式，其值為一檔案編號。

string-expression　包含要寫入檔案中的資料之字串運算式。

如果檔案編號關聯的檔案已被成功地打開，此敘述會將資料寫入檔案中。如果檔案是一個通訊裝置檔，此敘述會將資料寫入傳送佇列(transmit-queue)。

【範例】WRITE 1 , “This is a test。\r\n”

XOR08() 函數

【功能】XOR08 傳回指定字串的 8-bit exclusive OR checksum 值。

【版本】1.0 版(含)以上

【格式】*N%* = **XOR08(***string-expression***)**

【註解】*string-expression*　可以是任何字串運算式。

【範例】N% = XOR08(“\x02ABC\x03”)　　// N%值為 65

YEAR() 函數

【功能】YEAR 傳回目前的年份。

【版本】1.0 版(含)以上

【格式】*N*% = YEAR**()**

【註解】此函數傳回目前的年份。

【範例】N% = YEAR()

Appendix A　環境限制

❒ 名稱、字串與數值的限制

	最大值	最小值
變數名稱長度	16 characters	1 character
字串長度	32,767 characters	0 character
整數	-2147483648	2147483647
實數:		
正數	1.79769313486231D+308	4.940656458412465D-324
負數	-4.940656458412465D-324	-1.79769313486231D+308

❒ 陣列的限制

	最大值	最小值
陣列大小 (包含所有元素)	32,767 bytes (32K)	1 byte
允許使用的維數	3	1
陣列索引數字	8，192	1

❒ 程序與檔案的限制

	最大值	最小值
程式檔大小	2G bytes	0 byte
程序大小	65535 lines	0 line
行標的數目	65535	0
TAG 標記的數目	1024	0
變數的數目	65535	0
資料檔的數目	16	1
巢狀運算的層數	512	0

Appendix B　關鍵字

ABS()	**ACOS()**	**ALARM()**
ALMGRP()	**ALMPRI()**	**ALMTAG$()**
ASC()	**ASIN()**	**ATAN()**
BEEP	**CASE**	**CD**
CHOICE()	**CHR$()**	**CLOSE**
COMMODE	**COMOPEN**	**CONTINUE**
COPY	**COS()**	**COSH()**
CRC16()	**CRC32()**	**CREATE**
DATETIME$()	**DAY()**	**DEFAULT**
DEL	**DIM**	**DIR$()**
ELSE	**ELSEIF**	**END**
ENDIF	**ENDSW**	**ERRID()**
ERRORTAG	**EXEC**	**EXIT**
EXP()	**FAC()**	**FCHECK()**
FILE$()	**FLEN()**	**FMBCD()**
FMDBL()	**FMFLT()**	**FOR**
FORMAT$()	**FPOS()**	**FPRINT**
GOSUB	**GOTO**	**HOUR()**
IDLE	**IF**	**INT()**
ISTR$()	**IVAL()**	**LEFT$()**
LEN()	**LN()**	**LOG()**
LOOP	**LOWER$()**	**LTRIM$()**
MAX()	**MD**	**MESSAGE**
MID$()	**MIN()**	**MINUTE()**
MONTH()	**MOVE**	**MSGBOARD**
NERR()	**NOW()**	**NOW$()**
OPEN	**PASS**	**PI**
PLAY	**RAND()**	**RD**
READ	**RETURN**	**RIGHT$()**
RSTERR	**RTRIM$()**	**SECOND()**
SEEK	**SETDIR**	**SHORTCUT**

	SIN()	**SINH()**
SLEEP	**SQRT()**	**STOP**
STR$()	**STRING$()**	**SUM08()**
SWITCH	**SYSINFO$()**	**TAG()**
TAN()	**TANH()**	**TICK()**
TIMER()	**TOKEN**	**TONE**
TRAPOFF	**TRAPON**	**UPPER$()**
VAL()	**WEEKDAY()**	**WHILE**
WRITE	**XOR08()**	**YEAR()**

Appendix C　運算子的計算優先順序

運算子	運算	方向	優先順序
-	取負值(Negation)	←	High
!	邏輯 NOT(Logic NOT)		
NOT	反相		
^	指數乘冪(Exponentiation)	→	
*	乘法(Multiplication)	→	
/	除法(Division)		
\	整數除法的餘數運算(Integer Modulus)		
+	加法(Addition)	→	
-	減法(Subtraction)		
<<	向左移位(Shift left)	→	
>>	向右移位(Shift right)		
^<	向左循環(Rotate left)		
>^	向右循環(Rotate right)		
<	小於(Less than)	→	
>	大於(Greater than)		
<=	小於或等於(Less than or equal to)		
>=	大於或等於(Greater than or equal to)		
==	相等(Equality)	→	
!=	不相等(Inequality)		
AND	AND	→	
XOR	Exclusive OR	→	
OR	OR	→	
&	Logic AND	→	
|	Logic OR	→	Low

Appendix D　錯誤代碼

代碼	訊　　息
1	記憶體不足(Out of Memory) !!
2	變數過多(Too many Variable) !!
3	常數過多(Too many Constant) !!
4	行標過多(Too many Label) !!
5	超出運算子堆疊(Out of Operator Stack) !!
6	無效的行標(Invalid Line Label) !!
7	無效的命令(Invalid Command) !!
8	語法錯誤(Syntax Error) !!
9	無效的字串常數(Invalid String Constant) !!
10	無效的變數(Invalid Variable) !!
11	運算式過複雜(Expression too Complex) !!
12	過多的引數(Extra Argument) !!
13	資料型別不符(Type Mismatch) !!
14	陣列錯誤(Array Error) !!
15	數學運算錯誤(Math Error) !!
16	運算式錯誤(Expression Error) !!
17	未定義的行標(Undefined Label) !!
18	除以零(Division by Zero) !!
19	沒有 GOSUB 的 RETURN(RETURN without GOSUB) !!
20	沒有 IF 的 ENDIF(ENDIF without IF) !!
21	沒有 ENDIF 的 IF(IF without ENDIF) !!
22	沒有 SWITCH 的 ENDSW (ENDSW without SWITCH) !!
23	沒有 ENDSW 的 SWITCH (SWITCH without ENDSW)!!
24	沒有 FOR 或 WHILE 的 LOOP (LOOP without FOR or WHILE) !!
25	沒有 LOOP 的 FOR 或 WHILE(FOR or WHILE without LOOP) !!
26	無效的選項(Invalid Option) !!
27	程式未編譯(Program no Compiled) !!
28	程式已結束(End of Program) !!
29	程式中斷(Program Break) !!
30	超出範圍(Out of Range) !!
31	超出字串長度(Out of String Length) !!
32	操作失敗(Operation fail) !!
33	變數重覆定義(Variable redefined) !!
34	檔案代號使用中(File in used) !!
35	代號檔案未開啟(File not open) !!
36	無效的 TAG(Invalid Tag) !!
37	內部錯誤(Inner Error) !!