

ISaGRAF

Version 3.4

USER'S GUIDE

ICS Triplex ISaGRAF Inc.

Information in this document is subject to change without notice and does not represent a commitment on the part of ICS Triplex ISaGRAF Inc. The software, which includes information contained in any databases, described in this document is furnished under a license agreement or nondisclosure agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the license or nondisclosure agreement. No part of this manual may be reproduced in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of ICS Triplex ISaGRAF Inc.

© 2003 ICS Triplex ISaGRAF Inc. All rights reserved.

Printed in Canada

ISaGRAF is a registered trademark of ICS Triplex ISaGRAF Inc.

MS-DOS is a registered trademark of Microsoft Corporation.

Windows is a registered trademark of Microsoft Corporation.

Windows NT is a registered trademark of Microsoft Corporation.

OS-9 and ULTRA-C are registered trademarks of Microware Corporation.

VxWorks and Tornado are registered trademarks of Wind River Systems, Inc.

All other brand or product names are trademarks or registered trademarks of their respective holders.

目錄表

A.1	快速入門.....	A-2
A.1.1	安裝 ISaGRAF	A-2
A.1.2	使用線上求助	A-5
A.1.3	簡單的例子	A-5
A.2	案件管理.....	A-11
A.2.1	創造與使用案件	A-11
A.2.2	ISaGRAF 案件群組	A-13
A.2.3	選項	A-14
A.2.4	工具	A-14
A.3	程式管理.....	A-16
A.3.1	案件元件	A-16
A.3.2	編輯程式	A-18
A.3.3	執行應用程式碼產生工具	A-22
A.3.4	其他的 ISaGRAF 工具	A-24
A.3.5	增加工具功能表的命令	A-25
A.3.6	模擬及除錯	A-25
A.4	使用 SFC 編輯器.....	A-28
A.4.1	SFC 語言主題	A-28
A.4.2	SFC 流程圖形輸入	A-31
A.4.3	編輯存在的流程圖	A-32
A.4.4	輸入第二層程式	A-34
A.4.5	使用 SFC 倉庫	A-38
A.5	使用流程圖編輯器.....	A-40
A.5.1	基本的 FC 語言	A-40
A.5.2	輸入流程圖形	A-41
A.5.3	編輯已存在的圖形	A-45
A.5.4	輸入第二層程式	A-45

使用者指南

A.5.5	以 Quick LD 設計第二層程式.....	A-47
A.5.6	顯示選項.....	A-48
A.6	使用 QUICK LD 編輯器.....	A-49
A.6.1	階梯圖語言基礎.....	A-49
A.6.2	輸入階梯圖.....	A-52
A.6.3	編輯已存在的圖形.....	A-55
A.6.4	顯示選項.....	A-56
A.7	使用 FBD/LD 編輯器.....	A-59
A.7.1	FBD/LD 語言基礎.....	A-59
A.7.2	輸入 FBD 圖形.....	A-62
A.7.3	編輯已存在的圖形.....	A-64
A.7.4	顯示選項.....	A-66
A.7.5	樣式及更動記錄.....	A-66
A.8	使用文字編輯器.....	A-68
A.8.1	編輯命令.....	A-68
A.8.2	選項.....	A-69
A.9	有關程式編輯器的更多說明.....	A-70
A.9.1	呼叫其他 ISaGRAF 工具.....	A-70
A.9.2	程式的參數.....	A-71
A.9.3	“檔案”功能表中其它的命令.....	A-72
A.9.4	更新程式日記 (program diary).....	A-72
A.9.5	從字典中挑選變數.....	A-73
A.9.6	輸出視窗.....	A-74
A.10	使用字典編輯器.....	A-75
A.10.1	主字典視窗.....	A-77
A.10.2	變數管理.....	A-78
A.10.3	物件的描述.....	A-80
A.10.4	快速宣告.....	A-82
A.10.5	Modbus SCADA 位址對應.....	A-83

A.10.6	與其它應用程式交換資訊.....	A-83
A.11	使用 I/O 連結編輯器.....	A-89
A.11.1	I/O 板定義.....	A-90
A.11.2	板子參數設定.....	A-91
A.11.3	連結 I/O 接點.....	A-91
A.11.4	直接表示變數.....	A-92
A.11.5	編號.....	A-93
A.11.6	設定個別保護.....	A-93
A.12	創造轉換表.....	A-95
A.12.1	主要的命令.....	A-95
A.12.2	輸入表格的點.....	A-96
A.12.3	規則及限制.....	A-97
A.13	使用碼產生器.....	A-98
A.13.1	主要的命令.....	A-98
A.13.2	編譯選項.....	A-99
A.13.3	產生 C 原始碼.....	A-102
A.13.4	觀看訊息.....	A-102
A.13.5	定義資源.....	A-103
A.14	交互查詢.....	A-111
A.15	使用圖形除錯器.....	A-113
A.15.1	除錯器視窗.....	A-113
A.15.2	控制應用程式.....	A-114
A.15.3	選項.....	A-116
A.15.4	“寫入”命令.....	A-117
A.15.5	線上變更.....	A-119
A.15.6	DDE 動態交換.....	A-123
A.16	檢測變數集.....	A-124
A.17	除錯 ST 與 IL 程式.....	A-126

使用者指南

A.18 使用焦點除錯	A-127
A.18.1 建立圖形外觀	A-127
A.18.2 表列外觀	A-130
A.18.3 定義項目樣式	A-130
A.18.4 “檔案”功能表命令	A-131
A.18.5 ISaGRAF 3.2 版使用者注意事項	A-132
A.19 上載應用程式	A-133
A.19.1 上載案件	A-133
A.19.2 通訊設定	A-134
A.19.3 準備上載案件	A-134
A.19.4 壓縮檔如何儲存於目標硬體中	A-135
A.19.5 目標硬體的記憶體需求	A-135
A.19.6 關於上載案件	A-136
A.19.7 相容性	A-136
A.20 使用診斷工具	A-137
A.21 使用ISAGRAF 模擬器	A-138
A.21.1 除錯器的連接	A-138
A.21.2 I/O 模擬	A-138
A.21.3 資料庫元件	A-139
A.21.4 選項	A-140
A.21.5 儲存及回復輸入狀態	A-141
A.21.6 週期描繪器	A-141
A.21.7 模擬草稿	A-142
A.22 使用程式庫管理員	A-152
A.22.1 管理程式庫元件	A-152
A.22.2 I/O 組態	A-155
A.22.3 I/O 複合設備	A-156
A.22.4 I/O 板	A-157
A.22.5 用IEC 語言寫成的函數和方塊	A-159
A.22.6 “C”函數和功能方塊	A-160

A.22.7	轉換函數	A-161
A.23	使用檔案拷貝工具.....	A-163
A.23.1	呼叫檔案拷貝管理員	A-163
A.23.2	選項	A-164
A.23.3	備份和回存	A-164
A.23.4	備份檔	A-165
A.24	列印完整文件.....	A-166
A.24.1	自訂內容表	A-166
A.24.2	選項	A-168
A.25	密碼保護.....	A-171
A.26	進階程式技巧.....	A-175
A.26.1	關於ISaGRAF 工具的更多資訊.....	A-175
A.26.2	鎖住 I/O 點與虛擬 I/O 點.....	A-175
A.26.3	PC-PLC 連結.....	A-178
A.26.4	ISaGRAF 目錄.....	A-179
A.26.5	應用程式符號	A-182
A.26.6	ISaGRAF “無限點數” (WDL) 版工作平台的限制	A-188
B.1	案件架構.....	B-2
B.1.1	程式.....	B-2
B.1.2	循序的與週期的運作	B-2
B.1.3	SFC 及 FC 子程式.....	B-3
B.1.4	函數與副程式	B-4
B.1.5	功能方塊	B-6
B.1.6	描述語言	B-7
B.1.7	執行的規則	B-7
B.2	共同物件.....	B-9
B.2.1	基本型態	B-9
B.2.2	常數表示法	B-9
B.2.3	變數	B-13

使用者指南

B.2.4	註解.....	B-19
B.2.5	定義字.....	B-19
B.3	SFC 語言.....	B-22
B.3.1	SFC 圖表的主要格式.....	B-22
B.3.2	SFC 基本元件.....	B-22
B.3.3	發散和收斂.....	B-25
B.3.4	巨集步驟 (Macro steps).....	B-27
B.3.5	步驟中的行為.....	B-29
B.3.6	轉移條件的判斷條件.....	B-35
B.3.7	SFC 動態規則.....	B-38
B.3.8	SFC 程式的組織.....	B-39
B.4	流程圖語言.....	B-41
B.4.1	FC 元件.....	B-41
B.4.2	FC 複合結構.....	B-45
B.4.3	FC 動態行為.....	B-46
B.4.4	FC 檢查.....	B-46
B.5	FBD 語言.....	B-47
B.5.1	FBD 方塊圖的主要格式.....	B-47
B.5.2	跳回 (RETURN) 敘述.....	B-48
B.5.3	跳躍和標籤 (Jumps and labels).....	B-49
B.5.4	布林反相 (Boolean negation).....	B-50
B.5.5	在FBD 程式中呼叫函式或功能方塊.....	B-51
B.6	LD 階梯圖語言.....	B-52
B.6.1	電力軌和連接線.....	B-52
B.6.2	多連接.....	B-53
B.6.3	基本的LD 接點和線圈.....	B-54
B.6.4	跳回 (RETURN) 敘述.....	B-62
B.6.5	跳躍 (Jumps) 和標籤 (labels).....	B-63
B.6.6	在LD 中使用方塊.....	B-64

B.7	ST (結構化文字) 語言	B-66
B.7.1	ST 主要語法	B-66
B.7.2	表示式和括弧	B-67
B.7.3	函式或功能方塊呼叫	B-67
B.7.4	ST 的特定布林運算符號	B-69
B.7.5	ST 基本敘述	B-72
B.7.6	ST 延伸函數	B-80
B.8	IL 指令集語言	B-87
B.8.1	IL 主要語法	B-87
B.8.2	IL 運算元	B-89
B.9	標準運算元、功能方塊及函數	B-99
B.9.1	標準運算元	B-99
B.9.2	標準的功能方塊	B-127
B.9.3	標準的函數	B-150
C.1	簡介	C-2
C.2	安裝	C-3
C.3	ISAGRAF DOS 目標硬體入門	C-4
C.3.1	執行 ISaGRAF : ISA.EXE	C-4
	特殊事項	C-5
C.4	ISAGRAF OS9 目標硬體入門	C-10
C.4.1	執行單工的 ISaGRAF 目標程式 : isa	C-10
C.1.1	執行 ISaGRAF 的多工目標程式 : isaker , isatst , isanet	C-12
C.4.2	特殊事項	C-17
C.5	ISAGRAF VxWORKS 目標硬體入門	C-23
C.5.1	系統資源管理 : isassr.o	C-23
C.5.2	isa.o、isakerse.o 與 isakeret.o 的一般事項	C-23
C.5.3	執行 ISaGRAF 單工程式 : isa.o	C-24
C.5.4	執行 ISaGRAF 多工目標程式 : isakerse.o 與 isakeret.o	C-27

使用者指南

C.5.5	特殊事項.....	C-33
C.6	ISAGRAF NT 目標硬體入門.....	C-38
C.6.1	執行 ISaGRAF	C-38
C.6.2	一般的選項訊息.....	C-38
C.6.3	特殊事項.....	C-45
C.6.4	使用者介面.....	C-50
C.7	“C” 程式設計.....	C-57
C.7.1	概觀.....	C-57
C.7.2	“C” 轉換函數.....	C-59
C.7.3	“C” 函數.....	C-65
C.7.4	“C” 功能方塊.....	C-75
C.7.5	編譯與連結技術.....	C-96
C.8	MODBUS 連結.....	C-105
C.8.1	MODBUS 網路與協定	C-105
C.8.2	ISaGRAF 上的通訊協定.....	C-106
C.9	電源中斷管理.....	C-116
C.9.1	基本觀念.....	C-116
C.9.2	應用程式的變數備份.....	C-117
C.9.3	程式狀態備份.....	C-122
C.10	附錄：錯誤碼列表與描述.....	C-124

A. 使用者指南

A.1 快速入門

這一章包含 ISaGRAF 工作平台的安裝，以及一個簡單的 ISaGRAF 例子，給使用者一個簡短的整體輪廓，能立即的使用 ISaGRAF。

A.1.1 安裝 ISaGRAF

這一節教導使用者安裝 ISaGRAF 工作平台至你的電腦上，以及如何設定你的電腦來發展應用程式。

□ 軟硬體需求

ISaGRAF 工作平台能架在任何滿足 Windows 3.1 最小需求的電腦上，但是建議使用下面所列的硬體來發展你的應用：

80486 (含) 以上等級的個人電腦

(最好使用 Pentium 處理器)

最少 8 megabytes 的記憶體

(最好是 16 megabytes)

- 一個 3.5 吋 (1.44 megabyte) 軟碟機
- 一個至少 20 megabyte 可用空間的硬碟
- VGA、SVGA 或相容的顯示器
- 滑鼠 (圖形編輯時需用到)
- 平行 LPT1 埠 (給保護鎖 (protection key) 使用)

在安裝 ISaGRAF 工作平台之前，下列的軟體之一應該早已安裝於你的系統上：

- Windows 3.1 於 386 加強模式下
- Windows 95
- Windows NT 3.51 版或 4.00 版

使用安裝程式

使用 **INSTALL** 安裝程式來安裝 **ISaGRAF** 工作平台，**ISaGRAF** 會將光碟片或磁碟片的內容複製到使用者的電腦硬碟上，然後在程式管理員視窗下產生一個叫“**ISaGRAF**”的群組，以及於 **EXE** 次目錄底下產生初始設定檔“**ISA.ini**”。

INSTALL 是 Windows 的程式，必須於 Windows 檔案管理員下執行，或者於 Windows 95 開始下的執行命令下執行。安裝 **ISaGRAF** 請執行下列步驟：

插入 **ISaGRAF** 磁片 1 (或光碟片) 於 A 碟中 (或光碟機)。

選擇 Windows95 工作列的開始，或 Windows3.1 的程式管理員下的執行命令，從鍵盤鍵入 “A:\INSTALL.EXE” (使用磁碟片時) 或 “SETUP.EXE” (使用光碟片時，於光碟機的根目錄下)。

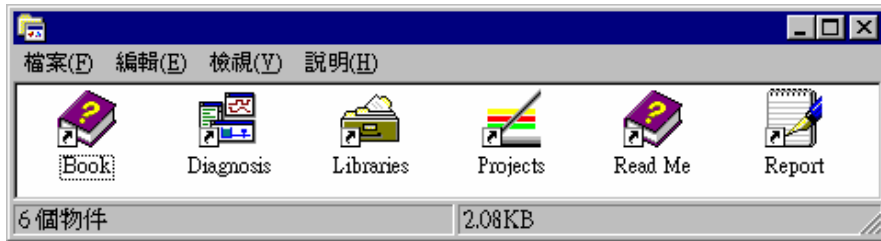
依據指示說明，完成安裝。你可以安裝多個版本的 **ISaGRAF** 工作平台於同一台電腦內，只要將它們安裝在不同的目錄內即可。

安裝時會要求使用者選擇想要安裝的元件，如下：

- **ISaGRAF** 執行檔
- 線上求助及說明檔
- **ISaGRAF** 標準的程式庫
- **ISaGRAF** 範例程式

建議於第一次安裝時將所有的元件都安裝進去，未安裝的元件或未來新增的元件可以再使用安裝程式將它們安裝進去。

ISaGRAF 主目錄內定為 “\ISAWIN”，且允許將不同版本的 **ISaGRAF** 安裝在相同的磁碟機上。**ISaGRAF** 的目錄結構可參考“進階程式設計技巧”的“**ISaGRAF** 目錄”章節。一但所有的 **ISaGRAF** 檔案都被安裝後，下面的群組會加到你的程式管理員視窗中：



下面是主要的 ISaGRAF 圖示說明：

Book： ISaGRAF 的線上求助

Diagnosis： 診斷工具

Libraries： 程式庫管理員

Projets： 案件管理員

Read Me： ISaGRAF 新版介紹

Report： 標準的 bug 報告格式

若你遭遇問題，使用標準 bug 報告格式來回覆。打開它，填寫報告內的欄位，然後使用主功能表的 檔案/儲存檔案 命令，給它一個檔名來儲存它，將它傳真或 e-mail 給 ICS Triplex ISaGRAF Inc. 公司。

□ 更新系統檔案

安裝完成後，於重新開機前需要更新 CONFIG.SYS 檔案，你不必將 ISaGRAF 目錄加入 PATH 變數內。ISaGRAF 沒有用到任何的 MS-DOS 環境變數，然而，你必須加入如下的指令到 CONFIG.SYS 檔案中：

```
files=20
```

```
buffers=20
```

ISaGRAF 工作平台使用序列埠與 ISaGRAF 目標硬體 PLC 作通訊，內定的 ISaGRAF 序列埠為 COM1，假如滑鼠已佔用 COM1，選擇 COM2 給滑鼠使用，如此內定的 COM1 對任何新的 ISaGRAF 應用程式都有效。

CONFIG.SYS 更改後，必須重新開機，新改的值才會生效。

⇒ Windows NT 使用者的注意事項：

當你的 ISaGRAF 工作平台是安裝在 Windows NT 3.51 或 4.00，必須更改 \ISAWIN\EXE 目錄下的 ISA.ini 檔案，將 NT=1 加入到 [WS001] 區段內：

[WS001]

NT=1

Isa=C:\ISAWIN

IsaExe=C:\ISAWIN\EXE

IsaApl=C:\ISAWIN\APLI

IsaTmp=C:\ISAWIN\TMP

使用序列埠通訊時，這些設定是絕對需要的。

A.1.2 使用線上求助

ISaGRAF 工作平台的線上求助有三個主題：

- ISaGRAF 語言參考
- 完整的使用者指南
- 標準函數庫元件技術說明

可從任何的 ISaGRAF 視窗，選擇“說明”功能表進入線上求助。

A.1.3 簡單的例子

這一章節一步一步解釋所有的基本指令，說明如何製作、設計、產生及測試一個簡短但是完整的多語言應用程式。

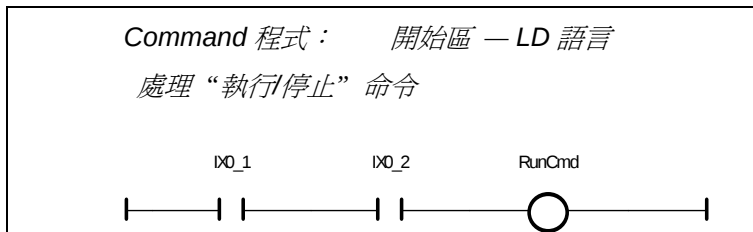
下面是這個應用程式的完整規格，混合了 LD 及 SFC 的表示式：

布林變數：

IX0_1,IX0_2: 程序命令，輸入變數

RunCmd: 執行/停止命令，內部變數

QX1_1: 程序狀態，輸出變數



開始執行ISaGRAF 工作平台

於 Windows95 工作列的開始，或 Windows3.1 的程式管理員的 ISaGRAF 群組內的“Projects”圖示上雙按滑鼠左鍵，打開案件管理員 (Project Management) 視窗。



開新案件

選擇“檔案”功能表下的“開新案件”命令，或按工具列上的開新案件按鈕，產生新案件，取名為“RunStop”，於開啓的對話框內輸入：

輸入案件名稱：“**RunStop**”

選擇 I/O 組態：“**Sim_Boo**”

按下“**確定**”按鈕

於案件管理員視窗視窗內可以看到案件被產生了。



開啓案件

選擇“檔案”功能表下的“開啓案件”命令，或者於 RunStop 的案件名稱上雙按滑鼠左鍵，或者使用工具列上的“開啓案件”按鈕，開啓 ISaGRAF 程式管理員。



開新程式

現在程式管理員視窗被打開了，裡面是空的（因為沒有任何程式被定義）。選擇“檔案”功能表下的“開新程式”命令，或於工具列上按“開新程式”按鈕，於出現的對話框內輸入：

輸入程式名稱：“**Command**”

選擇“**Quick LD**”語言

選擇“**開始**”區

按下“**確定**”按鈕，產生新程式。

重複上述動作產生第二個程式。

執行“檔案”功能表下的“開新程式”命令，或點選工具列上的“開新程式”按鈕，於開啓的對話框內輸入：

輸入程式名稱：“**RunStop**”

選擇“**SFC**”語言

選擇“**順序**”區

按下“**確定**”按鈕，產生新程式。

現在程式已被產生了，可以於程式管理員視窗內看到。



宣告變數

在編寫程式之前，程式所使用到的內部變數須先加以宣告，選擇“檔案”功能表下的“字典”命令或工具列上的“字典”按鈕來創造變數，I/O 變數於此案件創造時會自動宣告。



於出現的字典視窗中，選擇“檔案”功能表下的“其他”副功能表下的“全域變數”副功能表下的“布林”命令，切換至“全域”布林變數編輯視窗，使用工具列上的“全域物件”與“布林”兩個按鈕也可達到相同的效用。

使用者指南

 選擇“編輯”功能表下的“造新變數”命令，來創造新的布林變數，亦可以使用工具列上的“插入物件”按鈕。於開啓的對話框中輸入如下的內容：

變數名： RunCmd

註解： 內部執行停止命令

屬性： 選擇“內部”屬性

按“儲存”按鈕儲存：變數被產生了。

按“取消”按鈕關閉對話框。

最後，離開字典編輯器，並且儲存所做的變更：選擇“檔案”功能表下“離開”命令，然後按“是”儲存所做的變更。



編輯 Quick LD 程式

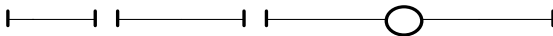
編輯 LD 程式“Command”，於程式管理員視窗內的“Command”程式名稱上雙按滑鼠左鍵，或者使用“編輯”按鈕。



出現 ISaGRAF Quick LD 編輯視窗，爲了增加工作區域，將視窗放至最大。

F2 F3 按鍵盤上的 F2 及 F3：

(*)



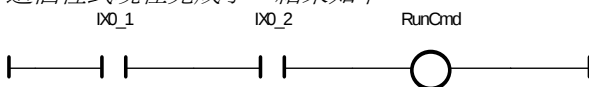
將變數連結到 LD 元件上：使用鍵盤上的方向鍵，移動游標至各個元件上，然後按 Enter 鍵，出現選擇變數對話框。

於第一個接點上，在選擇變數對話框內輸入：IX0_1 後按 Enter。

於第二個接點上，在選擇變數對話框內輸入：IX0_2 後按 Enter。

於線圈上，在選擇變數對話框內輸入：RunCmd 後按 Enter。

這個程式現在完成了，結果如下：



離開這個編輯器，並且儲存所做的變更：選擇“檔案”功能表下的“離開”命令，於出現的對話框內點選“是”按鈕儲存所做的變更。

編輯 SFC 程式

編輯 SFC 程式“RunStop”，於程式管理員視窗內的“RunStop”程式名稱上雙按滑鼠左鍵，或者使用工具列上的“編輯”按鈕。



現在 SFC 編輯視窗被打開了，為了增加工作區域，將視窗放至最大。



SFC 編輯視窗中一開始即會有一初始步驟 (initial step) 存在並且已被選取。按下鍵盤的“向下 (Down)”箭頭按鍵來選取初始步驟 (0,1) 之後的空位置。

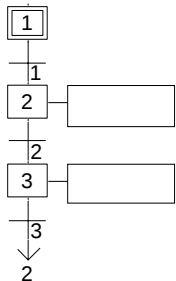
F4 F3 按下 F4，然後再按 F3 來插入一個步驟 (step) 及一個轉移條件 (transition)。

F4 F3 按下 F4，然後再按 F3 來插入更多的步驟 (step) 及轉移條件 (transition)。

F5 按下 F5 來插入一個“跳躍至步驟 (jump to a step)”符號，並且選擇 GS2 為此跳躍的目的地。



到此，SFC 流程圖就完成了。於工具列上點選“放大”按鈕來增加編輯區間的大小，並且給予第二層 (level 2) 程式顯示的空間，結果如下：



欲編輯編號為“2”的轉移條件的程式，請用鍵盤的方向鍵來選取它，並且按下“Enter”鍵，文字編輯視窗就會開啓，此時就可以輸入轉移條件 2 的第二層程式：

```
RunCmd;
```

使用者指南

^TAB 按下 “Control + Tab” 鍵將焦點移回 SFC 流程圖，選取步驟 3，並且按下 “Enter” 鍵來編輯它的第二層程式：

QX1_1;

使用相同步驟，編輯轉移條件 3 的第二層程式：

Not (RunCmd);

^F4 按下 “Control + F4” 鍵來關閉第二層視窗。

到此 SFC 程式完成了，選擇 “檔案” 功能表下的 “離開” 命令，離開 SFC 編輯視窗，然後點選 “是” 按鈕儲存所做的變更。



建立應用程式碼 (application code)

從程式管理視窗下選擇 “製作” 功能表下的 “製作應用程式” 命令，或使用工具列上的 “製作應用程式” 按鈕，建立應用程式碼。

當應用程式碼產生後，會出現一個對話框，問你要現在離開這個視窗或者留在這個視窗內繼續工作，請選擇 “離開” 按鈕。



模擬

從程式管理視窗下選擇 “除錯” 功能表下的 “模擬” 命令，執行 ISaGRAF 核心 (kernel) 模擬器，你也可以點選工具列上的模擬按鈕來執行此命令。

當模擬器視窗出現時，可以使用模擬輸入按鈕 (綠色的按鈕) 來測試程式，模擬輸出以紅燈表示。在這個例子中兩個輸入按鈕 (編號 1 及 2) 必須被按下，程序才會執行 (紅色的 LED 燈)。

使用 “檔案” 功能表下的 “離開” 命令，關閉除錯視窗，離開模擬器。

A.2 案件管理

欲執行 ISaGRAF 案件管理工具，可於 ISaGRAF 群組內的“Projects”圖示上雙按滑鼠左鍵，開啓案件管理員。一個案件代表目標硬體 PLC 內的一個 PLC 程式。視窗的上半部包含所有已存在案件的集合，被選擇的案件說明會顯示在視窗的下半部。

視窗縮放

視窗內有一條視窗分隔線，將視窗分爲兩部分：案件列表與描述，使用者可以移動這一條分隔線來調整這兩部分的大小。描述視窗無法被完全隱藏，它至少會保留一行的文字空間。



插入分隔線

分隔線能被插入在任何案件名稱之前，這個功能允許使用者將相似的應用集中在同一區間。使用“編輯”功能表下的“分隔線”命令來插入或刪除分隔線。



於案件列表中移動案件

移動案件之前，需先將所欲移動的案件選擇起來（反白），於案件的名稱上按滑鼠左鍵，然後將他拖至新的位置。當拖移的時候，左邊界上有一小箭頭指示所放置的位置，你也可以使用“編輯”功能表的“移動”命令，一次移動一行。假如分隔線位於所選擇的案件之前，他亦會跟著移動。

A.2.1 創造與使用案件

案件管理員功能表中的命令適用來產生新的案件、編輯案件以及管理已存在的案件。



產生新的案件

使用者指南

要產生新的案件，首先輸入案件名稱，然後沒有任何內容的空案件被產生。
I/O 組態能夠被依附到新產生的案件中，所依附的 I/O 組態必須已定義於程式庫中，假如你有選擇 I/O 組態，ISaGRAF 將自動設定 I/O 連結，並於新案件字典中宣告相對應的變數。當開新案件或案件更改名稱時，必須遵守下面的案件命名規則：

名稱不能超過 **8** 個字元

字首必須是**英文字母**

字首以後的字元可以是**英文字母**、**數字**以及**底線符號**

案件名稱不分大小寫

當案件產生時，使用“**編輯 / 設定註解文字**”命令來輸入案件的註解。



編輯案件描述

“**案件 / 案件描述**”命令用來編輯案件的文字描述，這個文件可用來識別此案件與其他案件的不同點，亦能於案件生命週期內紀錄任何的註記。



編輯案件

“**檔案 / 開啓**”命令開啓所選案件的程式管理視窗，從這個視窗中，所有的案件內容（程式，應用程式參數...）都能被管理。你也可以於案件名稱上雙按滑鼠左鍵來編輯案件。



變更的歷史紀錄

在案件的生命週期中，ISaGRAF 會記錄各元件的任何變動，每一項的變動會儲存變動標題、日期及時間。歷史紀錄檔包含最後的 **500** 次變動紀錄，每一個案件都有自己的紀錄檔。“**案件 / 歷史紀錄**”命令允許使用者觀看與列印所選案件的變動歷史紀錄，使用者可以選擇項目列中一個或多個項目，並按下下述的按鈕：

確定 關閉這個視窗

列印 將項目內容送至印表

說明 顯示這個對話框的說明

[刪除] 選擇部分 刪除所選擇的項目

[刪除] 全部 刪除所有的項目

尋找 尋找字串

“**尋找**” 按鈕上方的輸入盒用來輸入所欲尋找的字串，這個功能是與大小寫無關的，當搜尋至項目列底端的時候，下次尋找時會從頭再開始搜尋。



列印完整的文件

“**案件 / 列印**” 命令允許使用者建立並列印完整的文件，這個文件能夠將所選擇的案件的任何元件（程式、變數、參數...）集合在一起，若要建立指定的文件（非完整），使用者僅需定義表格的內容即可。

密碼保護

“**案件 / 設定密碼**” 命令可對所選案件用到的工具及資料作密碼保護，在手冊第一部份後面“**設定密碼**” 章節，可以看到密碼階層及資料保護的更多說明。密碼僅對所選的案件有作用，對其他的案件及 ISaGRAF 程式庫沒有影響。

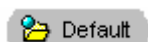
A.2.2 ISaGRAF 案件群組

一個 ISaGRAF 案件相對於一個磁碟上的目錄，所有的案件檔案皆會儲存在此目錄下。“**案件群組**” 相當於相同根目錄下的案件目錄群集合，案件群組以名稱為識別。ISaGRAF 下會產生兩個內定的案件群組：

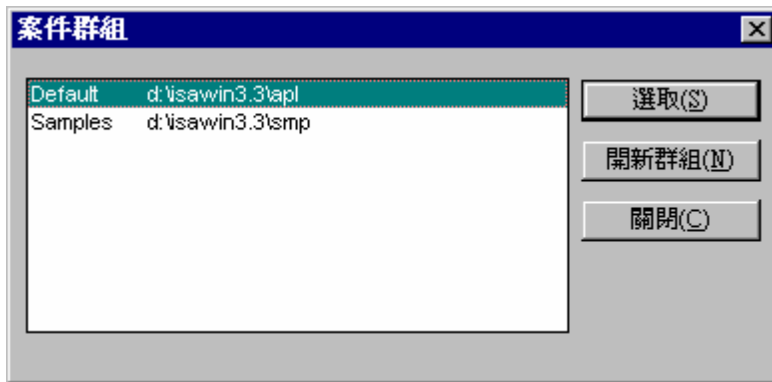
“**Default**” 位於 “\ISAWIN\APL” 下：你的工作區

“**Samples**” 位於 “\ISAWIN\SMP” 下：ISaGRAF 工作平台所附加的範例應用

目前所選的案件群組名稱顯示於工具列上，名稱旁邊的按鈕可用來選擇案件群組：



你亦能執行 “**檔案 / 選擇案件群組**” 來選擇已存在的群組或產生新的群組。下面是打開後的視窗：



於表列中選擇一個群組，然後按“**選取**”按鈕，使此群組作用於案件管理表列中。你亦可於群組名稱上雙按滑鼠左鍵，來選擇此群組。使用“**開新群組**”命令來產生新群組，這個命令能用於指定群組名稱到已存在的目錄，也可以產生新目錄下的新群組。

注意： 當其他的 ISaGRAF 視窗 (程式管理員、編輯器...) 開啓時，無法選取或產生新案件群組。

A.2.3 選項

“**選項**”功能表用來顯示或隱藏工具列、選擇字型以及設定案件管理員“自動關閉”模式，這裡所選的字型套用在案件說明以及所有的 ISaGRAF 文字編輯器。

若關閉“**保持案件管理員開啓**”選項，當開啓案件時，案件管理員會自動關閉。

A.2.4 工具

“**工具 (Tools)**”功能表下的命令可用於執行其他 ISaGRAF 應用。“**工具 / 備份 案件**”命令執行 ISaGRAF 備份管理員，用來儲存或回復案件。“**工具 /**

備份 共同資料 用來儲存或回復所有案件（如共同定義字）所用到的資料檔。

“工具 / 程式庫” 命令執行 ISaGRAF 程式庫管理員。

“工具 / 輸入 IL 程式” 會根據 PLC Open 組織所制定的檔案交換格式輸入文字檔，轉換成 IL 程式。

A.3 程式管理

程式管理視窗顯示這個應用的程式（亦稱為模組或程式單元），且將可用的命令群聚成功能表，使用者可用這些命令來制定案件架構、執行編輯器、編譯及除錯。在發展應用時，程式管理視窗是 ISaGRAF 工作平台的核心。要開啓程式管理視窗可於案件管理員下執行“開啓”命令。

A.3.1 案件元件

案件元件稱為**程式**，程式是控制流程執行的一部份。全域變數（譬如 I/O 變數）能夠給這個應用中的任何程式使用，區域變數僅能給一個程式使用。各程式以**樹狀體系**（hierarchy tree）表示，且分為幾個不同的**邏輯區域**（logical section）。這個視窗顯示所有的程式及它們的關係。“**頂層**”程式列於樹狀體系的最左邊。

□ 頂層程式

頂層程式列於樹狀體系的最左邊，在前三個區域的頂層程式永遠都是執行的，於執行週期（run time cycle，或稱為掃描週期）中，它們以下面的順序執行：

（讀取輸入點）

執行**開始**區內的頂層程式

執行**順序**區內的頂層程式

執行**結束**區內的頂層程式

（更新輸出點）

在“**開始**”區或“**結束**”區的程式是週期反覆執行的，與時間無關。在“**順序**”區的程式則是順序執行的，其中計時器變數可用來區別不同的基本操作。“**開始**”區中的主程式在每一執行循環的開始都會被執行一次，“**結束**”區中的主程式在每一執行循環的結尾都會被執行一次，“**順序**”區的主程式則依據 **SFC** 或 **FC** 的規則執行，且必須用 **SFC** 或 **FC** 語言來撰寫，而週期

區的程式則不能用 **SFC** 或 **FC** 語言撰寫。所有的程式皆可以擁有一個或多個 **副程式**。

▣ 函數與功能方塊

“**函數**”區的程式能被這個案件中的任何區的任何程式所呼叫。所謂函數是指將一些輸入值經過運算後產生一個輸出值。函數運算僅使用到暫時性的變數，當從一個呼叫到另一個呼叫時，這些變數即會消失，這意味著函數不應呼叫功能方塊。“**函數**”區的程式不能以 **SFC** 或 **FC** 語言來撰寫。

不像函數，“**功能方塊**”使用到隱藏的靜態資料來處理輸入值，這些資料在每一次不同的呼叫時，系統會自動複製一份。“**功能方塊**”區的程式能夠被同一案件中的任何區的任何程式所呼叫。功能方塊區的程式不能以 **SFC** 或 **FC** 語言來撰寫。

▣ 副程式

副程式是函數的一種，但是只提供給一個父程式 (**SFC**、**FC** 或其他語言) 來呼叫，其他的程式無法呼叫它，每一個區的程式都可以有一到多個副程式，除了 **SFC** 和 **FC** 語言外，都可以用來撰寫副程式。

▣ **SFC** 子和 **FC** 副程式

SFC 子程式是一種與父程式並行執行的程式，他的父程式可以執行它或刪除它。父程式與子程式都必須以 **SFC** 語言來撰寫。

當父程式執行 **SFC** 子程式時，他放一個 **SFC 執行權杖 (token)** 進入子程式的每個初始步驟。當刪除 **SFC** 子程式時，將清除子程式所有存在的執行權杖。

順序區中的任何 **FC** 程式皆可控制其他的 **FC** 副程式。在 **FC** 副程式執行期間，**FC** 父程式會暫停 (等待)。FC 父程式和它的任何一個 **FC** 副程式同時執行是不可能的。

▣ 程式與副程式間的連結：

使用者指南

副程式與子程式在其樹狀體系中以一條連結線來表示和父程式的連結關係，**SFC** 程式與 **SFC** 子程式間的連結線則在尾端多一個箭頭，這樣的連結表示是**平行執行**的。

▣ 程式設計語言

每一個程式只能以一種**語言**撰寫，每當開新程式時得選一種語言來撰寫，並且往後都不能更改成其他種的語言，但是，**FBD** 和 **LD** 的元件可以混合運用在同一個程式中。**ISaGRAF** 提供的圖形式語言有：**SFC** (順序式功能圖，Sequential Function Chart)，**FC** (流程圖，Flow Chart)，**FBD** (功能方塊圖，Functional Block Diagram) 與 **LD** (階梯圖，Ladder Diagram)，文字式語言有：**ST** (結構化文字，Structured Text) 與 **IL** (指令集，Instruction List)。**SFC** 和 **FC** 語言是被保留給順序處理區中的主程式與子程式的。於程式管理視窗中，每一個程式名稱旁邊都有個顯示其語法的圖示。各種語言的表示符號如下：

	SFC	順序式功能圖
	FC	流程圖
	FBD	功能方塊圖
	LD	階梯圖
	ST	結構化文字
	IL	指令集

A.3.2 編輯程式

“**檔案**”功能表集合了創造、更新與變更程式的所有命令，他亦可以開啓應用程式所用到的適當編輯器。

產生新的程式

“**檔案**”功能表下的“**開新程式**”命令允許使用者在程式的任何區域中創造頂層程式、子程式或副程式。視窗內的名稱方框是用來輸入新程式的名字，必須遵守下列的命名規則：

- 程式名最長不可超過**8** 個字

- 程式名的第一個字必須為**英文字母**
- 第一個字後面的字可以是**英文字母**、**數字**或**底線** ‘_’
- 程式的名稱不分大小寫

接下來，選擇一種語言來撰寫新程式：

SFC 順序式功能圖

FC 流程圖

FBD 功能方塊圖 (可以包含 LD)

LD 階梯圖 (指使用 Quick LD 編輯器)

ST 結構化文字

IL 指令集

最後，幫程式選擇一種執行模式：

開始 “開始” 區的頂層程式

順序 “順序” 區的頂層程式

結束 “結束” 區的頂層程式

函數 “函數” 區程式

功能方塊 “功能方塊” 區程式

...的子程式 一個已存在程式的 SFC (或 FC) 子程式或副程式

選擇上述前五項的其中一項後，程式將被放在**開始**、**結束**、**順序**、**函數**或**功能方塊**區的最上層，如果選取了最後一個選項則新的程式將會是一個 **SFC** 子程式，一個 **FC** 副程式或副程式。記住，一個順序區最上層的程式必須以 **SFC** 或 **FC** 語言撰寫，且 **SFC** 和 **FC** 語言是不能在週期區中使用的或當此區的副程式。

☞ 輸入每個程式的註解

ISaGRAF 允許使用者將案件的每個程式附上一段文字說明，這段註解文字以較小的字形顯示於程式名稱旁邊。使用“**檔案 / 程式註解文字**”命令來輸入或更改所選擇程式的註解。



編輯程式內容

使用者指南

這個命令允許使用者修改程式內容，不同的語言會有不同的編輯器。每個程式是在一個單獨的視窗中編輯，因此可以同時打開多個程式編輯視窗。除了使用功能表來打開程式編輯器，也可以使用方向鍵，移動反白區至所欲編輯的程式，按下 **ENTER** 鍵時便可編輯此程式，使用者也可將滑鼠游標指到此程式名稱上雙按滑鼠左鍵，來打開編輯視窗。



編輯日記檔

日記檔是伴隨著每一個程式產生的，它是個文字檔，內容記錄了對程式所做過的所有修改。日誌檔是可編輯的，可以在任何時間自由的修改或列印。當離開一個做過修改的程式編輯視窗時，會有一個可輸入摘記的視窗自動開啓，這裡所輸入的摘記會加上正確的日期及時間被插入到日誌檔中。



變數字典

“**檔案 / 字典**”命令會開啓字典編輯器，你可以在裡面宣告案件所用到的變數。變數可以是全域的變數（可以被案件裡的所有程式所認知）或被選取程式的區域變數。字典編輯器亦可以用來宣告**定義字**，定義字等於是一種化名，可以用來取代一個名稱，或取代程式原始碼中的一個表示式。



函數、副程式與功能方塊的參數

“**檔案 / 參數**”命令允許使用者對所選取的副程式、函數或功能方塊做呼叫及回傳參數的定義，這個命令對“**開始**”或“**結束**”區的主程式或 SFC 程式是沒有作用的。

副程式、函數或功能方塊可以有多達 **32** 個參數（輸入加上輸出），函數及副程式會有一個輸出參數（且僅能有一個），它的名稱必須與函數或副程式名稱一樣，這是爲了 ST 語言撰寫上的方便。

視窗的左上部分顯示出參數，順序爲：前面是呼叫參數，後面是回傳參數。視窗的下半部分顯示所選參數的詳細敘述，ISaGRAF 中的任何資料型態都可以使用在參數上。回傳參數必須位於輸入參數之後。參數名稱必須遵守如下的命名規則：

- 名稱長度不能超過 16 個字

- 第一個字元必須為英文字母
- 其後的字元可以是英文字母、數字或底線符號
- 參數名稱不分大小寫

“**插入**”命令是用來在所選取的參數之前插入一個新的參數，“**刪除**”命令是用來刪除所選取的參數，“**排列**”命令會自動地將參數做排序，因此返回參數會被放在最後面。

二 於樹狀體系中移動程式

“**檔案**”功能表下的“**更名 / 搬移**”命令用來改變程式的名稱，或是將程式搬移到結構樹中的另一個區域，但是已經存在的程式其語言是無法改變的。當執行這項命令時，會出現一個與開新程式一樣的視窗，而各個欄位的內定值為被選取程式之前的屬性，程式的名稱是可以更改的，而選取另一個區域或父程式就可以將程式搬移到結構樹中的不同位置。

“**檔案**”功能表下的“**排列程式**”指令是用來調動同一區下或同一個父程式下程式的順序。如果所選取的程式是位於最上層，則此命令針對所選取區域的頂層程式做排序，如果所選擇的程式是位於較低的層，則此指令是針對所選取並擁有同一個父程式的副程式或子程式做排序。當“**排列程式**”的對話視窗開啓後，選取你想要移動的程式，然後按“**向上**”或“**向下**”鍵來移動程式到你想要的位置。



複製程式

想要複製一個程式，首先選取欲複製的程式，然後執行“**檔案 / 複製**”命令。執行這項命令時，會出現一個與開新程式一樣的視窗，而其中各個欄位的內定值就是被選取程式之前的屬性。輸入目的地程式的名稱及其位於程式結構樹中的位置，假如欲複製的目的地已存在，將會被覆蓋掉。程式定義的所有區域變數及定義字會隨之複製過去，目的程式與被複製的程式必須用相同的語言撰寫。按下“**確定**”鍵來複製程式。

“**檔案**”功能表中的“**複製到其他案件**”命令將選取的程式複製到另一個案件中，且程式名稱不變。所選取程式底下的 SFC 子程式與副程式會一併複

使用者指南

製，所選取欲複製的程式及其子、副程式的名稱皆不可出現在目標案件中，程式無法經由此指令而將之覆蓋掉。所有經程式宣告的區域變數及定義字將會隨著程式複製過去。



刪除程式

想要刪除一個程式，首先選取欲刪除的程式，然後執行“**檔案 / 刪除**”命令。一個程式如果擁有副程式或子程式時，此程式是無法被刪除的，必須先刪除程式下層的副程式或子程式。所有經程式宣告的變數及定義字將會隨著程式一起被刪除。

⇒ 輸入程式庫中的函數或功能方塊

“**工具 / 從程式庫輸入**”命令用來複製程式庫中以 IEC 語法撰寫的函數或功能方塊到開啟案件的“**函數**”或“**功能方塊**”區中，所有經程式宣告的區域變數及定義字將會隨著輸入的函數複製過去。當一個函數確定從程式庫中被輸入到案件後，藉由執行“**檔案**”功能表中的“**更名 / 搬移**”命令，此函數可被放置在樹狀程式中的任何區域中的任何位置。為了避免名稱衝突，輸入至案件中的函數或功能方塊必須更改名稱，但不要忘了，回傳參數名稱也要一起更改。

⇒ 輸出函數或功能方塊到程式庫中

“**工具 / 輸出到程式庫**”命令是將存在於“**函數**”或“**功能方塊**”區的程式輸送到程式庫中，所有經程式宣告的區域變數及定義字將會一起複製過去。輸出的函數應在程式庫管理員中被重新編譯（驗證），以保證此函數能在程式庫環境中使用。程式庫中的函數是不能使用全域變數的。

A.3.3 執行應用程式碼產生工具

“**製作**”功能表命令用來執行應用碼產生器，以及輸入產生應用碼時的選項與附加資料，參考“**使用應用程式碼產生器**”章節，以得到更多的說明。



製作應用程式碼

“**製作**”命令用來產生應用程式碼，執行這項命令之前，目的碼的選項必須設定正確。在產生目的碼之前，任何程式都會經過語法的驗證，ISaGRAF只會針對未編譯過的程式加以編譯。



驗證選取的程式

“**驗證**”命令允許使用者驗證所選取的程式，對於一個已驗證過且沒有語法錯誤的程式，在產生應用程式碼時不會再重新編譯一次，除非它的程式內容或相關的定義字與變數改變了，才會再重新編譯。

▣ 模擬變更

“**更新**”命令模擬各程式已被更改過了，因此下一次編譯時所有的程式皆會被重新編譯。



應用程式執行期選項

這個命令會開啓一個對話盒，從這個對話盒裡可以輸入應用程式執行時期的參數，它包括程式的執行週期、執行期錯誤管理、開始模式及可回復變數的設定。參考“使用應程式碼產生器”章節可得到更多的說明。

▣ 編譯選項

這個命令用來設定應用程式碼產生器的選項，可產生最佳的目的碼。參考“使用應程式碼產生器”章節可得到更多的說明。

▣ 定義資源

“**資源**”是使用者定義的資料（例如一個檔案），用來和目的碼結合，因此它是可以跟目的碼一起下載的。參考“使用應用程式碼產生器”章節可得到更多有關資源定義檔案格式的說明。

A.3.4 其他的 ISaGRAF 工具

“**案件**” 功能表集合了被選取案件可執行的 ISaGRAF 工具，參考對應的章節可得到更多的說明。



I/O 變數連結

“**I/O 連結**” 命令開啓 ISaGRAF I/O 變數連結編輯器，這個工具建立所宣告的 I/O 變數與硬體 I/O 板的關係。



執行交互查詢編輯器

“**交互查詢**” 命令允許使用者對案件的交互查詢做統計、檢視或將其列印出來。交互查詢列出在整個案件中於程式原始碼裡面所有出現過的變數，這項功能對蒐尋變數或資源來說是非常有用的，它也能列出在原始碼中所有出現過的全域變數。

▢ 輸入案件描述

“**案件描述**” 命令用來編輯案件的文字描述，由這個文件可以識別案件的內容，這個案件說明檔亦能用來加註案件生命週期的任何註記。案件描述會顯示在案件管理視窗內。

▢ 列印完整文件

“**列印案件文件**” 指令准許使用者對一所選物件去建立和列印一份完整的文件。任何關於此專案的資訊（程式、變數、參數...）都可放入這專案文件內。爲了能作到這件事，使用者要建立起這份文件的內容表。

▢ 變更歷史

這個命令會開啓顯示變更歷史的對話盒。參考“**案件管理**” 章節可得到更多的說明。

A.3.5 增加工具功能表的命令

ISaGRAF 提供在“工具”功能表內加入其他命令的方法，在“`\\ISAWIN\\COM\\ISA.MNU`”文字檔中，使用者可以加入自己命令，你總共可以加入十個命令，註解可以加在“`;`”之後，每一個命令以兩行文字列表示，如下面的語法所示：

```
M=menu_string
      C=command_line
```

“`menu_string`”是顯示在“工具”功能表的文字字串，“`command_line`”是任何 MS-DOS 或 Windows 可執行的命令，你可以在字串的後面加上引數。在“`command_line`”中可以使用“`%A`”字串來代替開啓中的案件名，使用“`%P`”字串代替所選取的程式名稱。下面的例子執行“記事本”來編輯所選的程式（使用 ST 或 IL 程式）：

```
M=Edit with Notepad
C=Notepad.exe \\isawin\\ap\\%A\\%P.lsf
```

A.3.6 模擬及除錯

“除錯”功能表下的命令用來執行 ISaGRAF 圖形除錯器，有模擬與真實連線兩種模式。



模擬

“模擬”命令開啓模擬模式下的除錯器，在此模式下，會出現一個叫模擬器的視窗。當沒有目標硬體時，可以使用這項命令來測試你的應用程式。啓動模擬器時會關閉程式管理視窗，在除錯器與模擬視窗兩者皆被開啓之後，程式管理視窗將被重新開啓。如果應用程式碼還沒產生，您將無法啓動模擬器，若子視窗（編輯器、應用碼產生器、I/O 連結.....）尚未關閉，也無法啓

使用者指南

動模擬器，必須將他們都關閉才可以使用這項命令。這項命令也可以從 ISaGRAF 編輯器的功能表下來執行。

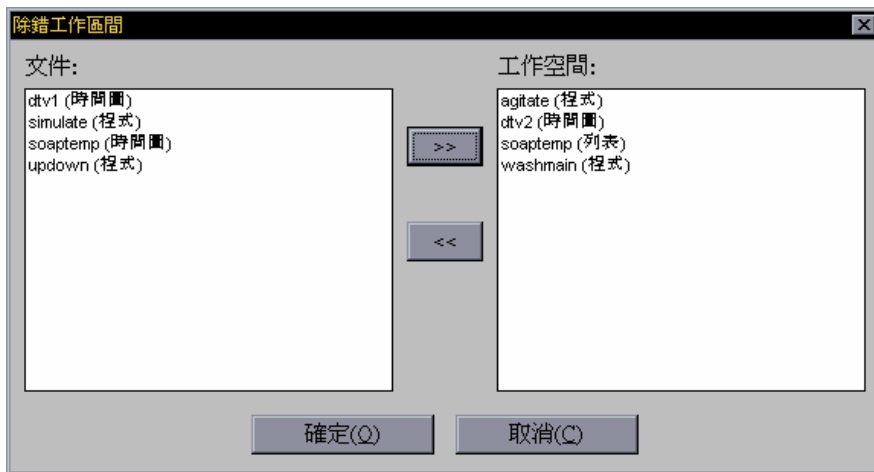


真實環境的除錯

“除錯”命令開啓除錯器主視窗，並關閉程式管理視窗，然後，當除錯器與目標硬體應用程式之間的通訊建立後，程式管理視窗將立刻被重新開啓。如果應用碼還沒產生，您將無法啓動除錯器，若子視窗（編輯器、應用碼產生器、I/O 連結.....）尚未關閉，也無法啓動模擬器，必須將他們都關閉才可以使用這項命令。這項命令也可以從 ISaGRAF 編輯器的功能表來執行。

準備 ISaGRAF 工作空間

“除錯 / 工作空間”命令讓你能夠定義工作空間的初始文件集，這些文件可以是程式、焦點圖形、以及變數集，先前 ISaGRAF 版本的圖形及時間圖集亦列於案件文件集中。當模擬或線上監視時，這些定義的文件將自動開啓。



上述之對話框左邊顯示已存在的案件文件，右邊顯示初始工作空間選取的文件。使用“>>”及“<<”按鈕來搬移文件至另一邊。每一個案件都可以擁有自己的初始工作空間。



連結設定

“**連結設定**”命令能讓使用者設定 PC 主機與目標硬體 ISaGRAF 系統 (如 PLC) 間通訊連結的參數。

“**目標硬體 Slave 編號**”用來識別 ISaGRAF 的目標硬體系統或程式，範圍從 **1** 到 **225** 之間，可查閱供應商提供的目標硬體技術手冊，以得到它的編號設定。

“**通訊埠**”用來識別 ISaGRAF 工作平台與目標硬體之間的通訊媒介，它可以是 “**序列埠**” 或 “**Ethernet**”，Ethernet (乙太網路) 使用 “Winsock” 1.1 版的 TCP/IP 通訊。

“**失敗時間**”是目標硬體系統在接收到除錯器命令後可以接受的最大回應時間，且以**毫秒**為單位。“**重試次數**”欄位是指當偵測到通訊錯誤時重複執行同一項通訊命令的次數。

□ 序列埠連結設定

當選擇序列埠 (COM1..4) 後，點取 “**設定**” 按鈕來設定其他的序列通訊參數。

鮑率 (baud rate)、同位元 (parity) 與格式 (format) 可以在這裡設定，當 “**流量控制**” (flow control) 中選取了 “**hardware**” 選項後，則 ISaGRAF 工作平台將控制 CTS 與 DSR 線作硬體的通訊交握。

□ 乙太網路連結設定

當選擇 “Ethernet” 當作通訊埠，點取 “**設定**” 按鈕來進入 “網路位址” (internet address) 及 “埠號碼” (port number) 的設定畫面。

這些欄位使用 Socket 介面的標準模式。ISaGRAF 工作平台使用 “WINSOCK.DLL” 1.1 版的程式庫來作 TCP/IP 通訊，此檔必須確定已安裝在硬碟中。若沒有特別的指定，則 “**1100**” 將是內定的埠號碼。

A.4 使用 SFC 編輯器

SFC 語言是用來描述循序式的程序 (sequential process) 操作，它使用簡單的圖形來表示不同的程序步驟與轉移條件。ISaGRAF 提供 SFC 圖形編輯器來撰寫 SFC 程式，SFC 是 IEC1131-3 標準的核心語言，它的步驟 (step) 行為及轉移條件 (transition) 內的邏輯條件則使用其他的語言來描述。ISaGRAF 的 SFC 圖形編輯器允許使用者編輯完整的 SFC 程式，包含了圖形及文字編輯能力，也就是可以編輯 SFC 流程，也可以針對流程內的步驟及轉移條件做各別的定義。

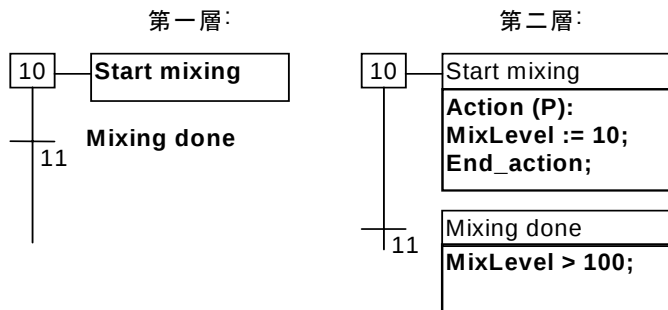
A.4.1 SFC 語言主題

SFC 語言是用來表示順序式的程序，它將流程區分為多個定義好的連續步驟，各個步驟以轉移條件分開。參考 ISaGRAF 語言參考手冊，可得到更詳細的說明。

SFC 元件以有方向性的線來連結彼此，內定的方向是由上到下。基本的 SFC 語言圖形元件為：

	 初始步驟
	步驟
	轉移條件
	跳躍至步驟
	巨集步驟
	巨集開始步驟
	巨集結束步驟

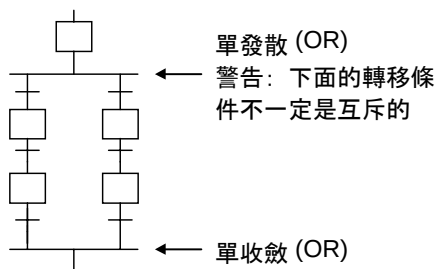
SFC 在設計程式時分為兩個不同的層次：**第一層 (Level 1)** 列出流程圖形、步驟及轉移條件的參考號碼以及步驟與轉移條件的註解，**第二層 (Level 2)** 表示步驟 (使用 **ST** 或 **IL** 語言) 的行為或轉移條件的條件，行為或條件可以呼叫其他語言 (**FBD**、**LD**、**ST** 或 **IL**) 寫的副程式。下面是第一層與第二層的例子：



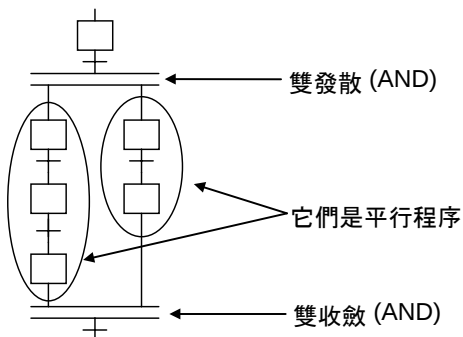
步驟的第二層程式可以於文字編輯器中輸入，包含以 ST 或 IL 語言寫的行爲方塊，而轉移條件的第二層程式除了可用 ST 與 IL 語言外，也可以使用 Quick LD 語言來撰寫。

發散與收斂

發散與收斂是用來表示步驟與轉移條件間的**多連結**，單發散或單收斂表示不同副流程間的不同走向。

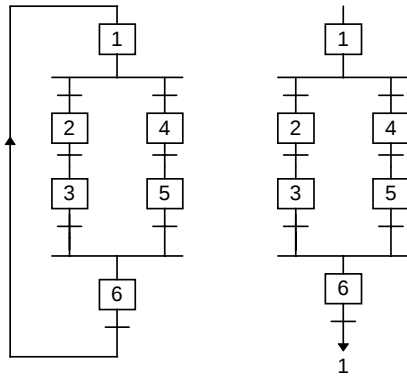


雙發散表示**平行**程序。



跳躍至步驟

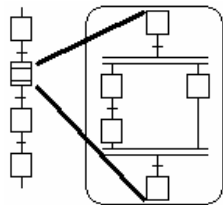
SFC 編輯器僅允許由上到下的連結，**跳躍 (jump)** 至步驟可用來表示與上面步驟的連結。下面是兩個相等表示法：



跳躍至轉移條件是不合法的。

巨集步驟

巨集步驟 (marco step) 是另外**單獨**造出的步驟及轉移條件的**單一**表示方式，巨集步驟開始於**開始步驟** (beginning step)，結束於**結束步驟** (ending step)。



巨集的內容須另外造於相同的 SFC 程式內，巨集步驟的**參考號碼**得與巨集開始步驟的參考號碼相同，巨集內容內也可以包含另一巨集。

A.4.2 SFC 流程圖形輸入

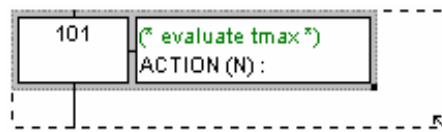
要畫出一個 SFC 流程圖，使用者只要使用它所提供的流程圖元件即可，各元件之間 SFC 會自動以水平線或垂直線連接它們。要插入一個流程元件，先選擇編輯器工具列上的圖示，在你要插入的位置按滑鼠左鍵即可。欲插入 SFC 元件至圖形中，使用者必須移動選擇區至適當的位置，然後選擇編輯工具列上的元件型態，則符號將被插入至現在的位置。下面的元件亦可使用鍵盤來輸入：

- F2:  插入初始步驟
- F3:  插入步驟
- F4:  插入轉移條件
- F5:  插入跳躍至步驟
- F6:  F7:  插入 OR 發散或收斂 / 加入分支
- ⇧F6:  ⇧F7:  插入 AND 發散或收斂 / 加入分支
- F8:  插入巨集步驟
- F9:  ⇧F9:  插入巨集步驟本體的開始或結束步驟

(“⇧” 符號表示使用 SHIFT 組合鍵)

編輯輔助格點顯示出可編輯的位置，使用者可選擇顯示或隱藏它。輔助格點在 SFC 流程元件初始佈局時相當有用。使用“**選項 / 外觀**”命令來顯示或隱藏輔助格點。

ISaGRAF SFC 編輯器總是顯示目前編輯的位置。被選取的位置會被標示成灰色。你可以用小方格的右下角來自由地放大縮小編輯位置，並可改變編輯位置 X/Y 維度的比例關係。



⇨ 製造發散或收斂元件

使用者指南

發散及收斂元件總是**從左畫到右**。要畫出一個發散及收斂元件，它的**左轉角**必須放置在流程圖的區域中，繪圖的型態（單或雙）由選擇工具列上的按鈕來設定。

  插入 OR 發散或收斂 / 加入分支

  插入 AND 發散或收斂 / 加入分支

＝ 加入發散分支

每一個**輔助分支**的**開始與停止**位置可以使用工具列上的按鈕，放置在發散及收斂線上，在插入新分支之前，發散或收斂的左轉角必須已存在，右轉角將會與主要的左轉角有相同的型態（單或雙），在主要的左轉角尚未加入之前，右轉角無法被放置。

  插入 OR 發散或收斂 / 加入分支

  插入 AND 發散或收斂 / 加入分支



插入巨集步驟

這個按鈕用來於主流程內插入巨集步驟，巨集步驟的本體必須另外造於相同 SFC 程式的某處。





巨集步驟的本體

巨集步驟必須另外定義於相同 SFC 程式主流程內，它開始於**開始步驟**，結束於**結束步驟**，且於主流程內的巨集步驟與它的開始步驟，兩者的**參考號碼**必須一樣。

A.4.3 編輯存在的流程圖

你可使用滑鼠或鍵盤的箭頭（方向）鍵在 SFC 流程圖中選取一個長方形的區間。所有被選取的區間會標示成灰色。然後，“**編輯**”功能表的命令將會作用在此區間上：



剪下 / 複製 / 刪除 / 貼上命令

當工具列上的“箭頭”按鈕被選擇時，可從“編輯”功能表利用如下的命令：

剪下 將選擇區間搬移到 SFC 剪貼簿

複製 將選擇區間複製到 SFC 剪貼簿

清除 清除 (刪除) 選擇區間

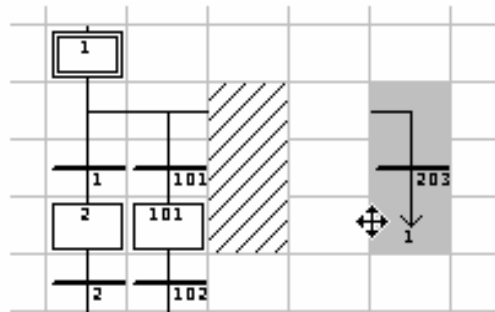
貼上 在目前的位置插入 SFC 剪貼簿的內容

“編輯 / 貼上”命令複製 SFC 剪貼簿的內容到螢幕上，複製與貼上命令可適用在 SFC 流程圖與步驟 / 轉移條件的第二層程式上，他亦可以複製一個程式的流程圖到另一個 SFC 程式內。選擇貼上的命令後，將會把 SFC 剪貼簿的內容插入在你目前所選取的位置之前。



移動元件

當 SFC 流程圖中的元件被選取後，你就可以用滑鼠左鍵拖曳它們，將它們移到 SFC 流程圖中的另一個位置上。當你拖曳這個選取區域時，被選取元件的初始位置會被畫上斑馬線記號。



元件欲移往的目的區域必須是空白的，它不會做插入的動作。

重編步驟與轉移條件的參考號碼

在 SFC 流程圖裡，每一個步驟與轉移條件都有一個邏輯號碼，“編輯 / 重新編號”命令允許使用者將所有的步驟及轉移條件自動編排成連續的號碼。當

使用者指南

步驟的號碼改變後，所有跳躍至此步驟的編號會自動更新（這項功能亦會應用到巨集步驟與它的開始步驟號碼上）。

➔ 直接跳至步驟或轉移條件

“編輯 / 跳至” 命令允許使用者進入已存在的步驟或轉移條件，螢幕會自動捲動至欲編輯的步驟或轉移條件位置處。

⇨ 尋找與取代字串

“編輯 / 尋找取代” 命令能用來尋找或取代程式內（所有的步驟及轉移條件）的文字字串，尋找 / 取代對話框能用來輸入欲尋找的文字字串，並且會直接開啓含有此文字字串的第二層程式。

A.4.4 輸入第二層程式

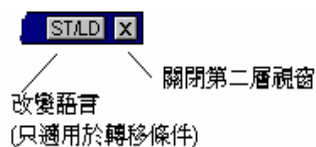
欲輸入第二層的程式，使用者必須於步驟或轉移條件符號上雙按滑鼠左鍵，第二層程式將出現於 SFC 視窗的右邊。SFC 流程圖與第二層程式之間的分隔線可以自由的移動。

你可以一次打開一個或兩個第二層程式區。下面的命令可以從鍵盤、滑鼠或“編輯”功能表下達：

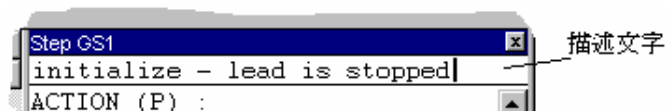
	鍵	滑鼠	“編輯”功
	盤		能表
開啓於上次內定視窗	E	雙按左	編輯第二
	nt	鍵	層
	er		
開啓於分離視窗	Ct	Ctrl+ 雙	編輯第二層於分離
	rl	按左鍵	視窗
	+		
	E		
	nt		
	er		

當同時使用兩個第二層編輯視窗，它們之間的分隔線可以自由移動。位於第二層標題欄右邊的按鈕可用來關閉此視窗。

第二層程式的內定語言為 **ST** (Structured Text，結構化文字)，轉移條件的第二層程式也可以用 **Quick LD** 編輯器來輸入。使用第二層視窗標題列的“**ST/LD**”按鈕可以改變撰寫的語言，這個命令僅有當第二層視窗內容是空的時候才有效。



第二層視窗的頂端會出現單行編輯框，可用來輸入簡短的描述文字，這些文字為 **SFC** 符號的註解。這些文字可用於“跳至...”等命令，或列印時當作 **SFC** 步驟及轉換條件的文件。



當第二層視窗開啓時，“**選項 / 更新**”命令可隨時更新變更的第二層程式至 **SFC** 主流程圖內。

插入變數名稱

當使用文字語言來設計程式時，按這個按鈕來選擇已於字典中宣告的變數，它會將所選的變數名稱插入到游標所在的位置。當使用 **Quick LD** 語言來設計程式時，按這個按鈕來將選擇的變數依附到接點或功能方塊的 **I/O** 參數上。

於步驟中輸入脈衝行為 (Pulse action) 區塊

當設計步驟的第二層程式時，使用這個按鈕於游標處插入脈衝行為區塊。下面是脈衝行為區塊的格式：

使用者指南

Action (P) :

ST statement;

...

End_Action;

當步驟處於執行中，脈衝行為內的指令僅於開始時執行一次。參考 ISaGRAF 語言參考手冊，可得到關於 SFC 程式更詳細的說明。

N 於步驟中插入非儲存行為 (Non stored action) 區塊

當設計步驟的第二層程式時，使用這個按鈕於游標處插入非儲存行為區塊。

下面是非儲存行為區塊的格式：

Action (N) :

ST statement;

...

End_Action;

當步驟處於執行中，非儲存行為內的指令會在每個 PLC 週期皆執行一次。參考 ISaGRAF 語言參考手冊，可得到關於 SFC 程式更詳細的說明。

P0 P1 新的 P0 及 P1 行為修飾語

ISaGRAF 支援新的 **P0** 及 **P1** 行為修飾語，於撰寫步驟的第二層程式時，按這些按鈕將 P0 或 P1 行為區塊文字插入到游標所在的位置。文字方塊格式如下：

Action (P0) : Action (P1) :

ST statement; ST statement;

... ...

End_Action; End_Action;

P1 行為指示當步驟變成執行時，方塊內的指令才執行一次（跟脈衝行為一樣），**P0** 行為指示當步驟變成非執行時，方塊內的指令才執行一次。參考 ISaGRAF 語言參考部分，以得到 SFC 程式設定的詳細說明。

⇒ 布林行為

另有一些語法可應用在布林變數上，這些行為會根據步驟的執行狀態套用在內部或輸出屬性的布林變數上。下面是基本的布林行為的語法：

`<boolean_variable> (N);` 指定變數值為步驟執行狀態
`<boolean_variable>;` 同上述指令相同效果
`/<boolean_variable>;` 指定變數值為步驟執行狀態的反相

當步驟變成執行中時，另一些語法可用來設定或重置布林變數。下面是設定與重置的布林行為語法：

`<boolean_variable> (S);` 當步驟執行狀態變成真 (TRUE) 時，設定變數為真 (TURE)
`<boolean_variable> (R);` 當步驟執行狀態變成真 (TRUE) 時，重置變數為假 (FALSE)

□ SFC 行為

另有一些語法可用來控制 SFC 子程式的執行。一個 SFC 行為是指根據步驟的執行狀態來開始或停止 SFC 子程式流程。可以有 **N** (非儲存)，**S** (設定)，或 **R** (重置) 等行為。下面是基本的 SFC 行為的語法：

`<child_program> (N);` 當步驟狀態變成執行時，開始子程式流程；當步驟狀態變成非執行時，停止子程式流程
`<child_program>;` 與上述的行為一樣
`<child_program> (S);` 當步驟狀態變成執行時，開始子程式流程
`<child_program> (R);` 當步驟狀態變作執行時，停止子程式流程

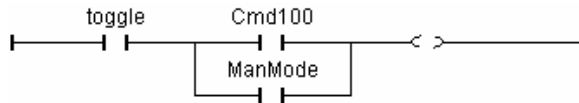
SFC 行為中的子程式必須是已經由 ISaGRAF 程式管理員所產生的 SFC 子程式。

□ 以 ST 語言來設計轉移條件

轉移條件的第二層是一個布林表示式，欲以 ST 語言來撰寫，只要根據 ST 語法來輸入布林條件式。你可以在表示式的結尾加上分號，也可以不加。

□ 以 Quick LD 語言來設計轉移條件

你也可以用 Quick LD 編輯器來輸入轉移條件第二層的條件。這樣的方法僅可以有一個導軌 (rung) 及一個輸出線圈 (coil)。輸出線圈不需要連結轉移條件名稱。下面是以 Quick LD 撰寫的轉移條件條件：



當以 Quick LD 來撰寫時，使用鍵盤上的方向鍵來移動游標至所要的位置上，然後使用下面的快捷鍵插入符號：

F2： 插入接點於所選的符號之後 / 導軌初始化

F3： 插入接點於所選的符號之前

F4： 插入一個與所選符號平行的接點

F6： 插入功能方塊於所選的符號之後

F7： 插入功能方塊於所選的符號之前

F8： 插入一個與所選符號平行的功能方塊

你亦能點選第二層視窗底端的功能鍵列，來代替鍵盤上的功能鍵。

當游標位於接點或功能方塊的輸出入參數上，按 **ENTER** 鍵來選擇所要連結的變數或輸入常數值。當游標位於功能方塊上，按 **ENTER** 鍵來選擇功能方塊的型態，你亦可以在符號上雙按滑鼠左鍵，也可達到相同的效果。

當游標位於接點上時，使用空白鍵來改變接點的屬性 (直接、反相或脈衝偵測)。參考本手冊的“使用 Quick LD 編輯器”章節，可得到關於 Quick LD 能力的詳細說明。

A.4.5 使用 SFC 倉庫

ISaGRAF SFC 編輯器管理一個 SFC 倉庫：收集 SFC 構造，且可以插入於任何 SFC 圖形中。SFC 倉庫元件可以選擇將步驟及轉移條件的第二層程式放入。使用下列的“工具”功能表命令：

複製至 SFC 倉庫複製所選的元件至 SFC 倉庫中
從 SFC 倉庫貼上貼上 SFC 倉庫元件於目前選擇區上

當複製至 SFC 倉庫時 (譬如：產生新的 SFC 倉庫元件)，你可以選擇要求一併放入所選 SFC 符號之第二層程式。

A.5 使用流程圖編輯器

ISaGRAF 流程圖 (Flow Chart) 圖形編輯器允許使用者輸入完整的 FC (流程圖) 程式，它的動作 (action) 及測試 (test 或決定, decision) 的程式可以使用 ST、IL 或 Quick LD 語言來撰寫。流程圖是一種決定性的圖形，可以用於描述循序式的操作，它加強一些諸如非凍結 (non-blocking) 向後跳躍等的進階元件。

A.5.1 基本的 FC 語言

流程圖 (FC) 是一種圖形式語言，用於描述循序式的運作。流程圖圖形由動作與測試所組成，動作及測試之間以有方向的連結線連結，表示資料流向。下面是流程圖語言的圖形元件：



FC 圖形開始：“開始”符號必須出現於流程圖程式的開始處，它是唯一的，且無法被刪除，表示當此程式執行時的初始狀態。



FC 圖形結束：“結束”符號必須出現於流程圖程式的結束處，它是唯一的，且無法被刪除，亦可能沒有連結線連到它 (迴圈圖形)，但是“結束”符號依然得畫於圖形的底端。“結束”符號表示圖形的最終狀態。



FC 流程連結：流程連結表示圖形兩點流向的連結線，連結線的尾端以箭頭為結束。兩個連結線不能開始於相同的連結點。



FC 動作：動作符號表示實行的指令，上面會標上編號及名稱作為識別。同一個流程圖中的不同物件不可以具有相同的名稱或邏輯號碼。動作的設計語言可以是 ST、LD 或 IL。動作總是與連結線連結，一個連結至它，一個從它開始連結至其他物件。



FC 測試：測試表示布林條件 (condition)，上面會標上編號及名稱作為識別。根據所依附的 ST、LD 或 IL 表示式的結果，連結流向可以指向“是”

或“否”兩個路徑。當以 ST 文字來設計時，表示式最後可以加上分號，也可以不加。當以 LD 來設計時，唯一的線圈表示條件值。

 **FC 副程式：**系統可以以 FC 程式的垂直架構來表示，這些 FC 程式以樹狀體系方式組成，每一個 FC 程式可以呼叫其他的 FC 程式。被呼叫的 FC 程式稱為呼叫程式的子程式，而呼叫的 FC 程式稱為父程式。FC 程式使用“父－子”的關係連結成結構樹。流程圖中的副程式符號表示呼叫流程圖副程式的關係。呼叫的 FC 程式會產生中斷，直到副程式執行完成之後才會再繼續執行。

 **FC I/O 特定動作：**I/O 特定動作 (specific action) 符號表示一種動作。如其他的動作一樣，I/O 特定動作以編號及名稱做為識別。I/O 特定動作與標準動作的意義一樣，只是 I/O 特定動作的目的是使流程圖形更具有可讀性，且用於圖形的非可攜性部分。I/O 特定動作是選擇性的元件。I/O 特定方塊與標準動作一樣，具有相同的行為。

 **FC 連結器：**連結器 (connectors) 用於表示圖形中兩點的連結關係，而不需直接繪出它們之間的連結線。連結器以圓圈表示，且必須連結到其他的圖形。連結器必須加上目標點做為識別（一般為目標符號名稱，此名稱必須根據資料流的方向，放置於適當的位置）。連結器總是以已定義的流程圖符號為目標，這個目標符號以它的邏輯號碼為識別。

 **FC 註解：**註解方塊包含一些與圖形語意無關的文字，它能被插入於流程圖視窗的任何空白處，以做為這個程式的文件。

A.5.2 輸入流程圖形

欲輸入圖形，你必須於圖形區域內放置元件（動作、測試、連結器..），且將它們連結起來。



插入物件

使用者指南

欲於圖中插入物件，首先於工具列上選擇對應的按鈕，然後於你要插入的位置處點選滑鼠。你能放置元件於空白區，或於連結線上點選滑鼠，將元件插入於流程中。於連結線上插入元件僅允許頂端至底端的垂直連結。你能插入下述之基本元件：

-  動作 (以 ST、IL 或 Quick LD 語言做為設計程式)
-  I/O 特定動作 (特別的非可攜性動作)
-  測試 (以 ST、IL 或 Quick LD 語言做為設計程式)
-  連結器
-  呼叫 FC 副程式
-  註解 (描述文字)

ISaGRAF 流程圖編輯器亦提供基本的流程圖結構，這些結構僅能插入於已存在的流程連結上，而無法放置於空的區域上：

-  If / Then / Else (二元選擇)
-  Repeat until (等待條件成立)
-  While (條件成立時的迴圈)

選取物件

大部分的編輯命令皆需要先選取物件。ISaGRAF FC 圖形編輯器可以讓你於圖形區內選取一個或多個已存在的物件。選取物件之前，必須先選取編輯工具列上的“**選擇**”按鈕 (箭頭符號)。欲選取一個物件，使用者僅需於符號上點選滑鼠即可。

欲選取物件集，可於圖形區使用滑鼠拖曳出一長方形區域，所有於選擇區內的圖形物件都會被標上“**被選取**”的記號。

被選取的物件會以藍色來表示，圖形符號旁加上小黑方塊。你可以加上 **Shift** 或 **Ctrl** 鍵來加入或移除選取的物件。

若重新選取新的物件，則先前選取的物件將不再被選取。若想不選取任何物件，只要於空白處點選滑鼠即可。

對於只選取單一物件，你也可以使用鍵盤之方向鍵，來選取流程圖上的另一物件。流程連結亦可以被選取。



插入註解

註解可以插入於圖形中的任何空白處。註解對於程式的執行沒有任何影響，只是提高程式的可讀性而已。欲插入註解方塊，於工具列上選取相對的按鈕，然後於所欲放置的位置上點選滑鼠。於註解上雙按滑鼠左鍵可輸入或改變註解文字。當你輸入註解方塊的文字時，並不需要加上如“(”與“)”等開頭或結尾字元。當註解方塊被選取時，你可以拖曳它邊界上的轉角來改變註解方塊的大小。



繪出流程連結

選取工具列上這個按鈕，可畫出已存在元件之間的連結線。連結必須遵循流向來畫出。首先選取 FC 元件的未連結輸出點，然後拖曳滑鼠至目標點上，便可插入連結線。目標點可以是 FC 元件未連結的頂點（輸入點），也可以是已存在連結線的任何位置。連結間的收斂點以小的灰色圓圈表示。收斂點可以被選取且加以搬移，以便於排列圖形。

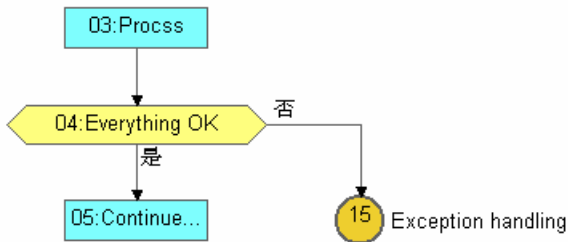


使用連結器

ISaGRAF 流程圖編輯器讓使用者可以使用圖形連結器，以取代可見的流程連結。連結器能避免太長的連結，且能增進圖形的可讀性。但是連結器不能用來建立與另一 FC 程式之連結。

連結器就如同其他物件一樣被放置於圖形中，以包含目標參考號碼（與目標的流程連結）的圓圈來表示。目標元件的簡短描述可以放置於連結器旁邊。

使用者指南



移動物件

欲於圖形中移動物件，你必須先選取它們，且拖曳滑鼠，來將它們搬移至圖中它處。你可以搬移單一元件，也可以搬移多重物件。當搬移元件時，你不能跟其他物件重疊在一起。移動物件不能用於將它們連結到已存在的連線線上。

當單一元件（動作、測試...）被搬移時，ISaGRAF 流程圖編輯器自動搬移其下之所有物件及連結線。此種特性並不適用於多重選擇物件上。



縮放物件

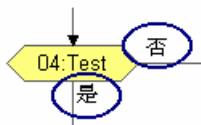
除了“開始”、“結束”符號及連結器之外的任何圖形元件都可以加以縮放。欲縮放一個元件，首先你必須選取它，然後以滑鼠拖曳此物件邊界上之小方塊，來改變它的大小。

當元件已具有連結線時，水平縮放它，將會同時改變左右兩邊界的大小，如此連結線仍可位於元件中央。



交換測試輸出

你可以交換測試的 是/否 輸出位置，想要這樣做，只要於測試符號旁之“是”或“否”記號上雙按滑鼠左鍵即可。



A.5.3 編輯已存在的圖形

“**編輯**”功能表的命令用於改變或完成已存在的圖形，這些命令大部分作用在目前選取的元件上。

▢ 修正圖形

DEL 鍵可以用來刪除所選取的元件，上面的連結線會一併被刪除。使用“**編輯 / 回復**”命令復原刪除命令所刪除的元件。刪除命令亦可用於多重選取物件上。“**編輯**”功能表下的“**剪下**”、“**複製**”、“**貼上**”命令用於移動或複製所選取的元件。

▢ 尋找及取代

“**編輯 / 尋找，取代**”命令可用於尋找或取代程式中的文字字串（以 ST、IL 或 Quick LD 為程式之所有動作及測試）。尋找 / 取代對話框可用來輸入欲搜尋的文字，找到之後，會自動開啓此文字的程式區。

➡ 直接進入元件

“**編輯 / 跳至**”命令允許使用者編輯已存在之圖形元件。視窗編輯區會自動捲動至目的元件處，然後選取此元件。

元件編號

“**編輯 / 重新編號**”命令用來重新編排流程圖元件的編號，任何放置於圖形中的 FC 元件都會編上一個唯一的參考號碼。當新元件插入至圖形中時，編輯器都會自動給予它一個參考號碼，“**重新編號**”允許你根據元件在圖中的位置，重新調整它的參考號碼。號碼會從上到下，從左到右依序編排下去。

A.5.4 輸入第二層程式

欲輸入第二層程式，使用者必須於動作或測試符號上雙按滑鼠左鍵，第二層程式將出現於 FC 視窗的右邊，FC 圖形與第二層程式之間的分隔線可以自由

使用者指南

的移動。你可以一次打開一個或兩個第二層程式區。下面的命令可以從鍵盤、滑鼠或“編輯”功能表下達：

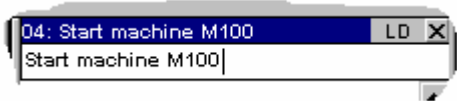
	鍵盤	滑鼠	“編輯”功能表
以上次內定視窗開啓	Enter	雙按左鍵	編輯第二層原始碼
以不同視窗開啓	Ctrl+Enter	Ctrl+雙按左鍵	於不同視窗中編輯第二層原始碼

當同時使用兩個第二層編輯視窗，它們之間的分隔線可以自由移動。位於第二層標題欄右邊的按鈕可用來關閉此視窗。

內定的第二層程式語言為 **ST** (Structured Text，結構化文字)，亦可以使用 **IL** 或 **Quick LD** 語言來撰寫。所選擇的語言會顯示於第二層標題列的小方格內。從功能表內執行“**選項 / 設定第二層語言**”命令或於點選小方格，可改變撰寫的語言，這個命令僅有當第二層視窗是空的時候才有效。



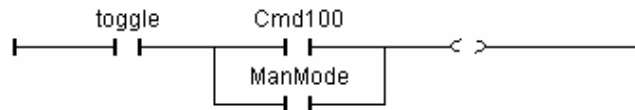
第二層視窗的頂端會出現單列的編輯框，可用來輸入簡短的描述文字，這些為 FC 符號的註解。這些文字可用於“跳至...”等命令，或列印時當作 FC 動作及測試的文件。



當第二層視窗開啓時，“**選項 / 更新**”命令可隨時更新變更的第二層程式至 FC 主圖形內。

A.5.5 以 Quick LD 設計第二層程式

Quick LD 編輯器可用於設計第二層的程式。若是作為測試的程式，LD 圖形僅由一個導軌（僅有一個線圈）所組成，表示測試的決定 (decision)。測試的名稱不可與線圈符號重覆。下面是以 Quick LD 設計測試的例子：



當以 Quick LD 設計程式時，使用鍵盤的方向鍵移動選擇區，然後以下述之快捷鍵來插入符號：

- F2 :插入接點於選取符號 / 初始導軌之後
- F3 :插入接點於選取符號之前
- F4 :插入與選取符號平行的接點
- F5 :加入與選取符號平行的線圈 (不可用於測試)
- F6 :插入方塊於選取符號之後
- F7 :插入方塊於選取符號之前
- F8 :插入與選取符號平行的方塊
- F9 :加入與選取線圈平行的跳躍符號 (不可用於測試)

跳躍會衍生出一個導軌名稱。當選擇區位於導軌的頭時，可以按 **ENTER** 鍵來輸入此導軌的名稱。ISaGRAF 編輯器記錄所有輸入的導軌標籤，不管它有沒有指定給導軌或跳躍運算元。“跳躍 / 標籤”對話框讓使用者輸入新的標籤，或選擇已存在的標籤，假如你輸入新的名稱，它將自動加入到表列中。“**移除**”按鈕可用來移除表列中選取的名稱，但它並不會移除所選導軌的標籤，若要如此，則只要使編輯欄內為空的，然後按**確定**即可。

你亦可以點選 LD 工具列上的按鈕，取代按功能鍵。

當選擇區位於接點或方塊的 I/O 參數上時，可按 **ENTER** 鍵，選擇變數或輸入常數值。當選擇區位於功能方塊上時，可按 **ENTER** 鍵選擇功能方塊的型態。你亦可於符號上雙按滑鼠左鍵，也可達到同樣的功能。

當接點選取時，可按 **Control + SPACE** 鍵來改變接點或線圈的型態。參考本手冊之“使用 Quick LD 編輯器”，以得到更多關於 Quick LD 能力的說明。

A.5.6 顯示選項

“**選項 / 外觀**”命令開啓一個對話盒，此對話盒群聚所有關於編輯空間及圖形繪製的參數及選項。使用“工作區間”群組內的檢查盒來顯示或隱藏編輯器工具列或狀態列。“文件”群組的選項允許你去顯示或隱藏編輯隔點，或以黑白或彩色來顯示圖示。



使用工具列上的“放大”按鈕來改變目前放大的比例。當編輯動作或測試所依附的 Quick LD 程式時，亦可使用這個命令來做縮放。



使用工具列上的“格點”按鈕來顯示或隱藏編輯輔助格點。當編輯動作或測試中的 Quick LD 程式時，此命令也可以使用。

使用“**選項 / 字型**”命令選擇用於所有 ISaGRAF 文件的字型。當使用 ST 或 IL 文字語言時，你可以指定字型大小。當使用圖形式語言時 (FC 或 Quick LD)，字型樣式及大小並不會改變圖形上的字型，所以此時並不需要加以指定，ISaGRAF 會自動根據目前的放大比例來計算字型大小。

A.6 使用 Quick LD 編輯器

LD 語言是布林表示式的圖形表示法，由這種圖形方式可以很明顯地表示出布林 AND、OR、NOT 等運算元。布林輸入變數連結到圖形式的接點上，而布林輸出變數則連結到圖形式的線圈上。ISaGRAF Quick LD 編輯器提供鍵盤與游標兩種方法，可以很便捷的方式來輸入階梯圖。元件會自動由 Quick LD 編輯器將之連結且排列至導軌上，使用者不需以手動的方式將接線加上。Quick LD 編輯器亦會排列圖形中的導軌，所以圖形內的空格將會保持最佳化狀態。

A.6.1 階梯圖語言基礎

階梯圖程式以包含接點與線圈的**導軌**集合來表示。下面是階梯圖的基本元件：

├─ 導軌的開頭 (左電力軌)

每一個導軌 (rung) 開始於左電力軌 (left power rail)，左電力軌表示初始的“真” (TRUE) 狀態。當第一個接點由使用者放置後，ISaGRAF Quick LD 編輯器自動產生左電力軌。每一個導軌可以有一個邏輯名稱，這個名稱是一個用於跳躍指令的標籤。

├─ 接點

接點 (contact) 會根據布林變數的狀態來改變布林資料流，變數名稱顯示於接點符號上。下面是 ISaGRAF Quick LD 編輯器所提供的接點型態：

- ├─ 直接接點
- ├─ 反相接點
- ├─ 正 (上升) 邊緣偵測接點
- ├─ 負 (下降) 邊緣偵測接點

├─ 線圈

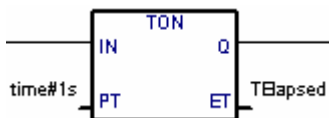
使用者指南

線圈 (coil) 表示一種動作，導軌的狀態 (線圈左方的狀態) 用來設定布林變數的狀態，變數名稱顯示於接點符號上。下面是 ISaGRAF Quick LD 編輯器所提供的線圈型態：

- { } 直接線圈
- {\} 反相線圈
- {S} “設定” 線圈
- {R} “重置” 線圈
- {P} 正 (上升) 邊緣偵測線圈
- {N} 負 (下降) 邊緣偵測線圈

功能方塊

它在階梯圖中以方塊表示，代表函數、功能方塊、副程式或運算元，它的第一個輸入和輸出參數總是連結到導軌上，其它的輸出入參數放置於矩形方塊外。



導軌的結束 (右電力軌)

導軌結束於右電力軌 (right power rail)。當使用 Quick LD 編輯器時，使用者放置線圈後，右電力軌會自動被插入。

跳躍符號

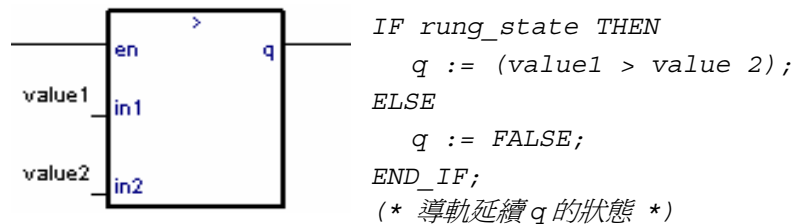
跳躍符號總是參考到一個導軌的標籤名稱，這個標籤名稱定義於相同的階梯圖中。跳躍符號放置於導軌的結尾。當導軌的狀態是真 (TRUE) 時，程式則跳躍至標籤所指示的導軌處執行。在使用向上跳躍時要特別注意，因為可能會產生無窮迴圈，使程式凍結住。

跳回符號

跳回符號放置於導軌的結尾，它會使得程式的執行停止於此。

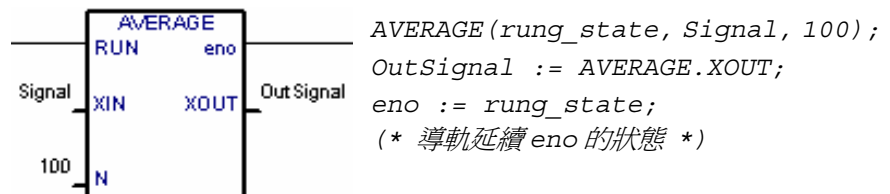
“EN” 輸入

在使用某些運算元、函數或功能方塊時，它的第一個輸入並不是布林的資料型態，而導軌總是連結至方塊的第一個輸入參數，因此系統會於第一個位置上自動插入一個稱為“EN”的輸入參數。這個方塊只有當 EN 的輸入為真 (TRUE) 時才會執行。下面是一個比較運算元以及使用相等作用的 ST 語言表示式的例子：

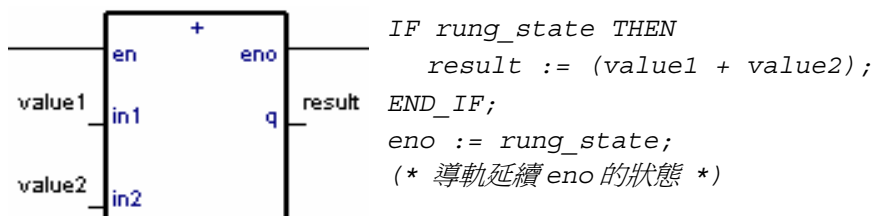


“ENO” 輸出

在使用某些運算元、函數或功能方塊時，它的第一個輸出並不是布林的資料型態，而導軌總是連結至方塊的第一個輸出參數，因此系統會於第一個位置上自動插入一個稱為“ENO”的輸出參數，它的狀態將會與第一個輸入參數狀態一樣。下面是一個做數值平均化的功能方塊以及使用相等作用的 ST 語言表示式的例子：



在某些情況下，EN 與 ENO 兩者皆需要插入。底下是一個算數運算元以及使用相等作用的 ST 語言表示式的例子：



HFO Quick LD 編輯器的限制

ISaGRAF Quick LD 編輯器不允許於線圈的右方繼續插入其他的導軌 (插入其他的接點或線圈)，若要在相同的導軌上造多個輸出，可以將他們造於平行的分支上。

A.6.2 輸入階梯圖

所有 Quick LD 編輯器的編輯命令可以使用鍵盤或滑鼠來完成。

編輯格點

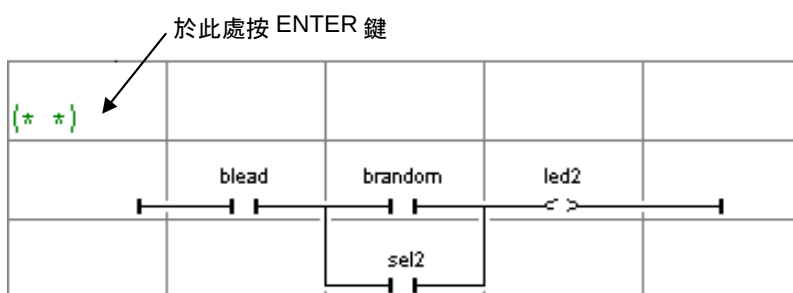
階梯圖元件必須輸入於邏輯矩陣內，每一矩陣單元可容納一個階梯圖符號。使用鍵盤上的方向鍵或點選所要的單元來移動現在的選擇區，被選到的物件以反白來表示，你也可以一次選擇多個物件，只要拖曳滑鼠或同時按住鍵盤上的 Shift 鍵與方向鍵即可。

□ 開始一個新的導軌

要將一個新的導軌加入圖形中，移動選擇區至上一次所造的導軌之後，插入一個接點 (按 F2 鍵或點選功能鍵工具列上相對應的按鈕) 後，將會產生包含接點與線圈的導軌。

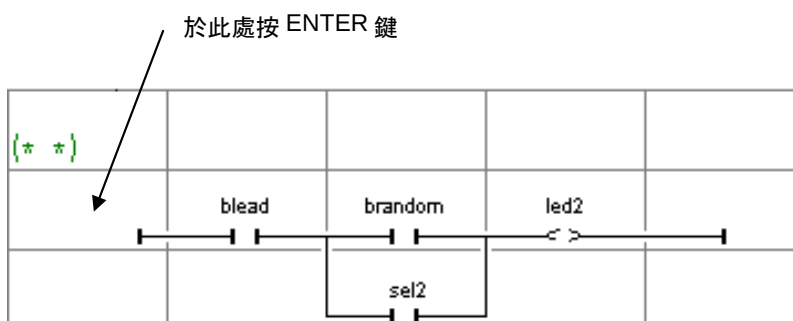
□ 輸入導軌的註解

每一個導軌可以加入一到兩行的文字註解，欲輸入導軌的註解文字，需移動選擇區至欲加註解的導軌上，然後按 Enter 鍵，或於註解區上雙按滑鼠左鍵：



輸入導軌的標籤

每一個導軌可以以名稱來做識別，這個名稱可搭配跳躍符號使用。要輸入或改變導軌的標籤，可移動游標至導軌的前頭，然後按下 Enter 鍵，或在上圖雙按滑鼠左鍵：



ISaGRAF Quick LD 編輯器將所有你所輸入的導軌標籤存於記憶體中，不管這個標籤是否有指定給導軌名稱或跳躍符號使用。“跳躍 / 標籤”對話盒讓使用者可以輸入新的標籤，或者直接選擇已存在的標籤名稱。

假如你輸入新的名稱，它會自動加到表列中。“移除”按鈕用來刪除表列中選取的標籤名稱，它並不會一併移除圖中對應的標籤。要移除圖上的標籤，只要清除編輯盒內的輸入文字，然後按確定按鈕即可。

於導軌上插入符號

要插入符號（接點、線圈、方塊...）到已存在的導軌上，你必須選擇一個位於導軌上的明確位置，然後按下下述的功能鍵來插入：

F2 插入位於所選的符號之前（左方）的接點

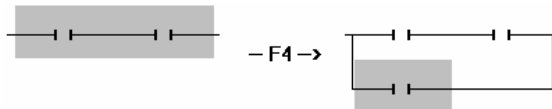
使用者指南

- F3 插入位於所選的符號之後 (右方) 的接點
- F4 插入與所選符號平行的接點
- F6 插入位於所選的符號之前 (左方) 的方塊
- F7 插入位於所選的符號之後 (右方) 的方塊
- F8 插入與所選符號平行的方塊

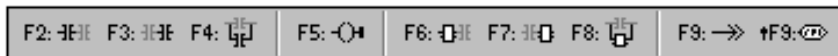
當所選的符號為線圈時，可以使用下述的命令：

- F5 加入與所選符號平行的線圈
- F9 加入與所選符號平行的“跳躍”符號
- Shift + F9 加入與所選符號平行的“跳回”符號

對平行的插入 (F4/F8) 來說，假如你一次選擇幾個接點，它會插入一個與這些元件平行的符號，如下面的例子所示：



要於圖中插入符號，你亦可以使用“**插入**”功能表下的命令，或用滑鼠點選位於視窗底部工具列上的功能鍵，選擇你所要的符號來插入：



輸入符號

要連結變數至接點或線圈上，可以選擇它後按 **Enter** 鍵。若於其上雙按滑鼠左鍵後，將開啓變數選擇視窗。參考手冊的“關於程式編輯器的更多說明”章節，可得到如何使用這個對話框的更多說明。要連結函數、功能方塊或運算元到方塊上，可以選擇它後按 **Enter** 鍵，要連結變數到方塊的輸出入參數上，則選擇方塊**外**的輸出入參數位置後，按 **Enter** 鍵。

對話框包含變數或方塊選擇列表，也可以用於文字輸入。假如“**選項**”功能表的“**手動鍵盤輸入**”模式被開啓時，變數符號及名稱直接輸入於純文字編輯盒中，輸入新的文字後按“**Enter**”鍵來確定，或按“**Escape**”鍵取消，

且關閉文字編輯盒。文字編輯盒於“手動鍵盤輸入”模式下不能以滑鼠來關閉。

改變接點與線圈的型態

“**編輯 / 改變線圈接點型態**”命令改變所選擇的線圈或接點的型態。接點可以是直接、反相、正或負邊緣偵測，而線圈可以是直接、反相、設定或重置、正或負邊緣偵測。按 SPACE 鍵也可以達到相同的效果。

⇒ 插入導軌於圖中

“**編輯 / 插入導軌**”命令在選擇的導軌之前插入新的導軌到圖中，這個導軌開始時會包含一個接點與線圈。

A.6.3 編輯已存在的圖形

“**編輯**”功能表下的命令用來修改或完成已存在的圖形，大部分的命令作用在選擇的物件上。

⇒ 修正圖形

Delete 鍵能用來刪除選擇的元件，但是你無法用它來刪除導軌上僅有的線圈、跳躍或跳回符號。使用“**編輯 / 回復**”命令來復原被 Delete 鍵所刪除的元件。Delete 亦可作用在多個選擇的元件上，也可以用在刪除選擇的導軌註解文字。當你選擇導軌的頭時，Delete 鍵會刪除整個導軌。

⇒ 複製元件

“**編輯**”功能表下的“**剪下**”、“**複製**”與“**貼上**”命令用來刪除或複製選擇的元件，這些命令無法作用在導軌的註解上。“**編輯 / 選擇性貼上**”命令讓你選擇欲插入的元件於：

所選元件的前面 (左邊)

所選元件的後面 (右邊)

與所選元件平行

▣ 管理整個導軌

假如你選擇導軌的頭（左電力軌），則所有的編輯命令（刪除、複製、剪下...）作用在整個導軌上，只要移動第一欄位的選擇區，就可以輕易的排列導軌。你亦可以垂直地擴展選擇區，讓它包含了幾個導軌的頭，這樣就可以將編輯命令應用在多個導軌上。

▣ 尋找與取代

“**編輯 / 尋找**”與“**編輯 / 取代**”命令用來尋找與取代圖形中的文字，僅有完整的名稱才能被找到。尋找命令可用來尋找接點、線圈、方塊名稱、方塊參數與導軌標籤，不能用來尋找導軌註解內的字串，取代命令也不能用來改變方塊的型態。尋找可以向前或向後尋找，當搜尋達到圖形界限時，則下次尋找時會重頭開始尋找。下面的快捷鍵可以用來快速地尋找變數名稱：

ALT+F2 尋找下一個相同名稱的元件，這個功能亦可以適用於功能方塊與導軌標籤上。

ALT+F5 尋找下一個相同名稱的線圈，這個主要用於除錯模式下，用來快速找出所懷疑的變數。

A.6.4 顯示選項

“**選項**”功能表下的命令用來設定螢幕上階梯圖顯示的狀態，以及顯示或隱藏訊息的某些型態。

▣ Tooltips

Use the "**Options / Tooltips**" command to hide or display variable comments appearing as tooltips in the whole diagram. The comment appears as a tooltip when the cursor moves over the corresponding variable block. This option is available in off-line and on-line modes.

▣ 導軌的註解

使用“**選項 / 導軌註解**”命令來顯示或隱藏導軌的註解。隱藏註解可以讓大的圖形更精簡，因為每個註解皆會佔去一列的編輯列。這個選項並不會改變已存在導軌註解的內容。它能在任何時間作切換（顯示或隱藏）。

⇒ 名稱及化名

連結到接點、線圈或功能方塊的輸出入參數上的變數，可以以它的名稱來作識別，ISaGRAF Quick LD 編輯器亦引進“**化名**”來代表每個變數，變數的化名就是變數註解第一個‘:’字元之前的字串，且化名限制在 16 個字元之內。如下的例子所示：

變數註解：化名：

short text short text

long text with no separator long text with n

short text: long description short text

化名對階梯圖的執行並沒有任何影響，你可以將它當作成一種註解。當你從變數集中選擇出變數名稱後，化名會自動從變數的註解中抽取出來，它無法以手動來改變。使用“**選項 / 接點與線圈**”命令選擇變數識別字的顯示模式。下面是可以利用的模式：

僅顯示變數名稱

僅顯示變數化名

顯示變數名稱及化名

當字典中的變數化名被改變時，Quick LD 編輯器不會自動地更新 LD 文件。使用“**選項 / 接點與線圈 / 更新化名**”命令來更新正在編輯的圖形的全部化名。你也可以從“**選項 / 接點與線圈**”功能中設定“**開啓時強制更新**”選項，要求 ISaGRAF 每次開啓 Quick LD 程式時自動更新所用到的全部化名。警告：設定此選項會很明顯地增加開啓程式的時間。

⇒ 繪圖選項

“**選項 / 外觀**”命令開啓一個對話盒，這個對話盒包含編輯工作區間及階梯圖繪出時的各種參數及選項。

使用者指南

使用“工作空間”群組內的核對方塊來顯示或隱藏編輯工具列、狀態列及 LD 工具列。“文件”群組的選項允許你顯示 / 隱藏編輯格點及致能 / 禁能繪圖顏色的使用。



“放大”群組的選項允許你選擇放大比例，你也可以點選編輯工具列上的“放大”按鈕來切換各種內定的放大比例。



你也可以設定自己的 X/Y 維度的比例關係，這個最後的選項能用來降低內定的格子寬度 (如果你都用短的變數名稱)，或使用編輯工具列上的“寬度”按鈕來改變 X/Y 維度的比例，而不需進入外觀對話框內設定。

使用“**選項 / 字型**”命令來選擇所有用在 ISaGRAF 圖形文件中的文字字型的名稱。選取字型的時候，字型的型態和大小不需加以指定，ISaGRAF 圖形編輯器會根據被選取物的放大比例來計算字型的大小。

A.7 使用 FBD/LD 編輯器

ISaGRAF FBD/LD 圖形編輯器允許使用者輸入部分階梯圖，以組成完整的 FBD 程式，它組合了圖形與文字的編輯能力，如此可以輸入圖形，也可以輸入相對應的輸入及輸出。這個編輯器比較致力於 FBD 語言，純粹的階梯圖建議最好使用 ISaGRAF Quick LD 編輯器來編輯。

A.7.1 FBD/LD 語言基礎

FBD 語言是許多不同型態程式的圖形表示法，**運算元**以長方形的函數方塊表示。函數的輸入連結到方塊的左邊，函數的輸出則連結到方塊的右邊。圖形式的輸入與輸出 (**變數**) 使用**邏輯連線**到函數的方塊上，函數方塊的輸出也可以連到另一個方塊的輸入。

LD 語言是布林運算式的圖形表示法，這種圖形方式可以很明顯地表示出布林 **AND**、**OR**、**NOT** 等運算元。布林輸入變數連結到圖形**接點**上，而布林輸出變數則連結到圖形**線圈**上。接點及線圈由**水平線**連結在一起，且連結到左右電力軌上。每一個線段皆會有**真 (TRUE)** 或**假 (FALSE)** 的布林狀態值，所有連結在一起的線段都有相同的布林狀態，任何連到左**垂直電力軌**上的水平線皆是**真 (TRUE)** 的狀態。

LD 與 FBD 圖形的程式執行順序皆是從左到右，從上到下。參考 ISaGRAF 語言參考手冊，可得到更多有關 LD 與 FBD 語言的更多說明。以下是 FBD/LD 編輯器所支援的基本 LD 及 FBD 語言元件：



左電力軌

導軌的左端必須連結到**左電力軌**上（表示初始的“真” (TRUE) 狀態），ISaGRAF FBD 編輯器亦允許連結任何的布林符號到左電力軌上。



右電力軌

使用者指南

線圈的右端可以連結到**右電力軌**上，當使用 ISaGRAF FBD/LD 編輯器時，這是可以選擇的。假如線圈的右方無右電力軌，它會自動產生一個右電力軌。

LD 垂直 “OR” 連結

LD 垂直連結接受一些左邊及右邊的連線，每一個右方連結的狀態相當於左方連結做 OR (或) 運算後的結果。

接點

接點會根據布林變數的狀態來改變布林資料流，連結的變數名稱顯示於接點符號的上方。下面是 ISaGRAF FBD/LD 編輯器所支援的接點型態：

- ┆┆ 直接接點
- ┆┆ 反相接點
- ┆┆ 正 (上升) 邊緣偵測接點
- ┆┆ 負 (下降) 邊緣偵測接點

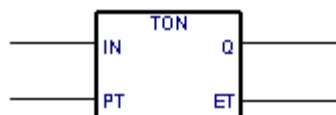
線圈

線圈表示一種動作，它的左端必須連結到布林符號 (如接點)，連結的變數名稱顯示於線圈符號的上方。下面是 ISaGRAF FBD/LD 編輯器所支援的線圈型態：

- ┆┆ 直接線圈
- ┆┆ 反相線圈
- ┆┆ “設定” 線圈
- ┆┆ “重置” 線圈

功能方塊

FBD 圖形中的方塊可以表示函數、功能方塊、副程式或是運算元，輸入及輸出必須連結到變數、接點、線圈或其它方塊的輸出或輸出點上，形式上的參數名稱顯示於矩形方塊內。





標籤

標籤可以放置於圖中的任何地方，用來當作跳躍指令的目的地，它可以改變程式的執行順序。標籤無法跟其它元件做連結。為了增加程式的可讀性，建議你最好將標籤放置於圖左方。



跳躍

跳躍符號總是參考到放置於圖中某處的標籤，它的左端必須連結到布林點上，當左端的狀態為真 (TRUE)，程式會直接跳到目的標籤所在的程式執行。使用向後跳躍時要特別注意，因為可能會產生無窮迴圈，使程式凍結住。



返回符號

返回符號連結到布林點上，當導軌的狀態為真 (TRUE) 時，程式的執行將被停止。



變數

變數在圖中是以小長方形表示，它的左右兩端可以連結到其它元件。



連結

各元件間以連結線來表示連結關係，連結必須由輸出點到輸入點（資料流的方向）。



布林反相連結

某些布林的連結線會在線的最右端加上一個小圓圈，代表反相的布林資料流。



使用者定義的轉角

使用者定義點可以定義在連結線上，它允許使用者手動控制連線的外型，在沒有加上轉角的情況下，ISaGRAF FBD/LD 編輯器使用內定的連線軌跡規則。

A.7.2 輸入 FBD 圖形

要輸入 FBD 圖形，你必須放置元件（方塊、變數、接點、線圈...）於圖形區域內，然後將他們連結起來。



插入物件

欲於圖中插入物件，需點選工具列上的相對按鈕，然後於圖形區域內欲插入的位置按滑鼠左鍵。



選擇物件

大部分的編輯命令都需要先選取物件，ISaGRAF FBD/LD 圖形編輯器可以選取一個或多個存在於圖形區內的物件。欲選取物件，必須先點選編輯工具列上的“**選擇**”按鈕（箭頭形狀）。僅選一個物件時，點選該物件即可，欲選取多個物件時，於圖中拖曳滑鼠成長方形，與長方形區域交錯的物件將被標上“**被選取**”的記號，被選到的物件會在它的四週出現黑色的小方點。當選取新的物件後，所有先前選取過的物件將不再被選取，不要選取任何物件時，只要點選被選取物件長方形邊界外的空區域即可。



插入註解

註解可以插入於圖形中的任何區域，它不會影響程式的執行，但是可增加程式的可讀性。要插入註解方塊，選取工具列上的相對按鈕，拖曳滑鼠，選取註解所需的長方形區塊，然後輸入文字註解。當你輸入註解方塊內的文字時，不需要加上如“**(*)**”與“***)**”等的開頭及結尾字串。當註解方塊被選取時，可以拖曳它的邊框來改變註解方塊的大小。



移動物件

要於圖中移動物件，先選取它們，再移動滑鼠，將它們搬移到目的地。若移動的物件已做過連線，只要於移動圖形符號後，則 ISaGRAF FBD/LD 編輯器會根據移動後的新位置重畫各物件間的連結線。



繪出連結線

選取工具列上的相對按鈕來繪出已存在元件間的連結線，假如你從一個連結點畫到一個空的位置，在終端處會自動加入使用者定義轉角，這樣你可以再繼續畫出另一個區段。

改變連結線的畫法

當連結線被選取後，“工具 / 改變線形” 可用來改變連線的軌跡，但是若這個連結線連到使用者定義轉角，這個命令將沒有作用。假如連結線有三個區段，這個命令會改變第二區段的位置，如下所示：



改變連結線的型態

只要在連結線的極右端上雙按滑鼠左鍵，你能很容易第改變連結線的型態（有或沒有布林反相）。

繪出 LD 導線

欲畫一個新的 LD 導線，首先須插入左電力軌，然後放置一個線圈，它會自動連結到電力軌上，其它的接點及垂直 OR 連線可以直接加在這個導軌線上，不需要自己加入任何的連結線。

當加入新的 LD 接點或線圈到空的編輯區內，會在插入的元件與已存在的左右電力軌之間加入新的水平導線。假如新加入的元件沒有放置在電力軌之間，這條導線不會自動加上去。你可以在導軌上自由移動插入的接點或線圈。當插入的 LD 接點或線圈被刪除後，編輯器會補上水平線。你可以插入新的 LD 接點或線圈到已存在導軌的水平線上，編輯器會自動剪斷導軌，且將它連到新插入接點或線圈的左右連結點上。

多重連結

多重連結可以建立在任何輸出接點的右端，他表示將訊息廣播到幾個其它點上，每個接點的極右端點的狀態都相同。輸出接點所連出去的連結線數目是

使用者指南

沒有限制的，但是不能將兩個連結線的極右端點連到相同的輸入點上，除非是如下的LD 元件：

-  右電力軌
-  左 (OR) 運算元的多連結

這些LD 符號沒有輸入數目的限制。

A.7.3 編輯已存在的圖形

“**編輯**”功能表的命令用來改變或完成已存在的圖形，大部分的命令作用在被選取的物件上。

□ 修正圖形

DEL 鍵可以用來移除被選取的物件，連結到這些被刪除元件的連結線會一並被刪除。DEL 鍵執行後，可以用“**編輯 / 回復**”命令來復原被刪除的元件。DEL 命令也可以用在所選的物件群組上。“**編輯**”功能表下的“**剪下**”、“**複製**”及“**貼上**”命令用來搬移或複製所選的元件。

□ 尋找與取代

“**編輯**”功能表下的“**尋找**”與“**取代**”命令用來尋找與取代圖形中的文字，僅有完整的名稱才能被找到。尋找可以用在接點、線圈、方塊名稱、變數與標籤上，不能用來尋找註解內的字串，取代命令不能用來改變方塊名稱。尋找能從選擇的位置往前或往後搜尋，當搜尋至圖形區的邊界後，則重新從另一端再搜尋起。

□ 顯示執行順序

當 FBD 圖形包含向後跳躍的迴圈時，執行的順序就不再是單純的左到右 / 上到下的方式，爲了避免混淆，使用“**編輯 / 顯示執行順序**”命令或按 **Control+F1** 鍵來顯示執行順序（編譯時的順序），在會導致動作的符號（線圈、設定變數與功能方塊）附近標上 1 到 N 的數字。

輸入符號及文字

在元件上雙按滑鼠左鍵可輸入連結的符號或文字，這個方法可以用在變數、接點、線圈、註解文字及標籤上。當用在接點及線圈上時，可以改變它的型態（直接、反相..）。

對話框包含變數或方塊選擇列表，也可以用於文字輸入。假如“**選項**”功能表的“**手動鍵盤輸入**”模式被開啓時，變數符號及名稱直接輸入於單純文字編輯盒中，輸入新的文字後按“**Enter**”鍵來確定，或按“**Escape**”鍵取消，且關閉文字編輯盒。文字編輯盒於“手動鍵盤輸入”模式下不能以滑鼠來關閉。

假如“**選項**”功能表下的“**自動輸入**”模式被開啓，每次插入新的接點時，變數符號必須立即輸入；當插入變數或標籤時，符號也必須立即輸入。



選擇功能方塊型態

於方塊上雙按滑鼠左鍵可以改變它的型態，方塊的型態可以從可利用的運算元、函數與功能方塊等列表中選出，這命令亦允許改變輸入點的個數（如 AND, OR, ADD, MUL...）。

加入空白區

當你於 FBD 繪圖區中按滑鼠右鍵時，將顯示出一個彈出式功能表，它包含下述之命令，可以用來插入或移除滑鼠所在位置之空白區：

插入列： 這個命令會於彈出式功能表開啓處之滑鼠游標的起始位置插入水平的空白區（根據格點間隔計算出來的 4 列）。

刪除列： 這個命令會於彈出式功能表開啓處之滑鼠游標的起始位置刪除未使用的水平空白區（列）。此命令不能用來刪除 FBD 元件。

當彈出式功能表開啓時，FBD 繪圖區會出現一條灰色線來顯示即將插入或刪除空白區的位置。

A.7.4 顯示選項

“選項”功能表的命令用來設定螢幕上的 FBD 圖形的顯示。

□ 外觀

“選項 / 外觀”命令開啓一個對話盒，這個對話盒聚集有關編輯工作區間及圖形顯示的參數及選項。使用“工作區間”群組內的檢查方塊 (check box) 用來顯示或隱藏編輯工具列或狀態列，“文件”群組的選項允許你顯示或隱藏編輯格點。



“放大”群組的選項允許你選擇一個主放大比例，你亦能使用編輯工具列上的“放大”按鈕來切換內定的比例。

使用“選項 / 字型”命令來選擇所有用在 ISaGRAF 圖形文件中的文字字型的名稱。選取字型的時候，字型的型態和大小是不相關的，而且不需詳細地設定。ISaGRAF 圖形編輯器總是根據被選取的放大比例來計算字型的大小。

A.7.5 樣式及更動記錄

ISaGRAF LD/FBD 編輯器讓使用者可以指定任何 LD/FBD 元件的圖形樣式。樣式就是將圖形元件定義成特別的顏色，ISaGRAF 亦使用樣式來做圖形變更的記錄，以便於作不同版本間的控制。

注意於模擬或線上除錯模式下，樣式是不可見的，因為變數的 TRUE / FALSE 狀態會以紅色及藍色來表示。

□ 預定的樣式

下面的樣式已預先定義好：

正常 為內定的樣式 (黑色)。對變動記錄而言, "正常" 樣式表示此元件為原始圖形的一部份。在執行時, 具有 "正常" 樣式的元件將會加以執行。

變更 具有 "變更" 樣式的元件以粉紅色來標示。對變動記錄而言, "變更" 樣式用來突顯改變或加入原始圖形的元件。在執行時, 具有 "變更" 樣式的元件將會加以執行。

刪除 具有 "刪除" 樣式的元件以灰色虛線來標示。程式在執行時會忽略這些元件, 這種元件主要是用來記錄移除後的元件, 以作為版本控制之用。

自行設定 除了預定的樣式外, ISaGRAF LD/FBD 編輯器允許使用者自定圖形元件成任意的顏色, 這樣的元件則被當作具有 "自行設定" 的樣式。使用 "自行設定" 樣式對程式執行不會造成任何的影響。

使用 "編輯" 功能表下的 "樣式" 副功能表下的命令手動調整選取元件的樣式。

□ 變動記錄

可用 "刪除" 樣式來作變更記錄的追蹤, 或使用 "編輯/樣式" 功能表下的 "標記變更" 命令可用來開啓或關閉變更的記錄。

當 "標記變更" 選項被設定, 所有變動或加入的圖形元件都會自動設定成 "變更" 樣式, 當元件使用 "刪除" 或 "剪下" 命令刪除, 它們不會自圖形中刪除, 而是標上 "刪除" 的樣式。這樣的功能可以自動記錄圖形中所有變動過的元件。

使用 "編輯/樣式/移除所有刪除的項目" 命令, 移除 LD/FBD 圖形中所有標記 "刪除" 樣式的元件, 這個命令不管目前的選擇區, 而是作用在整個的圖形上。欲 "回復" 標記 "刪除" 樣式的元件, 選取此元件, 然後改變為 "正常" 樣式、"變更" 樣式或任何 "自行設定" 樣式即可, 這樣的操作可能會導致無效的連結 (多於一個的連線連至相同的輸入點), 這樣的情況會於下一次程式驗證時才會偵測出來。

A.8 使用文字編輯器

這一章節僅說明指定的元件及 ISaGRAF 文字編輯器的命令，尤其是用在輸入 ST 與 IL 程式的原始碼時的命令。

A.8.1 編輯命令

“**編輯**”功能表的命令用來編輯所輸入的文字，大部分的命令作用在於圖中所選的字串上，或作用於游標所在位置的物件。



剪下與貼上

DEL 鍵用來移除所選的文字，在 DEL 命令執行後，可以使用“**編輯 / 回復**”命令來復原刪除的元件。“**編輯**”功能表下的“**剪下**”、“**複製**”與“**貼上**”命令用來移除或複製程式內的文字，或者插入從其他應用程式來的剪貼簿內的文字。

□ 尋找與取代

“**編輯**”功能表下的“**尋找**”與“**取代**”命令用來尋找與取代程式內的文字，任何的字串都可以被找到。尋找能從游標所在的位置往前或往後搜尋，當搜尋到邊界後，並不會從另一端再搜尋起。

□ 跳至某行

“**編輯 / 跳至某行**”命令用來移動游標至指定的行數，當 ISaGRAF 編輯 ST 或 IL 程式有錯誤發生時，會以行數當作錯誤的參考，這個命令用來直接跳至該行是非常有用的。



從字典插入符號

使用“**編輯 / 插入變數**”命令用來插入案件字典中宣告的變數符號或物件到游標所在位置，注意這個命令僅能處理純 ASCII 文字檔案。

⇐ 插入檔案

“**編輯 / 插入檔案**” 命令插入全部的檔案內容到游標所在的位置，注意這個命令僅能處理純 ASCII 文字檔案。

A.8.2 選項

“**選項**” 功能表的命令用來顯示或隱藏工具列以及選擇字型，這裡所選的字型將會套用在所用的 ISaGRAF 工作平台的文字編輯器中。

當輸入 ST / IL 程式的原始碼時，“**選項 / 顯示關鍵字**” 命令可用來顯示或隱藏 ST 或 IL 語言最常用的識別字工具箱。點選工具列上的按鈕來插入相對的識別字或運算元到游標所在的位置。

A.9 有關程式編輯器的更多說明

本章包含所有有關 ISaGRAF 程式編輯器特色的資訊。這些資訊與 ISaGRAF 工具以及共通的 ISaGRAF 對話框連結有關。

A.9.1 呼叫其他 ISaGRAF 工具



驗證 (編譯) 程式

“**檔案 / 驗證**”命令會執行一個叫“ISaGRAF 碼產生器”(Code generator)的程式去驗證程式語法。如果是 SFC 語言，則第一層與第二層都會被驗證。當語法驗證完畢後，“碼產生器”視窗必須關閉以便繼續程式上的工作。如果沒有語法上的錯誤，而且應用中又只有一個程式，則應用碼便會被產生出來。“**選項 / 編譯選項**”命令是被用來設置 (set) 參數的編譯及最佳化時使用。參考“如何使用碼產生器”一章，您將會得到進一步有關編譯及碼產生器的訊息。



模擬或除錯

“**檔案 / 模擬**”以及“**檔案 / 除錯**”命令可以在模擬模式或真實連接模式中執行 ISaGRAF 圖形除錯器，且於除錯模式時，再將 SFC 程式開啓。於除錯模式時，你無法修改程式內容。



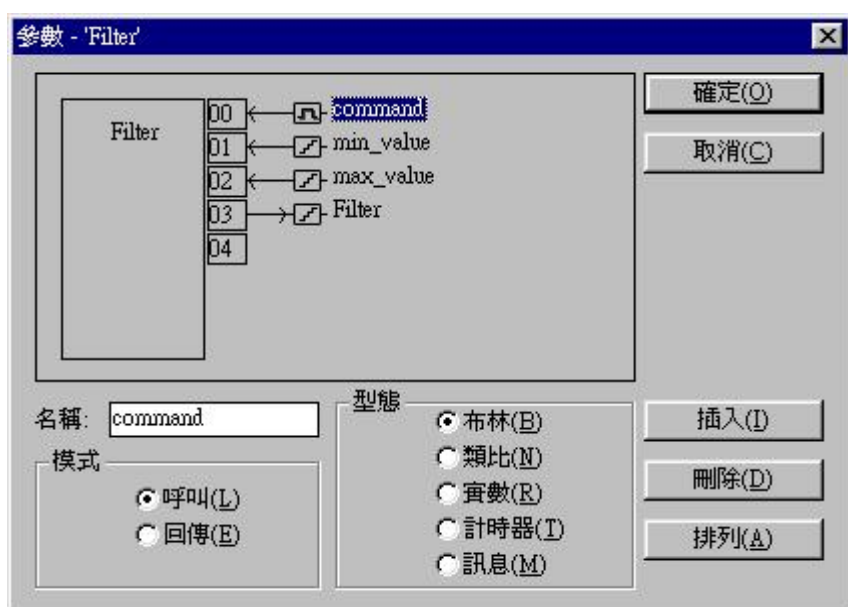
編輯變數字典

“**檔案 / 字典**”命令是用來編輯目前應用程式所用到的變數，它也可用來編輯使用者自訂的定義字。“**區域**”宣告或定義字只適用於目前編輯的程式中。

A.9.2 程式的參數

當編輯的程式為函數、功能方塊或副程式時，則“**檔案 / 參數**”命令被用來定義呼叫及回傳的參數。當程式為 SFC 或為**開始**、**結束**區中的程式時，這個命令無效。

副程式、函數或功能方塊最高可以有 **32** 個參數（包含輸入及輸出）。函數或副程式通常只有一個回傳參數，而且這個參數必須與函數名稱相同，以便符合 ST 語言的撰寫法則。下列對話框是被用來描述副程式的參數：



在視窗左上方的位置列出所有的參數且以呼叫模式的順序來排列。前面是“呼叫參數”，再來為“回傳參數”。在視窗的下半部顯示被選取參數的詳細描述，參數可以是任何一個 ISaGRAF 的資料形態。“回傳參數”必須位於“呼叫參數”之後。參數名稱必須符合下列規則：

名稱不能超過 16 個字元

第一個字元必須是英文字母

第一個字元以後的字元可以是字母、數字或底線字元

名稱不分大小寫

使用者指南

“**插入**”命令是用來在選取的參數之前插入新參數。“**刪除**”命令是用來刪除選取的參數。“**排序**”命令是用來自動重排列（排序）參數，所以“回傳參數”皆會放置於列尾。

A.9.3 “檔案”功能表中其它的命令

下列命令可用於所有程式編輯器內的“**檔案**”功能表中：



打開其它的程式

“**檔案 / 開啓**”命令允許使用者關閉目前編輯的程式，且開啓目前案件中相同語言的程式。這個功能不能用來編輯以另一種語言撰寫的程式。被選取的程式會替換目前在編輯視窗中編輯的程式。



程式列印

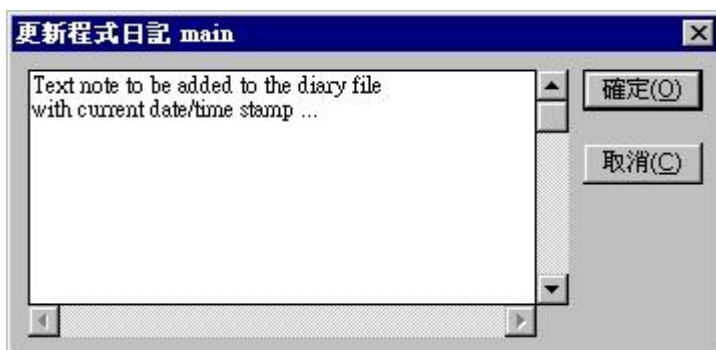
“**檔案 / 列印**”命令用來輸出程式至印表機。這個命令會自動執行 ISaGRAF 文件產生器來列印正在編輯的程式和相關的區域變數。

於圖形語言程式中（SFC、FBD 及 Quick LD），您可用“**編輯 / 複製圖形**”命令去拷貝在剪貼簿中以 metafile 格式所繪的圖形，以便於去剪貼於其他的應用中，例如文書編輯器“Word”。如果是 SFC 程式，僅有第一階的圖形（流程圖，數字編號以及第一層註解）會拷貝到 metafile 檔中。

A.9.4 更新程式日記 (program diary)

可以用“**檔案 / 日記**”命令將附帶於編輯程式的日記檔以手動方式輸入。每一次程式編譯時，這個日記檔會自動加上語法檢查輸出訊息。編譯輸出完成後會有一個時間 / 日期的蓋印。

☞ 如果程式編輯器“**選項**”功能表下的“**更新日記**”模式中被選取，則每次程式被儲存至硬碟時會出現下列對話框。



假如“確定”按鍵被按下，則所輸入的文字註解便會加上時間 / 日期而儲存至日記檔的末端。這個特色對於維護程式非常有用，因為它提供了程式生命週期的協助。

A.9.5 從字典中挑選變數

 當編輯文字式程式 (ST 或 IL) 時，“編輯 / 插入變數”允許選取一個已宣告的變數名稱，插入到游標所在的位置。當編輯 LD 或 FBD 程式時，線圈、I/O 參數或 FBD 變數方塊必須依附於變數。在兩種情況下，將開啓下列的對話框，以便選取已宣告的變數：



使用者指南

“範圍”列示盒是用來選擇全域或區域變數。列示盒的右邊可以選擇資料型態。在型態選擇欄的旁邊有一些小的圖示 (icon) 為選擇資料型態的快捷按鈕：



布林 (Boolean) 型態



整數 / 實數 (Integer / Real) 型態



計時器 (Timer) 型態



訊息 (Message) 型態

欲選擇變數，只要在名單上點選即可。它的名稱以及註解便會出現在名單正上方。然後按“確定”按鈕確認您的選擇，如此便完成了程序。當然，您亦可以直接將變數名稱輸入欄位中。

A.9.6 輸出視窗

所有的語言編輯器下的工具功能表皆可使用下述的命令，這些命令可用來於編輯視窗底端顯示訊息，且可用來瀏覽程式。

“顯示編譯輸出”於輸出視窗內顯示最後之程式編譯錯誤訊息。

“尋找...” 尋找所有編輯程式內的文字，且將之列於輸出視窗。對於 SFC 及 FC 語言而言，這個命令將搜尋所有的第二層程式。

“隱藏輸出視窗”關閉輸出視窗。

當錯誤訊息或找尋到的字串顯示於輸出視窗時，於其行列上雙按滑鼠左鍵，可直接移動畫面至對應的位置，且將之選取。對 SFC 及 FC 語言而言，這個命令將開啓對應的第二層程式視窗。

A.10 使用字典編輯器

ISaGRAF 字典 (dictionary) 是一個宣告內部變數、I/O 變數、功能方塊樣例以及定義字的編輯工具。字典集合了宣告變數、功能方塊樣例以及定義字。變數、功能方塊以及定義字必須在程式原始碼中使用前，先在字典中宣告它們。變數以及定義字能夠用於任一種語言上：SFC、FC、FBD、LD、ST 及 IL。功能方塊用於 FBD 語言上不需要加以宣告，因為 ISaGRAF 的 FBD 及 Quick LD 編輯器會自動宣告已用過的方塊樣例。

▣ 變數

變數依“範圍”和“型態”分類來儲存。僅有相同的“型態”和相同的“範圍”的變數可以在相同的編輯視窗中輸入。下列為變數基本的“範圍”：

-  **全域** 可以被目前案件中任何程式使用
-  **區域** 僅可以用於一個程式中

下列為變數基本“型態”：

-  **布林** 真假二進位值
-  **類比** 實數或整數值
-  **計時器** 時間值
-  **訊息** 字串

變數的資料由名稱、註解、屬性、網址以及其它特別指定之欄位所組成。以下為變數的基本屬性：

- 內部** 內部變數
- 輸入** 連接至輸入裝置的變數
- 輸出** 連接至輸出裝置的變數
- 常數** 唯讀內部變數 (有初始值)

注意：計時器永遠是內部屬性的變數。輸入以及輸出變數永遠具有全域範圍。

定義字

定義字是一個可以替代文字字串，而且用於任何語言中的“別名”。被替代的文字可以是一個變數名，或一個常數表示式或複雜的表示式。定義字依“範圍”分類。僅有相同“型態”及相同“範圍”的定義字可以在相同的編輯視窗中輸入。以下為基本的“範圍”：

-  **共用** 可以在任一案件的任一程式中使用
-  **全域** 可以在目前的案件中的任一程式中使用
-  **區域** 僅可用於單一程式中

定義字的資料是由名稱、ST 語法表示式以及註解所組成。

功能方塊樣例

在 ST 及 IL 語言中所用到的功能方塊樣例必須在字典中宣告，因為功能方塊具有隱藏的內部資料，所以每一個功能方塊的拷貝皆必須被識別。下列的範例表示在程式館中定義的功能方塊“R_TRIG”被用於做不同變數的邊緣偵測。每一個方塊的拷貝皆必須以一個唯一的名稱來識別。方塊型態名稱以及它的參數定義可以使用程式館管理員來完成：

方塊名稱：R_TRIG

參數：Input=CLK

Output=Q

樣例的命令可使用字典編輯器來完成：

樣例名稱：TRIG_B1 方塊名稱：R_TRIG

樣例名稱：TRIG_B2 方塊名稱：R_TRIG

宣告的樣例可以在 ST 程式中使用：

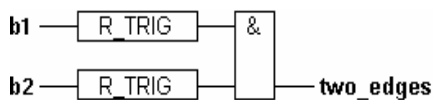
```

TRIG_B1 (b1);
edge_b1 := TRIG_B1.Q;    (* b1 變數邊緣偵測 *)

TRIG_B2 (b2);
edge_b2 := TRIG_B2.Q;    (* b2 變數邊緣偵測 *)

```

對一個程式而言，宣告的功能方塊樣例可以為**全域的**（案件中的每一個程式皆認識），或**區域的**（只用於單一程式）。用於 FBD 或 LD 語言的功能方塊無須宣告，因為 ISaGRAF FBD 編輯器會自動地宣告用過的方塊樣例。



(* 功能方塊永遠使用程式館中定義的方塊名稱。當每次圖形中插入一個新的方塊時，ISaGRAF FBD 及 Quick LD 編輯器會自動地宣告一個樣例 *)

在 FBD 及 Quick LD 編輯器中自動地宣告的功能方塊樣例永遠是“**區域性的**”。

➡ 網路位址

網路位址 (Network address) 是**選擇性的**。一個非零網路位址的變數可以在執行中被外部系統（例如監控系統）**偵測**到。更進一步的說，網路位址是提供給無法處理符號名稱的通信系統在每一次通信時的一個識別碼。當建立或修改一個變數時，可以將網路位址輸入給每一個變數使用。

A.10.1 主字典視窗

相同“**型態**”及“**範圍**”的變數會顯示於相同的字典編輯視窗中。變數的“**型態**”及“**範圍**”將會出現在視窗的標題中。

使用者指南

 編輯視窗內僅顯示出變數描述的主要欄位：名稱、屬性、網路位址以及文字註解，被選取的變數其完整描述永遠顯示在狀態列內。可利用下列工具列中的按鈕去選擇欲編輯的變數“範圍”：

-  **共用** 可用於任何案件的任何程式
-  **全域** 可用於目前案件的任何程式
-  **區域** 僅可用於一個程式中

可利用“Tab”鍵在標題列的按鈕上去選擇欲編輯變數的“型態”：

布林	整數/實數	計時器	訊息	功能方塊樣例	定義字
名稱	屬性		位址	註解	

 如欲搜尋變數名稱可以在工具列左方的文字輸入欄中輸入文字，在此情形下，依據目前的選擇，搜尋工作會從頭開始進行。**編輯 / 搜尋**命令亦提供變數名稱及註解中的字串搜尋功能，並能將游標移至此變數。**大小寫**不會影響搜尋的進行。

A.10.2 變數管理

在“**檔案**”功能表中所提供的命令適用於所有被選擇的變數族群、功能方塊樣例或定義字。使用“**其他**”命令可以選擇編輯不同的“型態”及“範圍”。

列印變數

使用“**檔案 / 列印**”命令可以將已編輯過的變數列或定義字列印到標準的Windows 印表設備。使用ISaGRAF 文件產生器來列印。列印輸出包括了每個變數或定義字的完整描述。

產生一個新的變數

“**編輯 / 新變數**”命令允許使用者於選擇的“型態”或“範圍”中創造新的變數、功能方塊樣例或定義字，新的變數會插入於目前選擇的變數之前。當此命令執行時，輸入欄框會被打開以便輸入變數描述。當描述完成後，可

按下“**儲存**”按鈕以便將變數放入表列中。新的輸入欄框會自動再度打開，所以使用者可以在相同的“**編輯**”命令中輸入其它變數。按下“**取消**”按鈕則會中斷變數產生程序。

▣ 修改存在的變數

“**編輯**”功能表中的“**編輯**”命令允許使用者去修改選擇列指到的變數的描述。當此命令執行時，輸入欄框會被打開以便修改變數的描述。當描述完成後，按下“**儲存**”按鈕來儲存所作的修改。使用者亦可按“**下一個**”及“**前一個**”按鈕修改相鄰的變數。按下“**取消**”按鈕會將對話框關閉，且不會儲存任何做過的修改。



剪下及貼上

ISaGRAF 的字典編輯工具可做**多行式選取**，許多提供的命令可用於目前所編輯的變數集。下列為“**編輯**”功能表中提供的命令：

複製 命令可將選出的一群變數拷貝到字典剪貼簿上。

剪下 命令可將選出的一群變數拷貝並自原列表中移除。

清除 命令可將原列表中選出的一群變數清除

貼上 命令可在選擇變數前插入字典剪貼簿的內容。

複製/剪下/貼上功能可以使用於一個變數集至另一個變數集上，但是它不能用在不同物件屬性上。

▣ 變數排序

“**工具 / 排序**”命令將現在的變數或定義字加以排序，分類順序是以變數屬性來排列；

先由內部變數排起

再來是輸入變數

最後處理輸出變數

具相同屬性的變數皆以字母順序排列，定義字也是以字母順序排列。

▣ 設定網路位址

使用者指南

網路位址是**選擇性的**。在執行狀態下一個非零值的網路位址可以被一個外部的系統（例如程序顯示系統）**連結到**。當變數被建立或修改時，網路位址可以輸入給每一個變數。“**編輯 / 重新編號**”命令可讓使用者將全部的變數群做網路位址設定，此命令一旦執行，它會對選取的變數集產生作用。當輸入**16 進位的起始位址**後，變數集的網路位址將被設定為**連續的位址**。若輸入一個空白的位址值，將使所選取的變數網路位址全部重置成**0**。

布林“真假”字串表示法

當編輯定義字時，“**工具 / 輸入真假定義**”命令允許使用者自定一個字串來取代真 (TRUE) 及假 (FALSE) 的狀態。這個取代的字串於除錯時使用。如果它被用在程式中，則它必須被指定為定義字。

A.10.3 物件的描述

一個完整的描述必須輸入給每一個變數、功能方塊樣例或定義字。每一種物件型態的描述欄位皆不相同。下列欄位是任何型態的變數皆共同具備的：

名稱 表示變數名稱：第一個字元必須是字母，其次的字元必須是字母，數字或‘_’。

網路位址 表示**16 進位數值**的網路位址（可選擇的）。當此欄位的值不是零時，於執行時變數可為外界之系統所連結到。

註解 變數描述。

可回復 這是一個選項，表示變數必須被存在不揮發的記憶體中。

 這些是布林變數的其它欄位：

屬性 可指定為內部、常數、輸入或輸出變數。

“False”字串 於除錯時表示假值的字串。

“True”字串 於除錯時表示真值的字串。

開始時為真 若此選項被打鉤，則初始值會設為真 (TRUE)，否則設為假 (FALSE)。



這些是**整數或實數**類比變數的其它欄位：

屬性 可指定內部、常數、輸入或輸出變數。

格式 可指定為整數或實數變數。於除錯時使用的顯示格式可以選擇。

單位字串 除錯時表示物裡單位所使用的字串。

轉換 變數關連的轉換表或轉換函數名稱（僅適用於輸出入變數）。

初始值 變數的初始值（必須與變數的格式相同）。若沒有指定，則初始值為“0”。



這些是**計時器**變數的其它欄位：

屬性 可指定為內部或常數變數。

初始值 變數的初始值（時間值）。若未特別指定時，初始值為time#0s。



這些是**訊息**變數的其它欄位：

屬性 可指定為內部、常數、輸入或輸出變數。

最大長度 指定訊息變數的最大字元個數。

初始值 變數的初始值（長度不能超過訊息變數的容量）。若未特別指定時，初始值將為空字串。



這些是**定義字**的欄位：

名稱 用於 ST 程式的名稱，第一個字元必須是字母，接著的字元必須是字母，數字或‘_’符號。

定義 於編譯時將取代定義字，並依據 ST 語法的字串。例如：Name = PI – Equivalence = 3.14159。

註解 定義字的註解。



這些是**功能方塊樣例**的欄位：

名稱 用於 ST 程式的樣例名稱，第一個字元必須是字母，接著的字元是字母，數字或‘_’符號。

使用者指南

型態 於程式館中相關的功能方塊名稱。

註解 功能方塊樣例註解。

A.10.4 快速宣告

“工具 / 快速宣告” 命令允許一次宣告多個變數。快速宣告所產生的變數以號碼規則來命名，因此你必須定義：

第一個及最後一個變數的索引號碼

變數符號中的號碼前後的文字

變數符號中的號碼位數

另外：你也能指定所產生的變數的基本屬性（內部、輸入或輸出...）以及不同變數型態專屬的性質（“可回復” 屬性、整數或實數格式、訊息字串最大長度）。

因為變數名稱不能以數字開頭，所以你必須於變數號碼之前插入開頭的文字。當“號碼位數”設定成“自動”時，ISaGRAF 將設定號碼位數成最小需求位數，當你指定號碼位數，ISaGRAF 會將所有號碼設定成指定的位數，不足之位數則於號碼之前以‘0’來補足。指定號碼位數將可避免變數排序時產生錯誤的次序。下面為一些範例。

範例： 快速宣告設定：

編號:

從: 9 到: 11

位數: auto

符號:

名稱: Var ##xx

產生三個變數，如下：

Var9xx Var10xx Var11xx

範例： 快速宣告設定：

編號:	
從:	1 到: 100
位數:	3
符號:	
名稱:	MyVar ##

產生名稱爲 MyVar001 到 MyVar100 的 100 個變數

A.10.5 Modbus SCADA 位址對應

ISaGRAF “網路位址” 常用於建立 ISaGRAF 系統與 SCADA 之間 Modbus 通訊連結，此時 SCADA 爲 Modbus master (主)，而 ISaGRAF 目標硬體爲 Modbus slave (從)。網路位址用於產生所有必須被 SCADA 控制的變數虛擬 Modbus 對應。“**工具 / Modbus SCADA 位址對應**” 可用來快速產生應用程式變數的 Modbus 虛擬對應。

對應工具顯示兩個表列；上面的表列爲 Modbus 對應的區段 (4096 個)，顯示對應的變數 (具有網路位址的變數)，下面的表列顯示未對應的變數 (不具有網路位址的變數)。“0” 位址爲保留值，不能用於變數位址。

使用“**編輯**”功能表的“**對應**”及“**移除**”命令搬移變數到另一個表列上，如此便可建立對應的關係。於變數符號上雙按滑鼠左鍵，也可以達到相同的效果。在任何時刻你都可以使用“**區段**”來觀看另一區段的對應。

“**選項**”功能表的命令能隨時觀看 10 進位或 16 進位的位址。

“**編輯 / 尋找**”命令可用來搜尋宣告的變數 (不管對應與否)。

A.10.6 與其它應用程式交換資訊

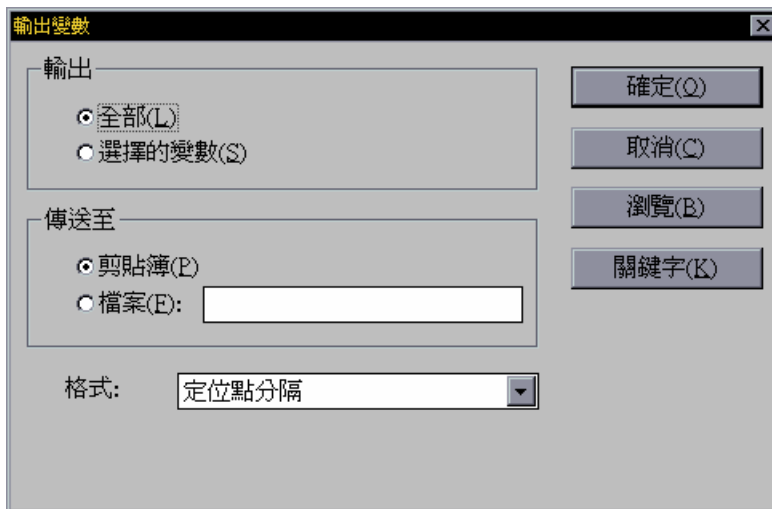
ISaGRAF 字典編輯工具提供了與其它應用程式交換資訊的輸入/輸出功能，例如文字處理器、試算表、資料庫管理等。這些命令在“**工具**”功能表中分爲幾群。“**輸出文字**”命令建立純粹的 ASCII 文字描述欄位去描述一組欲編輯的物件，然後將這些文字儲存至視窗剪貼板或檔案中，這些資訊通常可用於其他應用程式。至於“**輸入文字**”命令則從儲存於視窗剪貼板或檔案輸入

使用者指南

變數宣告描述欄位 (純 ASCII 文字格式)，並且以輸入的欄位更新目前的編輯列，這些文字格式通常由其它的應用程式產生。

輸出資料

下列的對話框於“**輸出文字**”命令執行時顯示。它允許使用者去控制輸出的機制 (mechanism)。



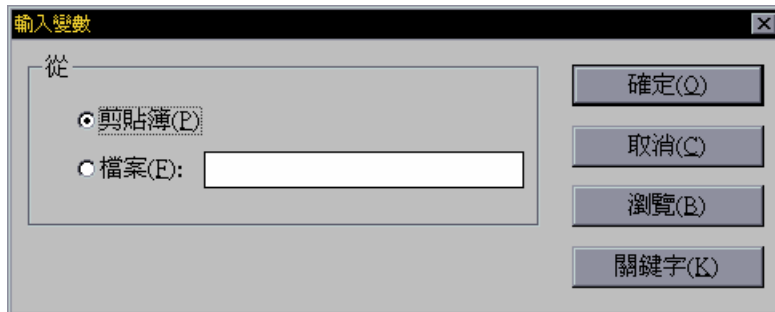
若“**全部**”選項被選取，則會輸出全部的編輯列。在此情形下，目前的選擇區域會被忽略。若“**選取的變數**”選項被選取，則僅有反白的變數會被輸出。如果“**剪貼簿**”選項被選取，則輸出資訊會以 ASCII 文字格式形式儲存於視窗剪貼板中。該文字在其它應用中可用“貼上”命令來利用。如果“**檔案**”選項被選取，則輸出文字會被儲存於 ASCII 檔案中，該檔案的完整路徑名稱必須被鍵入。“**瀏覽**”命令可以用來尋找存在的路徑名稱。

然後選擇使用者輸出文字的格式。所能提供的格式種類在別的章節中有更進一步的說明。按下“**確定**”按鈕便可執行輸出功能，按下“**取消**”按鈕可關閉對話框且跳離輸出命令。

所有被選取的物件欄位皆被儲存在輸出文字內，且以標準宣告次序排列。輸出文字的第一行包含了欄位名稱。每一個物件皆以一行文字描述。“列結束”分格符號是標準的 MS-DOS ASCII 碼“0d-0a”。在第一行中用以識別欄位的名稱是可以改變的，經由按下“**關鍵字**”按鈕的操作即可。這個命令在其它章節中有更進一步的描述。

□ 輸入資料

下列的對話框會在“輸入文字”命令被執行時顯示出來。它可以允許使用者去控制輸入的方法。



如果“剪貼簿”選項被選取，則輸入資料會以 ASCII 文字的格式從 Windows 剪貼簿輸入。如果“檔案”選項被選取，則輸入資料會從 ASCII 檔案讀入。該檔案的完整路徑名稱必須被鍵入方可，可用“瀏覽”命令可用以尋找目前存在的路徑名稱。

輸入文字所用的格式（分隔符號）於輸入時會自動辨識。可提供的格式在其它章節中會有詳細說明。按下“確定”按鈕便可執行輸入功能，按下“取消”按鈕可關閉對話框，並跳離輸入命令。按下“關鍵字”按鈕可改變輸入行中第一行的欄位辨識名稱，這個命令在其他章節有詳細的描述。

文字的第一行必須包含欄位名稱，這是依據隨後行列的排列次序而定的。每一個物件皆以一行文字來描述。“列結束”分隔符號是標準的 MS-DOS “0d-0a”。欄位能以任何次序顯示。如果少了某些欄位，它會自動地以內定值填入輸入物件描述之中。如果在編輯列中已有相同物件存在，則使用者必須確認是否要覆蓋。物件於是會被輸入欄位所更新。如果少了某些欄位，它們便不會在物件描述中被更新。

□ 可提供的文字格式

下面是輸出命令可提供的格式，輸入命令會自動地辨識這些格式。

定位分隔 (tab separators)

解說： 欄位以跳欄字元分離

使用者指南

例如：

Name	Attribute	Comment
level	internal	internal calculated water level
alm1	output	main alarm output

逗號分隔 (comma separators)

解說：

Name,Attribute,Comment
level,internal,internal calculated water level
alm1,output,main alarm output

例如：

Name,Attribute,Comment
level,internal,internal calculated water level
alm1,output,main alarm output

分號分隔 (semicolon separators)

解說：

Name;Attribute;Comment
level;internal;internal calculated water level
alm1;output;main alarm output

例如：

Name;Attribute;Comment
level;internal;internal calculated water level
alm1;output;main alarm output

逗號及引號 (commas and quotes)

解說：

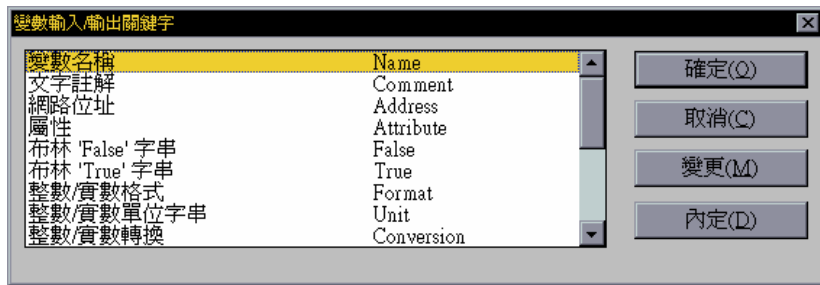
"Name","Attribute","Comment"
"level","internal","internal calculated water level"
"alm1","output","main alarm output"

例如：

"Name","Attribute","Comment"
"level","internal","internal calculated water level"
"alm1","output","main alarm output"

＝ 關鍵字 (Keywords)

按下“**關鍵字**”按鈕可以改變位於輸入/輸出行第一行用於識別欄位的名稱。這個命令打開下列的對話框：



以上的視窗所顯示出物件欄位以及相對應的關鍵字。若要修改關鍵字，使用者必須先選好列中的欄位，然後按“變更”按鈕。若按“內定”按鈕則會儲存內定的關鍵字集。關鍵字的命名必須符合下列規則：

名稱不可超過 **16** 個字元

第一個字元必須是**字母**

隨後的字元可以是**字母**，**數字**或‘**_**’符號

相同的名稱不可用於不同的關鍵字

下列是 ISaGRAF 提供的標準關鍵字：

物件名稱	Name
文字註解	Comment
網路位址	Address
屬性 (內部，輸入，輸出)	Attribute
布林‘假’字串	False
布林‘真’字串	True
類比格式 (實數或整數)	Format
類比單位字串	Unit
類比轉換名稱	Conversion
訊息最大長度	MaxLength
功能方塊程式庫型態	Library
定義字相等詞	Equivalence
內部屬性	Internal
輸入屬性	Input
輸出屬性	Output

使用者指南

常數屬性 **Constant**

實數類比格式 **Real**

整數類比格式 **Integer**

A.11 使用 I/O 連結編輯器

I/O 連結的操作目的是建立應用程式中的 I/O 變數與實際硬體板上的 I/O 點建立起邏輯的連接。使用者必須分辨及安裝目標硬體機器上所欲連結的 I/O 板，然後將相對應的 I/O 變數放在正確的 I/O 接點上。

視窗的左方表示實際硬體的 **I/O 板**，並且以**槽位**表示。槽位有可能是空槽，亦有可能被某一 I/O 板或複合板所使用。每一個槽位都有一個**順序識別號碼**。槽位可以容納 **255** 塊板子。視窗右方顯示出將被連接至 I/O 板的參數和變數。一塊 I/O 板最多可容納 **128** 個 I/O 點。所有的 I/O 板加起來（包括單一設備及複合設備的板子）不能超過 **255** 個 I/O 點。

圖示

正面的圖示表示連接到板子上的變數型態及屬性。ISaGRAF 不允許不同型態的變數連接到相同的板子上。下列是使用到的圖示的意義：

-  布林型態
-  整數/實數型態（二種型態的變數皆可連接）
-  字串型態
-  輸入型態（空心者表示未接變數）
-  輸出型態（空心者表示未接變數）
-  表示至少有一個輸入變數被連接
-  表示至少有一個輸出變數被連接

下面的圖示表示在槽位上的 I/O 型態：

-  複合 I/O 設備
-  真實的 I/O 板
-  虛擬的 I/O 板

下面的圖示用於參數或接點：

-  I/O 板參數
-  空接點
-  已連接接點



移動板子

可運用工具列上的這些按鈕或“**編輯 / 板子上移/下移**”功能表命令在 I/O 板主功能表中上下移動所選取的板子。“**編輯 / 插入空白插槽**”命令可在目前的位置上插入空白槽位。

A.11.1 I/O 板定義

“**編輯**”功能表中包含了可定義板子的命令（設定它的參數），以及連接 I/O 變數至板子的接點上。



選擇 I/O 板型態

在連接 I/O 變數前，I/O 板必須先輸入。程式庫或預先定義的板子在 ISaGRAF 工作平台中皆可提供，這個程式庫可能早已被一至多個 I/O 設備供應商編譯過。“**編輯 / 設定板子 / 設備**”命令是用來選取板子，這個命令可以從 ISaGRAF 的程式庫中被用來選擇單一的板子或複合的 I/O 設備，它亦可能在空槽上點兩下去設定相關的板子及設備。

所有板子上接點的型態及方向（輸入/輸出）皆相同（布林，整數/實數或字串）。整數或實數變數在 I/O 連接的階段並沒有任何區分。複合的 I/O 設備代表具有不同型態或方向的 I/O 裝置，複合的 I/O 設備是以一些 I/O 單板構成，但僅在機架上佔一空槽。



板子的移除

“**編輯 / 清除插槽**”命令可用以移除現有的板子及 I/O 設備。若板子上有變數連接，則在清除空槽時，這些變數會自動清除。



真實與虛擬板

“**編輯 / 真實板/虛擬板**”命令可設定適當的板子或複合的 I/O 設備。下列在槽位中的圖示顯示出適當的板子：



真實 I/O 板



虛擬 I/O 板

在**真實模式**下，I/O 變數與相關的 I/O 裝置直接連接在一起，在應用程式中的輸出入操作與實際的 I/O 裝置的輸出入狀況會緊密的配合。在**虛擬模式**下，I/O 變數被當成內部變數來處理，它們可以被除錯器讀取或更新，所以使用者可模擬整個 I/O 的程序，但是它與真實的環境並未實際連接。



技術註記

“**工具 / 技術註記**”命令線上顯示出選取的板子及複合設備的使用者註記，這些技術註記乃是由硬體供應商所撰寫，它包含了全部 I/O 板管理所需的資訊，亦解釋它的參數意義。



移除被連結的變數

“**工具 / 清除接點**”命令將所有連接在板子上的 I/O 移開。

⇒ 空接點的註解定義

“**工具 / 設定接點註解**”命令可加上未連結接點的註解。

A.11.2 板子參數設定

欲設定板子的參數數值，使用者必須在參數名單右方該名稱上雙按滑鼠，亦可以點選反白區中，並且在“**編輯**”選單中選取“**設定接點參數**”命令。參數被列在接點的上方。下列的圖示被用來表示參數：



板子參數

參數的意義和輸入格式由 I/O 板供應商所設計。使用“**工具 / 技術註記**”命令或參考硬體手冊可以得到更多有關 I/O 板參數的資料。

A.11.3 連結 I/O 接點

欲設定連結的接點，使用者必須在視窗的右邊位置上雙按滑鼠左鍵。亦可以點選它，並且執行“**編輯 / 設定接點參數**”命令。下列圖示用來表示接點：

-  表示仍然空置之接點
-  表示已被連結之接點

名單中包含了所有與 I/O 板型態及輸出入方向吻合的變數，僅有未被連結的變數才會顯示於此。“**連結**”按鈕將選取的變數連結至選取的接點上。“**解除**”按鈕將連接在接點上的變數移除。“**下一個**”以及“**上一個**”按鈕用來選擇版子上其它的接點。被選取的接點位置將顯示在對話視窗的標題中。

A.11.4 直接表示變數

空置之接點表示沒有連結已宣告過的 I/O 變數，ISaGRAF 可以在原始程式中用**直接表示之變數**代表空置接點，直接表示之變數以“%”字元放於前方做為識別。

下列為一些板子接點上可用之直接表示變數。“s”表示 I/O 板之插槽編號，“c”表示接點編號。

- %IXs.c** 表示布林輸入 I/O 板的空置接點
- %IDs.c** 表示整數輸入 I/O 板的空置接點
- %ISs.c** 表示字串輸入 I/O 板的空置接點
- %QXs.c** 表示布林輸出 I/O 板的空置接點
- %QDs.c** 表示整數輸出 I/O 板的空置接點
- %QSS.c** 表示字串輸出 I/O 板的空置接點

下列為一些複合裝置上可用之直接表示變數。“s”表示裝置之槽位編號，“b”表示在該裝置中 I/O 板的位置，“c”表示接點編號。

- %IXs.b.c** 表示布林輸入 I/O 板的空置接點
- %IDs.b.c** 表示整數輸入 I/O 板的空置接點
- %ISs.b.c** 表示字串輸入 I/O 板的空置接點
- %QXs.b.c** 表示布林輸出 I/O 板的空置接點

`%QDs.b.c` 表示整數輸出 I/O 板的空置接點

`%QSS.b.c` 表示字串輸出 I/O 板的空置接點

譬如：

`%QX1.6` 表示板上#1 上的第 6 點接點 (布林輸出型態)

`%ID2.1.7` 表示設備#2 上的板子#1 上的第 7 點接點 (整數輸入型態)

直接表示變數不能表示“實數”資料型態。

A.11.5 編號

使用“**選項 / 編號**”命令用於設定編號,你能指定第一槽及每槽第一個連結點的號碼,如下對話框所示：

編號對話框的截圖顯示如下：

- 對話框標題為“編號”。
- 左側區域標題為“第一個編號給”。
- 該區域內有兩個輸入框：
 - “槽位:” (Slot) 輸入框，當前顯示為“0”。
 - “接點:” (Point) 輸入框，當前顯示為“1”。
- 右側區域包含兩個按鈕：
 - “確定(Q)” (OK) 按鈕。
 - “取消(Q)” (Cancel) 按鈕。

槽位編號內定從“0”開始，連結點編號內定從“1”開始。

警告： 若於程式中使用變數直接表示法時必須小心，因為改變編號將可能導致編譯時產生錯誤的結果。

A.11.6 設定個別保護

ISaGRAF 工作平台提供完整的階層密碼資料保護系統。I/O 連結可以以密碼做全部保護，另外 ISaGRAF 也讓你做個別 I/O 接點的保護。但必須有下列的先決條件才可：

使用者指南

密碼早已定義於密碼定義系統中（使用案件管理視窗下的“**案件 / 設定密碼**”命令），如此便可將保護密碼應用於個別的保護上。

使用較高的優先保護層級（相對於全部 I/O 保護）作個別的保護。

當 I/O 連結點具有個別保護，於 I/O 連結視窗中的連結變數名稱旁將出現一個小的圖示。



使用“**編輯**”功能表下的“**設定接點保護**”及“**移除接點保護**”命令設定或移除所選連結點的個別保護，這兩個命令皆會要求你輸入有效的密碼，如此便可將保護層級的密碼依附到此連結點，然後每次你要改變此種連結點時，你必須輸入有效的密碼才行。

警告： 假如連結點相對應的密碼從保護系統中被移除，且沒有更高層級密碼被定義的話，此連結點將不可再加以改變，除非有效層級密碼再被定義好以後才行。

A.12 創造轉換表

ISaGRAF 工作平台允許使用者創造轉換表。轉換表就是一些點的集合，用來定義類比轉換。轉換表能附加到類比輸入或輸出變數上，它創造了電位值（從輸入感測器 (sensor) 讀取或送至輸出設備）與物理值（用於應用程式中）之間的比例關係。

藉由執行ISaGRAF 字典視窗中的“**工具 / 轉換表**”命令會出現一個對話盒來編輯轉換表。

已定義的轉換表可以用來過濾這個案件中的任何輸入或輸出類比變數的值。欲附加轉換表到變數上，必須打開變數宣告編輯器，然後選擇輸入或輸出類比變數，且編輯它的參數。變數不能附加未定義的轉換表。

A.12.1 主要的命令

“**轉換表**”對話盒會列出已定義過的轉換表及包含主要命令的操作按鈕，使用者可以編輯已存在的轉換表（定義轉換表的點），建立新的轉換表，重新命名轉換表或刪除。按下確定來離開“**轉換表**”對話盒及將它們儲存入硬碟中。



開新表格

“**開新表格**”命令允許使用者創造新的轉換表格，每個案件可以創造多達**127**個轉換表，但僅有用到的表格（有附加到類比變數）才會加入到應用執行碼中。

表格命令必須遵循下述的規則：

名稱不能超過**16**個字元

第一個字元必須是**英文字母**

其餘字元可以是英文字母、阿拉伯數字或底線符號等字元

表格名稱是不分大小寫的



改變表格內容

“**編輯**”命令用來輸入表列中所選取的表格的點，你也可以於表格名稱上雙按滑鼠左鍵來編輯它。當開新表格時，會自動執行“**編輯**”命令。每一個表格至少必須輸入兩個點。

A.12.2 輸入表格的點

“**編輯**”對話框允許使用者定義轉換表的點，它會將已定義的點集列於左邊，右下方則顯示定義表格的圖形曲線。使用對話盒命令來輸入點。使用者必須遵從定義點的數值規則（說明於這個章節的最後面）。對話盒左方總是包含目前編輯表格已存在的點集合，左邊的欄位顯示點的電位（外部）值，右邊的欄位顯示物理（內部）值。使用者必須選擇表列中的一個點，才能變更它的值或清除（移除）它。表列中最後一列（“... ..”）用來定義新的點。對話盒右下方顯示目前編輯表格的圖形曲線，它不會顯示軸及座標，而是以比例的關係來表示曲線，這種表示法用來快速檢查曲線正確與否是相當有用的。

□ 定義新的點

當要定義新點時，選擇列表中最後一列的（“... ..”），於開始定義新轉換表時，這一系列亦是內定的模式。使用者必須輸入每一點的電位（外部）與物理（內部）值，這些值以單精度浮點數來儲存。注意，必須輸入至少**兩個點**才能定義出一個曲線。當電位及物理值都輸入後，按“**儲存**”按鈕將點加入表列中。每一個轉換表最大定義的點個數為**32**點。

□ 更改點

欲變更已存在點的值，首先需從列表中選擇它，然後重新輸入這個點的新電位（外部）及物理（內部）值。這些值以簡單精度浮點數來儲存。當電位及物理值都輸入後，按“**儲存**”按鈕將點加入表列中。

□ 清除點

欲清除已存在的點，可以從表列中選取它，然後按“**清除**”按鈕。記住，必須輸入至少**兩個點**才能定義出一個表格。

A.12.3 規則及限制

當定義轉換表時，必須遵從下述的規則（轉換表可以用來轉換輸入及輸出的類比變數）：

兩點不可以定義相同的電位值

曲線必須是連續的漸增或漸減

兩點不可以定義相同的物理值

當定義一個案件的轉換表時需遵循下列的限制：

同一個案件中不能定義超過 **127** 個轉換表

同一個轉換表中不能定義超過 **32** 個點

A.13 使用碼產生器

藉由 ISaGRAF 工作平台視窗的“驗證”與“製作”命令，將自動開啓碼產生器視窗，而應用碼產生運作結束時，碼產生器視窗並不會自動關閉，所以使用者依然可以從視窗功能表中進入所有的碼產生用到的命令及選項。

A.13.1 主要的命令

“檔案”功能表包含程式語法檢查及碼產生等命令。

▣ 製作應用碼

“製作”命令產生案件的整個應用碼。在產生任何碼之前，這個命令會檢查宣告及程式的語法，單一程式單獨編譯時無法偵測到的任何錯誤，於碼產生時都可以偵測到。這樣的動作亦可應用到轉換表、I/O 變數連結及程式庫連結。當偵測到錯誤時，碼產生器會停止程式編譯，在繼續碼產生器之前必須修正程式錯誤。程式若早已被檢查過（沒有錯誤產生），且於上次“驗證”過後未再更改過，將不會重新編譯一次。而變數宣告及應用程式式一致的檢查，每次都會執行。於程式檢查期間，“製作”操作能使用 **ESCAPE** 鍵來中斷。

注意： 假如程式的區域變數宣告被變更過，程式會被重新驗證，假如全域變數被變更過，所有的程式都會被重新驗證。

▣ 程式語法檢查

“驗證程式”命令允許使用者驗證一個程式，即使所選的程式從上次驗證過後未被變更，依然會加以編譯。“驗證字典”命令允許使用者驗證案件中所所有用到的變數的宣告。

“驗證所有程式”檢查案件所有程式的語法，即使其中一些程式未被變更過，當偵測到程式的錯誤時，這個命令依然不會停止，這樣可以用來產生案件程式的所有錯誤的列表，這個命令可以按 **ESCAPE** 鍵來終止它。

二 模擬變更

“**更新**”命令將模擬所有案件程式已被變更過，所以於下次“**製作**”執行時，所有程式皆會被驗證。“**開啓**”命令用來開啓最後驗證的程式，對於直接進入偵測到語法錯誤的程式來說，這個命令是非常有用的。

A.13.2 編譯選項

“**編譯選項**”命令用來設定 ISaGRAF 碼產生器的主要參數，以建立及最佳化目標硬體碼。這個命令的目標是根據 ISaGRAF 目標硬體來選擇產生器的型態，以及根據所期望的編譯時間與應用執行期的需求去設定最佳化參數。

“**上載**”按鈕開啓另一個對話框，讓你可以選取壓縮原始檔的功能，以便與應用碼一起下載至目標硬體內。想要使用“上載”的功能，請參考“上載”章節以得到進一步的資訊。

選擇目標硬體

上半部列出可用目標硬體碼的列表，“>>”符號指出選擇的目標硬體。ISaGRAF 碼產生器在一次編譯動作中可以產生三種不同的碼。使用“**選擇**”與“**反向選擇**”按鈕來設定所需的目標硬體碼集（根據你的目標硬體）。下面是標準的 ISaGRAF 目標硬體：

SIMULATE： 這個碼用於工作平台上的 ISaGRAF 模擬器。假如這個目標硬體沒有被選擇，應用碼產生後，模擬器將無法執行。

ISA86M： 這是一個 TIC 碼 (Target Independent Code，與目標硬體無關的碼)，用於 Intel 處理器為基礎的 ISaGRAF 核心程式 (kernel)，這個處理器型態僅是於產生的碼中，位元組 (byte) 置放順序不同而已。

ISA68M： 這是一個 TIC 碼，用於 Motorola 處理器為基礎的 ISaGRAF 核心程式，這個處理器型態僅是於產生的碼中，位元組 (byte) 置放順序不同而已。

SCC： 選擇這個目標硬體將使 ISaGRAF 編譯器產生結構化的“C”語言原始碼，可被再編譯及與 ISaGRAF 目標硬體核心程式庫連結，產生一個可嵌入的可執行碼。

使用者指南

CC86M： 選擇這個目標硬體將使 ISaGRAF 編譯器產生無結構的“C”語言原始碼，可被再編譯及與 ISaGRAF 目標硬體核心程式庫連結，產生一個可嵌入的可執行碼。當 ISaGRAF 3.23 之前的版本不支援結構化的“C”原始碼，使用此選項可以與先前的版本相容。

根據你的硬體手冊去得知安裝於你的 PLC 上的 ISaGRAF 目標硬體核心程式的型態，其它的目標硬體型態（機器碼，C 原始碼..）可以被支援於未來的 ISaGRAF 工作平台版本。

□ SFC 處理

選取“使用嵌入的 SFC 引擎”對話框，可用來開啓 ISaGRAF SFC 引擎的使用。這種模式將是較好的方式，它會導致較高的執行期效能。然而在 ISaGRAF 目標硬體的某些特別的做法時，目標硬體引擎可能會遺失掉。若顧客的目標硬體有用到 ISaGRAF 碼前置處理器，你必須移除這個選項，讓 ISaGRAF 編譯器將 SFC 圖編譯成低階指令。參考你的硬體手冊，以得到使用這個選項的更多資訊。

□ 最佳化選項

下面是用於 ISaGRAF 碼產生器最佳化目標硬體碼的參數，你能夠從“編譯選項”對話盒來設定它。“內定”按鈕用於移除所有的最佳化選項，以便於減少編譯時間。

- ◆ 當“執行兩次最佳化”選項被設定，ISaGRAF 碼最佳化會執行兩次，第二次最佳化過程一般會比第一次過程較不明顯。
- ◆ 當“評估常數表示式”選項被設定，編譯器會求得常數表示式的值。譬如，數值表示式“2 + 3”於目標硬體碼中會以“5”來取代。若這個選項未設定，常數表示式會於執行期間才加以計算。
- ◆ 當“懸置未用的標籤”選項被設定，最佳化過程會簡化掉跳躍及程式標籤，以便懸置未用的標籤或空的跳躍。

◆ 當“**變數複製最佳化**”選項被設定，暫時變數（用於儲存暫時的結果）的使用會做最佳化，這個選項一般與“**最佳化表示式**”選項一起使用。當這個選項被設定，最佳化過程會重複使用表示式及副表示式（在程式中使用超過兩次）的結果。

◆ 當“**懸置未用碼**”選項被設定，最佳化過程會懸置沒意義的碼。譬如，若程式為如下的敘述：“**var := 1; var := X;**”，產生相對的碼僅為：“**var := X;**”。

◆ 當“**最佳化算數運算**”選項被設定，最佳化過程會根據特別的運算來簡化算數運算。譬如，表示式“**A + 0**”將被取代為“**A**”。當“**最佳化布林運算**”選項被設定，最佳化過程會根據特別的運算來簡化布林運算。譬如，布林表示式“**A & A**”將被取代為“**A**”。

◆ 當“**建立二元決定圖**”選項被設定，最佳化過程會以縮減的條件跳躍運算來取代布林方程式（混和**AND**，**OR**，**XOR**與**NOT**的運算元），轉換僅有當期待的跳躍順序執行時間少於原表示式的時間才會加以處理。

下述的表格集合每個參數期待的最佳化及需要的編譯時間：

效能提升 編譯時間

執行兩次最佳化 **XXXX (*)**

最佳化常數表示式 **XXXXXXXX XXXX**

懸置未用的標籤 **XXXX XXXXXXXX**

最佳化變數複製 **XXXX XXXXXXXX**

最佳化表示式 **XXXX XXXXXXXX**

懸置未用的碼 **XXXX XXXXXXXX**

最佳化算數運算 **XXXXXXXX XXXX**

最佳化布林運算 **XXXXXXXX XXXX**

建立二元決定圖 **XXXXXXXXXXXX XXXXXXXXXXXX**

(*) 編譯時間乘以兩倍

A.13.3 產生 C 原始碼

ISaGRAF 工作平台可以產生“C”語言的原始碼，用這樣的方法，所有的應用程式內容（包含 SFC 圖形描述）、資料庫定義及碼的順序都以“C”原始碼格式產生。有兩種樣式的“C”原始碼可以產生：

CC86M (C 原始碼 - 3.04 版) 產生非結構化的“C”原始碼，假如你的目標軟體為 3.23 之前的版本，請選擇此種型態。

SCC (結構化 C 原始碼) 產生非結構化的“C”原始碼，假如你的目標軟體為 3.23 或之後的版本，最好選擇此種型態。

下述的兩個檔案將被產生於案件目錄下：

APPLI.C 應用程式原始碼

APPLI.H “C”語言定義

若產生的是結構化“C”原始碼，應用程式的每個程式將產生一個“.C”原始碼及一個“.H”定義檔，另外再加上“**APPLI.C**”及“**APPLI.H**”兩個共用檔。這些檔案必須被編譯且連結至 ISaGRAF 目標硬體程式庫，以產生最後的可執行碼。參考“ISaGRAF I/O 發展工具使用者指南”，以得到關於建議的實行技術方面的更進一步資訊。

注意： 當 ISaGRAF 應用程式是用“C”來編譯，一些如應用程式下載、線上變更與中斷點等除錯功能將不再可用。

A.13.4 觀看訊息

“**編輯**”功能表包含觀看碼產生期間所建立的各種文字檔與於碼產生視窗上的語法檢查操作等命令。碼產生視窗是一個包含碼產生或語法檢查操作期間的文字區，所有的訊息皆儲存於磁碟上，所以可以使用“**編輯**”功能表命令來測試。

⇒ 編輯命令

“**清除螢幕**”命令用來清除視窗內的文字區。於每次碼產生或語法檢查以前，這個視窗也會自動清除。“**複製**”命令用於複製顯示的文字到視窗剪貼簿中，所以可以給諸如 ISaGRAF 文字編輯器等其它的應用程式所使用。

▣ 觀看編輯器輸出訊息

“**執行訊息**”命令顯示最後一次“**製作**”或“**驗證**”動作所顯示於視窗文字區的所有訊息，這個命令可以觀看所有的錯誤訊息。

“**編輯**”功能表的其它命令允許使用者監看創造於語法驗證及碼產生期間的輔助文字檔，這些檔案通常不用於 ISaGRAF 案件中。

A.13.5 定義資源

“**選項**”功能表的“**資源**”命令允許使用者去定義資源。資源是使用者定義的任何格式的資料（網路組態，硬體設定 ...），這些資料將會合併到產生的碼中，以便下載到目標硬體 PLC 中。ISaGRAF 硬體核心不會直接拿這樣的資料作運算，而是用於目標硬體 PLC 上其它的軟體。參考你的硬體手冊以得到關於可利用的資源的更進一步說明。

▣ 資源定義檔

資源被定義於 ISaGRAF 案件的“**資源定義檔**”中，它是一個純文字檔，由 ISaGRAF 資源編輯器所處理。當應用程式碼建立時，這個編譯器會自動執行。這一節解釋這個檔案的語法，資源定義檔使用 ST 語言的語法規則。註解（以“**(*)**”開始，“***)**”結束）能被插入於文字中的任何一處，字串以單引號為界線。參考本手冊的第二部分，可得到關於輸入數值的語法格式的更進一步解釋。

▣ 語言參考

下面是用於資源定義檔中的關鍵字與敘述表。

ULONGDATA

使用者指南

意義： 指示為整數值集合的資源，以不帶符號的 32 位元整數存於目標硬體碼中，且以資源定義檔指定的順序儲存。各種數值之間以逗點來區隔，資源名稱不能超過 15 個字元。

語法： **ULONGDATA** '<資源名稱>'

BEGIN

...選擇目標硬體...

...數值集合...

END

例子： **ULongData** 'MYDATA'

Begin

...

0, -1, 100_000, (*十進位*)

16#A0B1, 2#1011_0101(*16 進位,二進位*)

End

VARLIST

意義： 指定變數位址集合的資源。在原始定義檔中，變數以名稱來識別，而在目標硬體碼中，變數位址以不帶符號的 16 位元整數儲存，位址以資源定義檔中指定的順序儲存。各變數間必須以逗點分隔。資源名稱不能超過 15 個字元。

語法： **VARLIST** '<資源名稱>'

BEGIN

... 選擇目標硬體...

...變數名稱集...

END

例子： **VarList** 'LIST'

Begin

```
...
Var100, MyParameter, Command, Alarm
End
```

BINARYFILE

意義： 指定二元檔資源。資源中的資料儲存於一個 MS-DOS 檔中，目標硬體資源定義加上路徑檔名來完成。ISaGRAF 資源編譯器不會轉換換列字元。資源名稱不能超過 **15** 個字元。

語法： **BINARYFILE** '<資源名稱>'
BEGIN
 ... 選擇目標硬體...
FROM '<來源路徑>'
TO '<目的路徑>'
END

例子： **BinaryFile** 'MYFILE'
Begin
 ...
From 'c:\user\config.bin'
To '/dd/user/appl/config.dat'
End

TEXTFILE

意義： 指定文字檔資源。資源資料儲存於 ASCII 檔案中，目標硬體資源定義以路徑檔名加以完成。ISaGRAF 資源定義檔會根據目標硬體主系統轉換法轉換換列字元。資源名稱不能超過 **15** 個字元。

語法： **TEXTFILE** '<資源名稱>'
BEGIN
 ... 選擇目標硬體...

使用者指南

```
FROM '<來源路徑>'  
TO '<目的路徑>'  
END
```

例子： `TextFile 'MYFILE'`
`Begin`
 ...
 `From 'c:\user\config.bin'`
 `To '/dd/user/appl/config.dat'`
`End`

TARGET

意義： 指定必須包含於資源中的目標硬體碼名稱。參考先前所說的編譯選項區域，以得知更多關於所用的目標硬體更多的說明。“**Target**”敘述在同一個資源區塊中能出現多次，以便選擇多個目標硬體，若有指定“**Any Target**”敘述，則這個敘述不能使用。

語法： **TARGET** '<目標硬體名稱>'

例子： `BinaryFile 'MYFILE'`
`Begin`
 `Target 'ISA86M'`
 `Target 'ISA68M'`
 ...
`End`

ANYTARGET

意義： 指定碼產生器建立所有的目標硬體碼，都必須整合到資源中。**ISaGRAF** 碼產生器能於相同的“**製作**”命令產生幾個目標硬體碼。假如已指定一個或多個“**Target**”敘述，這個敘述就不能使用。

語法： **ANYTARGET**

例子： **ULongData** 'MYDATA'

Begin

AnyTarget

...

End

FROM

意義： 指定 **BinaryFile** 或 **TextFile** 資源的來源檔 (位於 ISaGRAF 安裝的 PC 上)。用於路徑元件 (磁碟機，目錄，檔名，延伸檔名) 的區隔字元必須遵循 MS-DOS 系統格式。

語法： **FROM** '<來源路徑>'

例子： **BinaryFile** 'MYFILE'

Begin

...

From 'c:\user\config.dat'

To '/dd/user/appl/config.dat'

End

TO

意義： 指定 **BinaryFile** 或 **TextFile** 資源的目的檔 (位於目標硬體系統上)。用於路徑元件 (磁碟機、目錄、檔名、延伸檔名) 的區隔字元必須遵循目標硬體系統格式。

語法： **TO** '<目標硬體路徑>'

例子： **TextFile** 'MYFILE'

Begin

使用者指南

```
...  
From 'c:\user\config.bin'  
To 'dd/user/appl/config.dat'  
End
```

例子

以下是一個完整的資源定義檔例子：

(* 資源定義檔 *)

ULongData 'DATA1' (* 許多數值的集合 *)

Begin

Target 'ISA86M' (* 僅用於這個目標硬體 *)

1, 0, 16#1A2B3C4D, +1, -1 (* 數值 *)

End

VarList 'VLIST1' (* 許多變數的集合 *)

Begin

Target 'ISA86M' (* 僅用於這個目標硬體 *)

Valve1, StateX, Command, Alarm1 (* 變數名稱 *)

End

BinaryFile 'FILE1' (* 二元檔資源 *)

Begin

AnyTarget (* 可用於所有的目標硬體 *)

From 'c:\user\updatef.bin' (* PC 上的原始檔 *)

To 'updatef.cfg' (* PLC 上的目標硬體檔 *)

End

TextDile 'FILE2' (* 文字檔資源 *)

Begin

Target 'ISA68M'

```

From 'c:\nw\nwbd.txt'    (* PC 上的原始檔 *)
To 'nw/dat/nwbd'    (* PLC 上的目標硬體檔 *)
End

```

➡ 資源編譯器

假如資源已被輸入資源定義檔中，於 ISaGRAF 碼產生後將出現一個對話框。按“**開始編譯**”按鈕執行資源編譯器，訊息及錯誤將顯示於視窗中。按“**離開**”避免資源編譯，則資源將不會加入 ISaGRAF 碼中。

➡ 實行

ISaGRAF 不限制資源數、資料列及檔案大小，資源將儲存於產生碼之後。下面是資源目錄的格式 (以 C 表示)：

```

_RESOURCE:
{
    long nbres;    /* 定義的資源數 */
    {
        char name[16]; /* 資源名稱 */
        long type;    /* 資源資料型態 */
        long size;    /* 資料區塊的確實大小 */
        void *data;
        char *path_offset; /* 指到字串的指標 */
    } /* 記錄大小 */
}

```

下面是“type”欄位的可能值：

- 1 = binary file (二元檔)
- 2 = text file (文字檔)
- 3 = ulong data (不帶符號長整數資料) (path_offset 欄此時不使用)
- 4 = variable list (變數集) (path_offset 欄此時不使用)

使用者指南

資源編譯器會根據目標硬體系統格式來轉換文字檔的換列字元。所有的指標都是 32 位元的相對位址偏移值，所有的資源名稱及路徑都以空字元當做字串結束。路徑及資料緊跟著目錄結構之後。

A.14 交互查詢

ISaGRAF 工作平台包含一個交互查詢編輯器，它提供使用者觀看案件程式中宣告的變數與它使用的地方的總關連表。交互查詢的目的乃是列出所有案件宣告的變數，且列出這些變數位用於每個程式原始碼中的位置。交互查詢在變數的生命週期總觀上是相當有用的，它增進了效能，且降低維護時瞭解案件的時間，亦可用於整個案件字典的總觀，所以未用到的變數可以很容易找到，且得知案件的複雜度。

視窗左邊的表列顯示案件所宣告的物件（程式，變數及定義字）與用於案件中的程式庫元件（函數及功能方塊），右邊的表列顯示所選物件出現於程式的地方。

右邊的項目描述包含所在的程式名稱、FC 或 SFC 步驟、轉移條件或測試的編號，以及文字語言的列編號或 LD（或 FBD）的圖形位置。對 Quick LD 圖形而言，項目描述則使用導軌號碼，假如變數被用於輸出（線圈），導軌號碼之後會加上星號（“*”）字元。

從“選項”功能表中設定“顯示未用變數”選項，會將程式未用到的變數一並列上去。

▣ 物件型態選擇

因為案件能群聚大量的宣告物件，所以工具列上的組合框 (combo box) 用來選擇顯示於視窗列表中的物件型態，也就是允許使用者有選擇資訊的進入權。

每次交互查詢重新計算後，會自動切換到“全部”選項，以便顯示完整的表集。

▣ 重新計算交互查詢

“檔案 / 重新計算”命令於任何時間都可根據其他 ISaGRAF 編輯視窗所作的變更來更新交互查詢。

▣ 輸出交互查詢

使用者指南

“**工具 / 輸出**”命令用來寫入完整的交互查詢集到一個 ASCII 文字檔中，然後這個檔案可以用 windows 記事本或小作家等的文字編輯器來開啓。



字典錯誤

“**編輯 / 字典錯誤**”命令顯示一個當案件字典載入後所偵測到的錯誤集的對話框。



統計

“**工具 / 統計**”命令顯示一個案件所宣告物件及變數個數的對話框（根據變數型態及屬性），這個命令的一個特別的用處是得知案件宣告的 I/O 變數總數，以確定可以被編譯（假如你用的 ISaGRAF 版本有 I/O 點數限制）。



物件表中作搜尋

“**編輯 / 尋找**”命令允許使用者直接選取編輯器表列中的物件，假如使用選擇性顯示，且物件不在表列中，將搜尋不到，因此建議在搜尋之前，將工具列上“**全部**”選項選擇起來。



開啓程式

右邊的表列包含選擇的物件出現於資源中或 I/O 連結中的位置。“**編輯 / 開啓程式**”命令讓使用者可以直接開啓物件出現的程式，你也可以在表集中雙按滑鼠左鍵，同樣可以開啓對應的程式。

A.15 使用圖形除錯器

ISaGRAF 包含一個完整的圖形及符號除錯器。於程式管理員視窗中的“除錯”命令執行除錯器來控制應用程式下載到目標硬體 PLC 中，這樣的模式下，除錯器與目標硬體系統間透過硬體連結來做通訊。程式管理員視窗中的“模擬”命令立即執行除錯器及一個完整的目標硬體模擬器，它讓使用者可以測試他的應用程式，而不需等待目標硬體的 I/O 系統完成。除錯視窗包含控制整個應用的命令。

當除錯器開始時，且目標硬體 PLC 與工作平台的應用程式相同，會立即開啓**程式管理視窗**（於除錯模式下），這個視窗中的命令可以用來開啓其他的 ISaGRAF 視窗（圖形與文字編輯器、字典、變數集、I/O 連結..）。所有於除錯期間開啓的視窗都是運作於“**除錯模式**”下，也就是無法做編修。顯示的程式元件（步驟，轉移條件，變數..）將顯示他們現在執行中的狀態或值，於元件上面雙按滑鼠左鍵將可改變目標硬體應用程式的狀態或值。

當處於**模擬模式**下的除錯器，與 ISaGRAF 目標硬體系統之間的通訊將停止，除錯器僅與模擬器視窗通訊。因為這個模式下目標硬體系統不存在，所以除錯器功能表的“**下載**”、“**停止**”或“**啓動**”等命令將不再可用。

A.15.1 除錯器視窗

除錯器視窗僅包含關於整個應用程式狀態的訊息，但可連結到其他的 ISaGRAF 視窗，創造出一個完整的交互作用除錯系統。偵測到的執行期錯誤將顯示於除錯視窗的底部區域。“**選項**”功能表的命令可以用來隱藏、顯示或清除錯誤列表。

控制面板（位於除錯器功能表下方）顯示目標硬體系統的全部狀態及執行期的資訊。可能的目標硬體狀態如下：

登入： 除錯器與目標硬體系統建立通訊。

斷線： 除錯器與目標硬體系統無法連線。確定連結線及通訊參數正確。

使用者指南

沒有應用程式： 連線成功，但是目標硬體系統目前沒有 ISaGRAF 應用程式存在。下載一個應用程式。

應用程式動作中： 連線成功且目標硬體系統中存在一個動作中的應用程式，除錯器現在正建立與這個應用的通訊（假如與工作平台是同一個的話）。

執行中： 目標硬體應用處於“即時”模式中。

停止： 目標硬體應用處於“單步”模式中。

中斷點： 目標硬體應用處於“單步”模式中（因為碰到中斷點）。

致命的錯誤： 目標硬體應用失敗（因為嚴重錯誤發生）。

執行期時間的資訊如下：

允許： 程式設定的時間

現在： 完成上一次執行週期的確實時間

最大： 從應用程式開始所發生的最大週期時間

溢位： 現在時間大於允許時間發生的次數

所有的時間以毫秒為單位，當除錯器處於模擬模式下，時間值將不會顯示。

A.15.2 控制應用程式

“檔案”及“控制”功能表包含安裝命令與 ISaGRAF 目標硬體系統上目前所編輯的 ISaGRAF 應用程式的控制。

注意： 於模擬下，其中某些命令無法使用，因為應用程式由模擬器處理，會自動由 ISaGRAF 工作平台所安裝。



停止目標硬體應用程式

“檔案 / 停止應用程式”命令停止目前執行於 ISaGRAF 目標硬體系統中的應用程式。

□ 啟動目標硬體應用程式

“**檔案 / 開始應用程式**”命令執行存在於目標硬體系統中的應用程式。當應用程式下載後，它會自動執行，所以並不需要使用“**開始**”命令，“**開始**”命令一般則用於“**停止**”命令之後。

注意： 在下載新應用程式之前，目標硬體中的應用程式必須停止（不動作）。



下載應用程式

“**檔案 / 下載應用程式**”命令用於下載應用程式碼到目標硬體系統內。使用者需根據目標硬體系統處理器及應用程式選項來選擇所要下載的碼型態。

顯示版本號碼

“**檔案 / 版本號碼**”命令用於顯示完整的工作平台及目標硬體應用程式的識別號碼。工作平台方面的應用程式是目前 ISaGRAF 工作平台所開啓的應用，而目標硬體方面的應用程式則是執行於目標硬體 PLC 中的應用程式。後面是所顯示的項目：

版本： 應用程式碼的版本，這個號碼是由碼產生器所計算出來的。

日期： 顯示碼建立時的日期及時間。

CRC： 符號表內容的檢查加總，由碼產生器所計算出來的，這個值視字典變數內容而定。

注意： “**版本號碼**”命令於模擬模式下也是可使用的，但於真實除錯模式下，若目標硬體 PLC 並未連結上，則這個命令無法使用。



線上變更

“**檔案 / 更新應用程式**”命令讓使用者完成應用程式的“線上變更”，這個命令於這章稍後會做詳細的說明。當除錯器處於模擬模式中，這個命令無法使用。



即時模式

使用者指南

“**控制 / 即時模式**” 命令於沒有應用程式執行時無法使用。這個命令設定目標硬體應用程式於“即時”模式下：正常模式，也就是由程式所用到的週期時間來觸發執行週期。

單步模式

“**控制 / 單步模式**” 命令於沒有應用程式執行時無法使用。這個命令設定目標硬體應用程式於“單步”模式下：於此模式下，會根據使用者於除錯功能表下“**執行下一個週期**”命令來執行一個週期。

執行一個週期

當目標硬體於單步模式下，“**控制 / 執行下一個週期**”命令執行一個週期。

週期時間

“**控制 / 改變週期時間**”命令讓使用者可以變更程式的週期時間，這個時間就是於除錯器控制列視窗中的“**允許**”欄位值。變更週期時間之前應該先設定“單步”模式。週期時間以毫秒單位的整數值來輸入。

□ 移除所有中斷點

“**控制 / 清除所有中斷點**”命令移除整個應用程式目前所有已設定的中斷點。當除錯器視窗關閉時，已存在的中斷點並不會自動移除。

□ 解除 I/O 變數

“**控制 / 解開所有 I/O 變數**”命令解除應用程式目前鎖住的所有 I/O 變數。當 I/O 變數被鎖住後，輸入或輸出狀態改變並不會連結到對應的 I/O 設備上，依附到這個 I/O 上的變數仍然能由應用程式或除錯器所寫入。當除錯器視窗關閉時，目前被鎖住的 I/O 變數並不會自動解開。

A.15.3 選項

“**選項**”功能表包含控制顯示於除錯器視窗訊息的選項。

□ 通訊參數

當除錯器執行時，可以調整通訊時間參數，但僅有**通訊失敗時間**能夠在這裡設定，其他的通訊參數（鮑率，同位元...）必須從程式管理視窗的“**除錯**”功能表設定。

“**通訊失敗時間**”是目標硬體系統開始回答一個工作平台要求的預留時間，“**週期更新時間**”是由除錯器爲了更新開啓視窗中的資料所送出“**讀取**”要求所需求的時間。

所有顯示及輸入的時間值皆爲**毫秒**單位的整數值。當處於模擬模式下的除錯器，不能設定通訊時間參數。

□ 顯示選項

“**顯示週期時間**”選項讓使用者顯示或隱藏除錯器控制列中的**週期時間**值。當這個選項被設定，所有的週期時間元件（允許，現在，最大，溢位）將顯示及更新，取消這個選項將降低除錯器通訊負擔。

當“**顯示錯誤**”選項被設定，偵測到的執行期錯誤將列於除錯器視窗的底部區域。當取消這個選項，錯誤列表則會關閉，且會降低除錯器顯示及通訊的負荷。“**選項 / 清除錯誤訊息**”命令清除目前顯示於除錯器視窗中的執行期錯誤列表。

“**選項 / 視窗最小化**”命令縮小除錯器視窗，以一個總是位頂端的小面板來顯示，這個面板僅包含應用程式狀態及最常用到的圖形按鍵命令。

A.15.4 “寫入”命令

ISaGRAF 符號除錯器提供許多命令去改變應用程式元件的**值或狀態**。當除錯器視窗開啓時，於元件名稱上或編輯視窗中的圖形上**雙按**滑鼠左鍵來改變元件的值。

□ 變數

於下列視窗中的變數名稱上雙按滑鼠左鍵改變它的狀態：

使用者指南

字典

變數列表或時間圖

LD 或 FBD 程式

I/O 連結

下述的命令提供於除錯對話盒中：

寫入變數成新的值

鎖住變數 (僅用於 I/O 變數)

解開變數 (僅用於鎖住的 I/O 變數)

開始或停止計時器變數 (設定自動更新模式)

符號值用於代表布林假及真值，乃是定義於字典中指定的布林變數的字串。

類比值用於“寫入”命令時必須根據字典的變數定義輸入整數或實數格式值，字串用於“寫入”訊息命令時不能比字典定義的訊息變數容量還長。

□ SFC 物件

欲於應用程式除錯時，觀測 **SFC 程式** 的控制操作，可使用程式管理員視窗下的“**檔案**”功能表下的命令，在此之前你必須先從程式列中選擇一個 SFC 程式。下述是可利用的命令：

開始 SFC 程式：啟動所選的程式（藉由於每個初始步驟內放入 SFC 權杖記號 (token)）

刪除 SFC 程式：刪除所選的程式（藉由移除所有存在的權杖記號）

暫停 SFC 程式：暫停所選程式的執行

重置 SFC 程式：重新開始暫停的程式

對子程式而言，這些命令可對應到“**GSTART**”，“**GKILL**”，“**GFREEZE**”及“**GRST**”等函數。

當應用程式除錯時，於 SFC 編輯視窗中，在所代表的 **SFC 步驟** 圖形上雙按滑鼠左鍵，可看到一個控制操作對話盒，於除錯對話盒中可看到如下的命令：
於步驟**動作**時設立中斷點

於步驟**不動作**時設立中斷點

清除所有加入到此步驟的中斷點

注意：動作與非動作的中斷點不能加到相同的步驟中。

當應用程式除錯時，於 SFC 編輯視窗中，在所代表的 **SFC 轉換條件** 圖形上雙按滑鼠左鍵，可看到一個控制操作對話盒，於除錯對話盒中可看到如下的命令：

加入 **中斷點** 於轉換條件上

清除 加入於轉換條件上的中斷點

手動**清除**轉換條件（移除或加入權杖記號）

有條件清除：於轉換條件之後的步驟上產生一個權杖記號，存在於前面步驟的權杖記號將被移除。**無條件清除**：於轉移條件之後的步驟上產生一個權杖記號，存在於前面步驟的權杖記號不會被移除。

A.15.5 線上變更

“線上變更”讓使用者於程式執行時變更應用程式，它有時對於化學程序是必須的，因為任何的中斷可能造成產品或安全性的破壞。這個功能使用上必須**非常小心**，因為 ISaGRAF 不可能偵測到所有線上變更所產生的衝突。

＝ 應用碼序列

ISaGRAF 提供許多從除錯器進入變數、程式或 I/O 板的可能性，這裡所提的“線上變更”僅應用於碼序列的變更。碼的序列是一個完整的 ST，IL，LD 或 FBD 指令列的集合，對於“開始”與“結束”區的程式而言，碼的序列就是這些程式全部的指令集合，對於 SFC 程式而言，碼的序列就是一個步驟或轉移條件的第二層程式。“線上變更”就是取代一個或多個碼的序列，而不需停止 PLC 的執行。因為控制 SFC 權杖記號是非常危險的，所以**無法更動 SFC 結構**，也就是無法加入、更改編號或移除步驟、轉移條件或 SFC 程式。

▣ 變數

變數的資料庫是應用程式相當有秩序的部分，它能在任何時間被其他程序（對多工的 PLC 而言）所讀寫，亦可以從除錯器更改變數的值，因此 **ISaGRAF** 不允許使用者線上加入、更名或移除變數。你可以在應用程式的第一個版本中保留“未使用”的內部或 I/O 變數，如此可以在以後變更時直接使用它們。

下面是 **ISaGRAF** 目標硬體資料庫中各種變數的樣式，限制將作用在它們上面：

- 宣告的變數

為 **ISaGRAF** 字典所宣告的變數，線上變更時不能改變它們，也不能更改它們的名稱。建議額外宣告一些未使用的變數，以便往後需要變更時使用它們。

- 功能方塊樣例

每一個以“C”或 IEC 撰寫的功能方塊樣例相當於 **ISaGRAF** 目標硬體即時資料庫的資料。當功能方塊樣例加入或移除之後，線上變更即不可行了，所以最好使用字典中宣告的功能方塊樣例，來撰寫於 ST 程式中，而不要於 Quick LD 或 FBD 圖中加入方塊（因為系統會自動宣告新的樣例）。另外，於 **ISaGRAF** 程式庫中變更任何有用的功能方塊也將導致線上變更不可行。

- 步驟

每一個 SFC 步驟相當於一塊儲存 SFC 步驟動態屬性（執行時間及旗標）的資料，加入或移除 SFC 步驟將改變應用程式資料庫，也就無法做線上變更。

- 編譯器所配置的隱藏變數

ISaGRAF 編譯器產生“隱藏”暫時變數，以解決複雜的表示式，改變表示式可能導致不同的隱藏變數，且將導致無法做線上變更。為避免此種情況，

你可以於 ISA.INI 檔中加入一些元件，以便強迫每個程式皆配置最小量的暫時變數（即使有些暫時變數於第一個版本中沒有使用到）。如下所示：

```
[DEBUG]
MNTVboo=8           ; for booleans
MNTVann=4           ; for integers and reals
MNTVtmr=4           ; for timers
MNTVmsg=2           ; for messages
```

當如此設定 ISA.INI 檔時，假如應用程式編譯後產生較多的暫時變數，編譯器會輸出警告訊息。

□ 輸入及輸出

因為 ISaGRAF I/O 系統是非常開放的，所需的變動可以由 OEM 使用硬體的特定元件來實行。ISaGEAF 系統不允許使用者線上加入、連結或移除 I/O 變數，或變更 I/O 板的描述，變更 I/O 板參數或鎖住 I/O 接點等操作則可以利用標準的 OEM 元件及“**OPERATE**”函數來達成。

□ 執行期操作程序

變更執行期的應用程式需做下述的操作：

變更工作平台上的應用程式原始碼

產生新的應用程式碼

使用“**更新**”命令取代“**下載**”命令，下載新的應用程式碼

於 PLC 執行期間使用“**真實更新**”命令切換舊的應用程式成為新的。

上述的程序保證目標硬體 PLC 可以有一個完整且可信賴的執行中的應用程式，且讓使用者以非常安全且有效的方法控制樣本操作的時間，它亦能讓使用者可以經常性的變更案件。不管程序的處理過程，“線上變更”本質上與“**停止、開始及下載**”命令集是一樣的，唯一不同的是變數狀態仍能保存，且切換時間非常短（通常是 1 或 2 個週期時間）。於切換時，沒有一個變數被更動，且所有的內部、輸入及輸出變數都能在應用程式變更前後保持相同的值。切換時，不會執行任何動作，且 SFC 權杖記號不會被移動。

☐ 記憶體需求

爲了支援“線上變更”的能力，目標硬體 PLC 必須有足夠的剩餘記憶體空間來儲存變更版本的應用程式碼。於切換動作時，兩種版本的應用程式碼都必須存放於 PLC 記憶體中。

☐ 限制

如前所述，僅允許碼序列變更，變數定義、應用程式參數及 I/O 連結不能被變更。當下載變更版本的應用程式，ISaGRAF 爲了偵測任何不安全的改變，會對變更與執行中的應用程式加以比較，假如切換會有危險或不可能，則會發生下載錯誤。ISaGRAF 會執行安全性保護，比較符號表檢查加總碼，所以任何變數、程式或 SFC 元件名稱被改變都會偵測到。當切換動作發生時，假如步驟處於動作中，它的非儲存 (N) 行爲將會遺失。

操作

欲更新執行中的應用程式碼，必須實行下述的操作：

在改變執行的應用程式時，強烈建議將目前的案件製作一份不同案件名稱的複製，然後在複製的案件上做變更。

在編輯任何程式之前，使用者應該檢查編輯工具的“**更新日記**”選項是否設定，以方便日後程式的維護。

當一個或多個序列被變更時（沒有更動到 SFC 結構及程式樹狀體系），在下載之前，新的應用程式必須在工作平台上產生。

從新的案件內使用除錯器，使用者必須連結到目標硬體 PLC。假如應用程式名稱被更改過，則無法進入目標硬體的資料庫中。使用者必須執行“**檔案 / 更新應用程式**”命令。

選擇“**稍後更新**”選項，下載變更的應用程式。於傳輸期間這個動作會稍微減慢 PLC 的執行。

當下載完成，使用者可以執行“**檔案 / 真實更新**”命令，於最適當的瞬間做切換，切換動作大約 1 或 2 個週期時間。

當切換正確執行時，將會顯示變更後的應用程式名稱，假如沒有，則執行的應用程式依然爲舊的應用程式。

A.15.6 DDE 動態交換

ISaGRAF 除錯器包含一個 DDE (Dynamic Data Exchange，動態資料交換) 伺服器。爲了動態地顯示變數的現值於非 ISaGRAF 應用程式中，可以於 ISaGRAF 除錯器與其他應用程式之間設立一個 advise 迴圈。

僅有“advise”及“poke”服務爲 ISaGRAF 除錯器的 DDE 伺服器所支援，你也可以使用“request”服務，但是只有已建立於 advise 迴圈的變數才可以，其他的 DDE 服務（如“execute”）則無法使用。當變數建立 advise 迴圈之後，變數值會在每次更動時更新 client 應用程式的值。任何型態的變數都能檢測。動態連結的識別名稱如下：

Service 名稱： “ISaGRAF”

Topic 名稱： ISaGRAF 的案件名稱

Item 名稱：變數名稱

假如變數爲區域變數，它的名稱必須加上父程式名稱（包含於括弧內），如下之語法：

變數名稱(程式名稱)

ISaGRAF DDE 伺服器僅用於目前除錯器所連結的 ISaGRAF 應用程式上。ISaGRAF 伺服器最多只可以連結 **256** 個變數。不管是執行於真實模式或模擬模式下的 ISaGRAF 除錯器，都可以使用 DDE 伺服器。伺服器的更新週期得視除錯器與 ISaGRAF 目標硬體系統或模擬器之間的通訊時間而定。

A.16 檢測變數集

除錯視窗內的“**檢測**”功能表中的“**檢測變數集**”命令讓使用者可以建立非連續的變數集。當應用程式處於除錯模式下時，可以建立變數集。這個變數集可以儲存於磁碟上，且可以於除錯模式下再度被打開。一個集合最多可以包含 **32** 個變數，且可以混合不同型態的變數，全域及區域的變數也都能插入同一個集合中。變數集合只能用於一個案件中。當應用程式做功能測試時，變數集合相當的有用。它允許使用者觀看控制程序的某些部分。當除錯 ST 及 IL 文字式程式時，變數集合亦是相當有用的，使用者可以很容易地將用於程式中的變數集合起來，以便於控制或監看程式指令的執行過程。

ISaGRAF 會顯示集合中每一個變數的名稱、目前數值和註解。使用滑鼠拖曳集合標題列的分隔線可以更改每一行的寬度。

☐ 儲存集合於硬碟中

“**檔案**”功能表中的命令用於產生、開啓及儲存變數集。ISaGRAF 並未限制一個案件的變數集合個數。當儲存變數集至磁碟上時，必須遵循下列的命名規則：

名稱不可以超過 **8** 個字元

第一個字元必須是**英文字母**

其餘的字元可以是英文字母、阿拉伯數字或底線符號

名稱與大小寫無關

集合編輯器於相同的視窗中不能一次顯示多個變數集，然而你可以執行集合編輯器多次，以便於可以同時監測不同的集合。



於集合中插入變數

“**編輯 / 插入**”命令插入變數至集合中。變數名稱可以選取案件字典中的物件，因此使用者不需要手動輸入變數名稱。變數會插入於目前選取的變數之前。一個集合不能包含多於 **32** 個變數，在相同的集合中，同一個變數不能出現兩次以上。

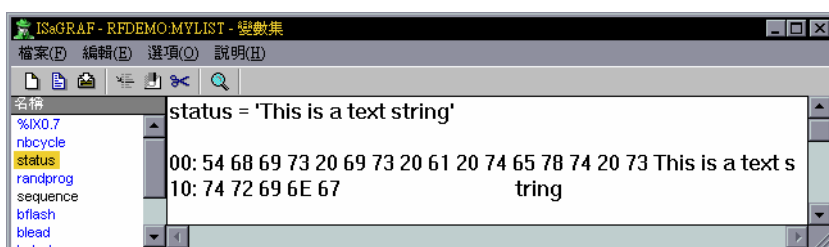
更改選取的變數

“**編輯 / 變更**” 命令以另一個變數取代目前選取的變數，你亦可以使用 “**剪下**” 命令來移除集合中所選取的變數。

二元列印顯示

於任何時刻，你都可以切換表列或 “二元列印” 觀看模式。按工具列之 “放大” 按鈕或使用 “**選項 / 二元列印**” 命令來切換觀看模式。

於 “二元列印” 模式下，僅能顯示一個變數值，它的值以數值/符號格式顯示於視窗頂端，且以 “二元列印” 模式顯示，這個模式允許你以 16 進位的方式來檢測變數值。



當訊息字串包含不可列印字元時，“二元列印” 的顯示方式將會有相當的幫助。

A.17 除錯 ST 與 IL 程式

ST 及 IL 程式於模擬或線上除錯時，無法修改程式內容。

IL 對 IL 程式而言，指令格式化成效列的樣子，指令中所用到的變數目前值顯示於相同列上，你可以於指令上雙按滑鼠，可改變對應的變數值。

ST 對 ST 程式而言，檢測集視窗被植入編輯視窗中，你可以拖曳視窗分隔線，改變視窗的大小。

每一個表列中的變數，ISaGRAF 顯示它的名稱、目前值及註解文字。你可以拖曳標題列中的欄位分隔線，改變欄位大小。

▢ 儲存集合於硬碟中

“**檔案 / 儲存表合**”命令儲存變數集於硬碟中。當你每一次在除錯模式下開啓 ST 或 IL 程式時，變數集就會自動重新載入。執行除錯視窗下的“**工具 / 檢測變數集**”命令可以讓你使用檢測集合工具來自由地開啓或修改集合。



於集合中插入變數

“**編輯 / 插入變數**”命令插入變數至集合中。變數名稱可以選取案件字典中的物件，因此使用者不需要手動輸入變數名稱。變數會插入於目前選取的變數之前。一個集合不能包含多於 **32** 個變數，在相同的集合中，同一個變數不能出現兩次以上。



當 ST 程式中的變數名稱被反白時，按下工具列上的按鈕或執行“**編輯 / 檢測選取物**”命令來直接地傳送此變數到嵌入的檢測集合中。



更改選取的變數

“**編輯 / 變更變數**”命令以另一個變數取代目前選取的變數，你亦可以使用“**剪下變數**”命令來移除集合中所選取的變數。

A.18 使用焦點除錯

ISaGRAF 焦點 (Spotlight) 工具允許使用者定義除錯時的圖形或表列。圖形項目必須連結到 ISaGRAF 案件的變數，且必須於“線上”模式（除錯或模擬）下做定義。

欲改變變數值，可於圖形或表列上雙按滑鼠左鍵，或者選取項目後按 ENTER 鍵也可以。

你可以使用“**檔案 / 鎖住**”命令鎖住這個文件（拒絕任何變更）。當文件被鎖住時，你仍能於變數符號上雙按滑鼠左鍵改變它的值。

A.18.1 建立圖形外觀

圖形由背景圖 (bitmap 或 metafile) 與圖形項目集所組成。欲輸入圖形，必須執行下述的步驟：插入背景圖、插入圖形項目、連結物件與變數。



背景圖

背景圖可以是“bitmap”(.BMP) 或“metafile”(.WMF) 檔案。圖形區內含括的圖形數目並沒有限制，也可以搬移圖形或加以放大。背景圖並不會出現於列表外觀中。背景圖得以其他工具來建立，因為焦點不包含繪圖工具。“**選項 / 背景顏色**”命令用於將圖形外觀中的空白區填滿。

注意： Bitmap 將耗去大量的記憶體，建議盡量修改圖形大小，且刪去未用的空間。



純文字顯示

“純文字”項目是顯示於長方形區域中的文字，此文字將顯示所對應之變數值，這種項目也可以連結訊息字串變數。

使用者指南

文字所處的長方形區間可以填滿顏色，也可以是透明的。顯示的文字字型在項目縮放時，會自動調整文字大小，以滿足長方形的高度。

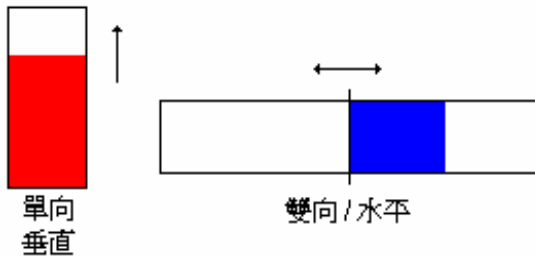


單向與雙向長條圖

長條圖是一種以顏色部分填滿的長方形，表示所依附變數的數值大小。長方形剩餘的區域可以選擇以色彩加以填滿。長條圖可以是水平或垂直方向。

單向長條圖可以向下述之方向填滿：向上、向下、向左或向右。

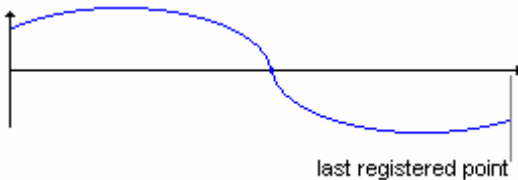
雙向長條圖可以根據依附變數的值向正或負方向成長。它的最大允許值同時適用在正或負的尺寸上。



曲線

ISaGRAF 可以在文件內插入曲線。曲線顯示依附變數的變動記錄。雖然它不是精密的測量工具，但它可以得知多個變數間的不同步關係。

曲線儲存變數最後 200 個值。當曲線項目改變大小时，其取樣數目不變。



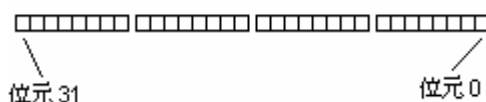
布林圖示

“布林圖示”項目用於顯示二元狀態。一個圖示 (.ICO) 檔被定義成“假”或 0 值；另一個圖示定義成另一個非零值。由於焦點不包括圖示編輯器，圖示檔必須由其它工具預先準備好。



位元欄

“位元欄”項目以 32 位元整數值的圖形面板來顯示。最低位元永遠顯示於右邊。為避免可能引起顯示訊息混亂，所以不建議對其他的資料型態（如實數）使用位元欄形式。



選擇、移動或改變項目大小

大多數的編輯命令都需要選取圖形物件，焦點允許在圖形區域內選取一個或多個已存在的物件。選取物件時，必須先於工具列上點選“**選取**”（箭頭）按鍵。若僅選取一個物件，使用者僅必須於物件符號上點選滑鼠。若選取多個物件，可用滑鼠在繪圖區上拖曳一個長方形，所有與長方形相交的物件都會被標記為“**被選取**”。選取的物件外圍會畫上小的黑方點。

選取新物件時，先前選取的物件將不再被選取。想要取消已存在的選擇，只要於選取物件的長方形外的空白區點選即可。

想要移動物件，必須先選取它們，然後游標移到選取物件的邊框上拖曳至其他位置。

想要改變物件大小，必須先選取它們，然後將滑鼠游標移至選取區邊沿的小方點上，沿適當的方向拖曳滑鼠來改變其大小。背景圖亦可以改變大小，相對的點陣圖 (bitmap) 或 metafile 將按指定的矩形拉伸。



項目群組 / 取消群組

你可以把多個項目群組在一起，進而把它們當作一個項目處理。想要產生群組，於圖形外觀中選取項目，且執行“**編輯 / 群組**”命令。“**編輯 / 取消群組**”命令可以分解群組成個別的項目。

群組可以包含背景圖，群組亦可以包含另一個群組。項目群組後，它們的樣式就不能加以改變了，群組的項目仍然可以顯示，但是無法變更（雙按滑鼠左鍵）依附變數的值。

一個群組在表列外觀中只以一行顯示。

A.18.2 表列外觀



於任何時候點選這個按鍵切換圖形或表列外觀，亦能使用“**選項 / 表列圖形外觀**”命令。

在表列外觀下，項目顯示在一個典型的表列框中，每個項目的高度根據它的繪製樣式而定。背景圖（點陣圖及 metafile）於表列外觀中沒有顯示。表列外觀中可選取項目，且能設定項目樣式或改變變數的值。此模式中無法使用多重選取。



你能使用“**編輯 / 在表列中移動**”命令重新排列表列中的項目，欲移動的項目必須先加以選取。

A.18.3 定義項目樣式

於圖形區中的項目符號上雙按滑鼠左鍵，或執行“**編輯 / 設定項目樣式**”命令變更選取項目之圖形樣式。當新加入項目時亦會開啓“樣式”對話框，它包含下述之訊息可讓使用者選擇：

▣ 圖形樣式及設定：

項目的顯示樣式（純文字、長條圖、曲線...）可以動態改變。如果使用了前景和背景顏色，可以使用對應的列示盒來改變顏色。當樣式為“布林圖示”時，必須指定相關聯的 .ICO 檔路徑，使用者可以使用控制盒旁的“...”按鈕，瀏覽存在於磁碟上的圖示檔。

▣ 尺寸：

為長條圖及曲線可以顯示的最大值。對雙向長條圖及曲線而言，正及負軸使用相同的值。

▣ 變數名稱：

當“名稱”欄位為作用的欄位時，按編輯盒旁的“...”按鈕，可讓使用者找尋已於案件字典中宣告的變數名稱。

▣ 標題：

標題可於圖形外觀中顯示於圖形項目旁。你可以自由選擇標題文字的位置（上、下、左或右）及它的內容。標題可以由變數名稱與它的值任意組合而成。標題於表列外觀中沒有影響。

▣ 命令變數：

假如“命令變數”選項被設定，於除錯期間可於項目圖形符號上雙按滑鼠左鍵改變所連結的變數值。

A.18.4 “檔案”功能表命令

“檔案”功能表命令允許使用者管理整個文件。

 “檔案”功能表下的“開新文件”命令開始新文件的編輯。ISaGRAF 並不限制案件所定義的文件個數。編輯新圖形之前，先前開啓的圖形將關閉，焦點無法同時編輯多個圖形，然而可以同時開啓多個焦點視窗，也就可以一次編輯不同的文件。



“**檔案**”功能表下的“**開舊文件**”命令允許使用者關閉目前所編輯的文件，且開啓另一文件，也就是新選的文件取代了目前編輯視窗中的文件。當選擇新文件時，可用“**刪除**”按鈕刪除已存在的檔案，以便清理案件的目錄結構。當刪除圖形時，其所用的圖示及點陣圖檔並不會一併刪除。



“**檔案**”功能表下的“**儲存文件**”命令將現在所編輯的文件儲存至磁碟上。假如文件尚未命名，此時使用者必須給予它一個名稱。命名規則如下：

- 名稱長度不可超過**8**個字元
- 第一個字元必須為**英文字母**
- 其餘字元可以是**英文字母**、**數字**或**底線符號**
- 名稱與大小寫無關

“**檔案**”功能表下的“**另存新文件**”命令允許使用者將目前所編輯的文件以另一名稱儲存。

A.18.5 ISaGRAF 3.2 版使用者注意事項

焦點可以讀取 ISaGRAF 3.0 或 3.2 版所建立的圖形或時間圖集，這些檔案可於“**開舊文件**”對話框中看到。焦點可讀取它們，也可以更改它們。

當打開 ISaGRAF 3.2 版的圖形時，會自動將此文件標記為“**鎖住**”的狀態，假如你要更改圖形時，可移除“**檔案**”功能表下的“**鎖住**”選項。

當開啓 ISaGRAF 3.2 版的圖形或時間圖集時，焦點會建議你以焦點的格式來儲存它。當關閉此種文件時，會自動開啓“**另存新文件**”對話框。

A.19 上載應用程式

ISaGRAF 可以上載存於目標硬體內的應用程式。透過通訊的方法與目標硬體連結，再上載壓縮的原始碼 (embedded zipped source code, EZS)，然後回復至工作平台中。

此項功能僅支援 3.22 版之後的目標硬體，且必須先將壓縮的原始碼植入應用程式中。這項功能 (植入原始碼至目標硬體中) 是可自由選擇的。

A.19.1 上載案件

從 ISaGRAF 案件管理員的“檔案”功能表下可執行“上載”對話框。上載時並不會參考到工作平台中已存在的案件，也就是上載動作與所選取的案件無關。上載目標硬體中的應用程式必須：

- 1 – 確定與目標硬體間已正確連線
- 2 – 根據之間的連接關係設定通訊參數
- 3 – 按下“執行”按鈕

上載動作及解壓縮程序可能花費一些時間。當上載完成或發生錯誤時會出現訊息對話框提醒你。

上載所產生的案件名稱與目標硬體中的應用程式名稱一樣。假如工作平台中早已有相同的案件名稱存在，將會提醒你要覆蓋它或以另一名稱來儲存。當上載完成時，你無法取消載入的動作。上載的案件此時已準備好了，隨時可以開啓它。

☞ 可能發生的錯誤

當上載案件發生錯誤時，訊息將會出現於“上載”對話框中。可能的錯誤如下：

無法與目標硬體建立通訊連結

使用者指南

連結的目標硬體為 3.22 之前的版本

目標硬體內無應用程式

目標硬體內未植入 EZS

A.19.2 通訊設定

按下“**設定**”按鈕後，使用者將能設定 ISaGRAF 工作平台與目標硬體間的連結參數。在上載前，必須先確定參數與連結的目標硬體一致。

A.19.3 準備上載案件

假如你要能上載案件，你必須於 ISaGRAF 碼產生前開啓植入壓縮原始碼功能，使用“**編譯選項**”對話框的“**上載**”按鈕可達到此要求。對話框另有一些選項可選取哪些原始碼要植入，若未加以選取，則僅植入最基本的原始碼。

重點事項：程式庫元件無法植入至下載的應用程式中，這些元件包括函數、功能方塊、I/O 板及設備。

☐ 可選擇的檔案

除了最基本的原始碼外，亦可以植入其他的檔案，但是會增加目標硬體的需
求記憶空間。

案件描述：若未植入，則上載後的案件描述內容僅會記錄上載時間。

密碼保護：若未植入，則上載後的案件將無密碼保護。

未連結 I/O 連結點的註解：ISaGRAF 提供使用者自由輸入未連結 I/O 的描述
文字，未選取此選項表示僅使用已連結的 I/O 點。

變更紀錄：整個案件的變更紀錄。

日記檔：每個程式的日記檔包含使用者自行撰寫的註記、以及程式的編譯輸出訊息紀錄。植入此日記檔可能會佔用相當多的目標硬體記憶體。

變數集與時間圖：這些檔案用於除錯期間，包含變數名稱集與監視時間圖。

圖形、圖示及點陣圖：包含 ISaGRAF 圖形及所用到的圖示及點陣圖（假如這些檔案位於案件目錄下）。注意：植入這些檔案可能會耗去大量的目標硬體記憶體。

A.19.4 壓縮檔如何儲存於目標硬體中

植入的壓縮原始檔 (EVS) 以資源的方式儲存，所產生的資源稱為“**EVS**”，假如你植入了此原始碼，就不能將此名稱用於其他資源上。植入的原始碼並不會增加資源定義的限制，也不會影響使用者所寫的資源定義。

請參考碼產生器章節，以得知更多的資源細節及資訊。

A.19.5 目標硬體的記憶體需求

植入的壓縮原始 (EVS) 碼與應用程式碼一起儲存於目標硬體中，它需要佔用額外的記憶體空間。若你植入的原始檔沒有選取額外的選項（最小的 EVS），則一般來講，它的大小約為應用程式碼的 1.5 倍，所以整個下載的大小約為未植入 EVS 之前的 2.5 倍。

某些目標硬體系統以節段區分記憶體，可能會造成特殊的限制，因為 EVS 以資源的型態儲存於應用程式碼中，它們會佔用相同的記憶節段。

A.19.6 關於上載案件

上載案件包含再編譯所需的所有檔案及資料，是先前編譯所選取的選項，它也可能包含案件描述及程式日記檔等輔助檔。

你必須於除錯或監視之前編譯案件。**警告：**ISaGRAF 會使用編譯時的日期記號來比較應用程式間的版本，所以當你開啓除錯器時，將會出現工作平台與目標硬體版本不同的訊息。

注意：程式庫不會與植入的原始碼一起下載，你在重新編譯上載應用程式之前，必須確定你的 ISaGRAF 工作平台內已安裝了正確的程式庫函數及功能方塊。

A.19.7 相容性

上載功能僅能於 ISaGRAF 目標硬體及工作平台 3.22 版即以後的版本使用。

ISaGRAF 目標硬體 3.03 到 3.21 版並沒有限制植入的壓縮原始碼 (EZS)，因為 EZS 會以標準的資源格式儲存於應用程式中，但是植入的訊息將無法上載，因為這些目標硬體並不支援這些通訊服務。

A.20 使用診斷工具

“**診斷工具**” (Diagnosis) 是 ISaGRAF 除錯工具下的一部份，它讓終端使用者工作於事先定好的變數集中，以方便做測試及程序控制。ISaGRAF 除錯器是個非常強大的工具，且包含高階的功能。診斷工具提供一個安全的方法來控制目標硬體應用程式，以便於最後的執行操作及維修。ISaGRAF 診斷工具直接由程式管理員的 ISaGRAF 群組內的診斷工具 (Diagnosis) 來執行，你可以於下面的圖示上雙按滑鼠左鍵來啟動它：



已存在的案件集將顯示於對話框中，它讓使用者可以對早已下載的 ISaGRAF 應用程式來執行有限制功能的 ISaGRAF 除錯器。按 “**確定**” 按鈕啟動所選案件的有限制功能的除錯器。按 “**取消**” 按鈕關閉對話框。“**設定**” 命令用來設定 ISaGRAF 工作平台與目標 PLC 之間的通訊連結。參考 “**程式管理**” 章節，以得到更多關於這個命令的資訊。

注意： ISaGRAF 診斷工具 (受限制的除錯器) 不能用來下載、停止或更新目標 PLC 內的應用程式。假如執行於 PLC 內的應用程式與診斷工具對話框所選的案件不同，將不會執行任何動作。

當受限制的 ISaGRAF 除錯器執行時，且正確地連結到目標應用程式，可利用下述的命令：

檢測變數集合

使用焦點來檢測圖形文件

注意： 當更新圖形時，僅有被定義成 “**命令變數**” 的變數可以被使用者變更。任何於變數集或時間圖中的變數都可被使用者變更。

A.21 使用 ISaGRAF 模擬器

當位在程式管理視窗上的“除錯”功能表中的“模擬”命令執行時，除錯器中的 ISaGRAF 核心模擬器就會被啟動。核心模擬器是一個完整的 ISaGRAF 目標硬體系統，完全支援 ISaGRAF 標準特性、所有的 C 函式和由 ICS Triplex ISaGRAF Inc. 公司所衍生出的功能方塊。在視窗中，I/O 板被模擬成圖形型式。任何型態的 I/O 板都可以被模擬。在 I/O 連結時定義成“虛擬板”的 I/O 板也同樣地出現在模擬視窗中。

A.21.1 除錯器的連接

核心模擬器支援所有可與 ISaGRAF 除錯器溝通的通訊方式，所以可在模擬期間使用任何除錯方式。核心模擬器總是工作在現行的 ISaGRAF 應用程式。在模擬期間，除錯器命令“開始”，“停止”，“下載”或“更新”不再適用。在建立目標硬體碼之前，假如在編譯器選項中沒有選取“模擬”目標硬體，模擬器就不能被使用。關閉模擬器視窗意指除錯器視窗（和除錯時期所開啓的任何 ISaGRAF 視窗）也同樣關閉。

A.21.2 I/O 模擬

出現在模擬器視窗中的 I/O 板，會標記出它們的名稱和位置號碼。任何 ISaGRAF 的 I/O 標準型態（布林、類比或是訊息）都能作處理。輸入板的連結點顯示成特殊按鈕和欄位。輸出板的連結點顯示成圖形狀態燈和資料欄。

✎ ➡ **布林輸入：**一個布林輸入表示成一個綠色方塊按鈕。連結點號碼和 I/O 按鈕一同顯示出來。當按鈕被按下時，輸入值將設定成真。點取按鈕可更改相關的 I/O 值。只有當按鈕壓下才可使用滑鼠右鍵來設定輸入。

✎ ➡ **布林輸出：**布林輸出以一個小圓圈來表示。連結點號碼顯示在 I/O 按鈕旁邊。當小圓圈亮起時，輸出值將被設定成真。

 **類比輸入**：一個類比輸入連結點即是一個可鍵入相關輸入值的簡單數值欄。點選方塊區將顯示出游標，然後就可以輸入一新數值到連結點，輸入完之後不必按 **ENTER** 鍵。類比輸入可以被輸入成十進位或是十六進位。使用上/下鍵來增加或是減少現有數值。

 **類比輸出**：一個類比輸出連結點即是一個可輸出相關值的數值欄。輸出值可以被輸出成十進位或是十六進位。使用者不能於此處更改輸出連結點。

 **訊息輸入**：一個訊息輸入連結點即是一個可鍵入相關輸入字串的簡單的文字欄框。點選方塊區將顯示輸入游標，然後就可以輸入一新字串到連結點，輸入完之後不必按 **ENTER** 鍵。

 **訊息輸出**：一個字串輸出連結點即是一個文字輸出欄框，使用者在輸出連結點上不能執行任何動作。

A.21.3 資料庫元件

由 ICS Triplex ISaGRAF Inc. 發展的 ISaGRAF 模擬器完全支援標準轉換、函數和功能方塊。所支援的物件如下：

▣ 轉換函數：

bcd , scale

▣ 函數：

abs , acos , ArCreate , ArRead , ArWrite , ascii , asin , atan , char , cos , delete , expt , find , insert , left , limit , log , max , mid , min , mlen , mod , mux4 , mux8 , odd , rand , replace , right , rol , ror , sel , shl , shr , sin , sqrt , tan , trunc

▣ 功能方塊：

使用者指南

average , blink , cmp , ctd , ctu , ctud , derivate , f_trig , hyster , integral ,
lim_alm , r_trig , rs , sema , sr , stackint , tof , ton , tp

使用者定義的轉換、“C” 函數和功能方塊通常都不和 ISaGRAF 模擬器整合在一起。基本上，這些物件被設計用來使用目標硬體系統的軟硬體資源，這些資源一般並不適用於 Windows 系統。ISaGRAF 模擬器對使用者定義的轉換、“C” 函數和功能方塊提供下列標準行為：

當新的轉換被模擬器處理時，它會被空的轉換所取代。意思是說類比變數的物理值總是等於電位值（表示輸入或顯示在模擬器面板上的數值）。

當新的“C” 函數或功能方塊被模擬器執行時，它不會執行任何的動作，結果值將不被設定。

A.21.4 選項

“**選項**” 功能表命令讓使用者能夠去控制模擬器面板上的 I/O 顯示。在所有的除錯期間使用者都能夠設定或移除這些選項。

⇒ 當“**顯示顏色**” 選項被設定時，I/O 連結點就被顯示成彩色 bitmap 圖。假如在一些不能顯示彩色的 LCD 螢幕，使用者應該移除這個選項來得到純粹黑白的輸出入圖形。

⇒ 當“**變數名稱**” 選項被設定時，一個標籤和所連結的變數名稱將被顯示在所有的 I/O 連結點旁邊。移除這個選項讓使用者能夠減少模擬器面板尺寸。

⇒ 當“**十六進位值**” 選項被設定時，任何輸出入類比連結點都將以 16 進位值顯示或輸入。

⇒ 當“**總是在最頂層**” 選項設定時，即使輸入的焦點視窗是在其他視窗，模擬器視窗依然是看得見的。

A.21.5 儲存及回復輸入狀態

使用 ISaGRAF 模擬器，可以手動強制輸入型態的接點，也可以使用按鈕動作，以及編輯模擬控制面板。你可以於任何時候使用“工具”功能表的下述命令來儲存及回復所有輸入點的狀態：

載入輸入表 設定輸入接點與之前“儲存輸入表”命令所產生的檔案內容相同的值。

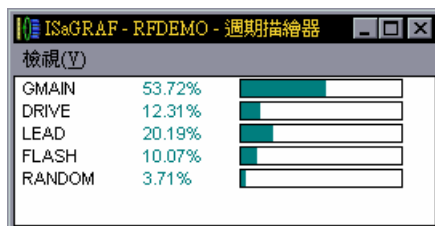
儲存輸入表 儲存輸入接點的狀態至檔案中。如此便可於稍後使用“載入輸入表”命令回復。儲存之檔案置放置於案件目錄下。

注意： 僅有已命名的連接點（有連結變數者）才會儲存。

A.21.6 週期描繪器

ISaGRAF 週期描繪器 (Cycle Profiler) 是一個強力的診斷器，它顯示各種程式、函數及功能方塊間所分配的時間比例。這個工具於快速診斷應用程式的執行效能上相當有用，且引導程式設計者知道哪部分程式碼需做最佳化。

週期描繪器由 ISaGRAF 模擬視窗下的“工具 / 週期描繪器”命令來執行，它顯示每個程式、函數或功能方塊所花費的時間百分比值：



當“檢視 / 平均值”選項被設定後，顯示的資訊為應用程式開始或上次“檢視 / 重置”命令之後的百分比平均值。

使用者指南

假如“**檢視 / 平均值**”選項未設定，顯示的資訊為前一個週期的測量值。
當應用程式處於“**單步**”模式之下時，你可以使用此種模式顯示上次週期的測量值。

使用“**檢視 / 複製**”命令以 ASCII 格式複製程式名稱及百分比至 Windows 剪貼簿內，然後資料可以被貼入文字編輯器或試算表內。

☐ 注意：

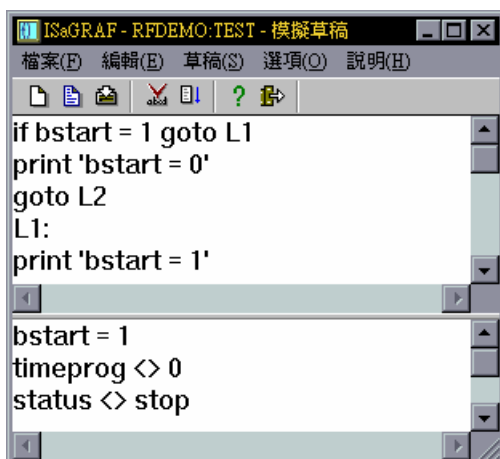
這並不是很精確的測量值，百分比是以 TIC 指令數乘上各指令所執行的時間計算出來的。此種計算方法並不包含“C”函數及功能方塊所花費的時間。函數或功能方塊的顯示值為所有呼叫時間的加總。

時間的計算是根據 TIC 碼計算出來的，假如實際的應用程式碼是以“C”語言編譯產生的，則所顯示的值將不可信賴。

A.21.7 模擬草稿

ISaGRAF 模擬器包含建立及執行模擬草稿的工具。草稿使用類似 ST 文字式語言的方式來描述，且用來搭配 ISaGRAF 模擬器做自動化測試。

模擬草稿編輯器由模擬器視窗下的“**工具 / 模擬草稿**”命令執行。下面是草稿編輯器的例子：



上層的視窗是一個文字編輯器，用來輸入草稿命令。這個命令編輯器包含高階元件，如可以使用滑鼠選取變數符號。你能使用“**選項**”功能表下的命令設定定位點寬度及選擇字型。

下層的視窗顯示草稿執行時的所有輸出訊息。上下層視窗間的分隔線可以自由拖曳，以改變視窗大小。輸出視窗可以於草稿編輯時隱藏，但是每次草稿執行時皆會自動開啓。

二 編輯草稿

使用“**檔案**”功能表下的命令管理草稿檔：

開新草稿 產生一個新的未命名草稿

開舊草稿 從檔案中載入已存在的草稿

儲存草稿 儲存草稿文字及輸出視窗內容至磁碟中（置放於案件目錄下）

另存新草稿 以另一個名稱儲存草稿

每一個草稿將於 ISaGRAF 案件目錄下產生兩個檔案：

<草稿名稱>.SCC 草稿文字（指令）

<草稿名稱>.SCO 輸出視窗內容

兩個檔案皆是標準的文字檔，可以以任何的文字編輯器來開啓。



當編輯草稿時，你能使用“**編輯 / 插入符號**”命令選取已宣告的變數名稱，然後將之插入於游標處。

▢ 執行草稿

草稿於執行前必須加以檢查及編譯，假如有需要的話，執行“執行草稿”命令時會自動檢查草稿。使用下述之“草稿”功能表下的命令：



檢查 檢查語法，且編譯草稿



執行草稿 開始執行目前編輯的草稿

假如為新的未命名草稿，在檢查之前必須先加以儲存（且必須給予一個草稿名稱）。假如草稿已命名，在檢查之前會自動儲存至磁碟上。

當草稿執行時，無法更改它的內容。當草稿執行至結束行時，會出現通知訊息。執行中的草稿可以使用“草稿”功能表的下述命令加以停止：



終止草稿 結束執行中的草稿

草稿於每個週期間會加以執行，假如程式中有無窮迴圈的情況，ISaGRAF 模擬器會中斷此迴圈，所以 ISaGRAF 週期仍然可以繼續執行，其他的 ISaGRAF 應用程式也就不會被凍結住 (block)。假如於同一個週期內執行了多次相同的“標籤” (label) 的話，ISaGRAF 草稿解譯器會自動中斷草稿的執行，你亦可以使用“cycle”或“wait”指令來加以中斷。

▢ 草稿描述語言

草稿描述語言是非常簡單的文字式語言，它類似 ST，但是每個指令必須寫在不同列，且不需要於最後以分號終結。使用工具列上的下述按鈕將可用的指令插入於游標處：



插入指令 (關鍵字及說明)

有各種不同型態的指令，第一個為變數指令 (強制)：

:= 指定

另有一些指令用於輸出訊息至輸出視窗：

Print 輸出文字字串或變數值

PrintTime 輸出目前的時間

另有一些指令用於草稿執行與 ISaGRAF 週期間的同步：

Cycle 讓 ISaGRAF 模擬器執行一個週期

Wait 等待一段指定的時間

另有一些指令用於控制草稿之指令執行順序：

Labels 可以置放於草稿中的任何地方

Goto 無條件跳躍至標籤

If goto 有條件跳躍至標籤

End 終止草稿

草稿所用的語言不分大小寫。註解可以放置於任何文字列的最後面，但必須放置於 “(” 及 “)” 之間，或是加於 “;” 之後。

“:=” 指定

意義： 強制 ISaGRAF 變數的值，變數可以是內部變數、輸入連結點或輸出連結點。

語法： <變數名稱> := <常數表示式>

<變數名稱> = <常數表示式>

引數： <變數名稱> 為應用程式中宣告的變數符號，或是使用 “%” 語法的變數直接表示法。

<常數表示式> 為與變數同型態的常數表示式。對布林型態而言，“0” 及 “1” 可以用來替代 “FALSE” 或 “TRUE”。對計時器型態而言，前置詞 “T#” 或 “TIME#” 可以省略。

注意： 於草稿中強制輸入型態的變數並不需先加以鎖住。當輸入變數被強制時，相對應的輸入連結點會跟著更新。

警告： 不要強制有附加轉換的輸入或輸出類比變數，草稿並不支援轉換函數或轉換表。

使用者指南

範例： MyBooVar := 1 (* 與 TRUE 同義 *)

MyIntVar := 1234

MyRealVar := 1.2345

MyMsgVar := 'Hello'

MyTmrVar := t#12s

Print

意義： 輸出字串或變數值至輸出視窗。文字以另一新列輸出。

語法： **Print** '<文字>'

Print <變數名稱>

引數： <文字> 是任何以單引號括住的文字字串。

<變數名稱> 為應用程式中宣告的變數符號，或是使用 “%” 語法的變數直接表示法。

注意： 變數值根據 IEC 的格式輸出。

範例： **Print** 'Hello'

Print MyBooVar

輸出： Hello

MyBooVar = TRUE

PrintTime

意義： 輸出目前的時間戳記至輸出視窗。文字以另一新列輸出。

語法： **PrintTime**

注意： 時間戳記格式根據 Windows 系統的目前設定值輸出。

範例：*Print 'Time now is :'*

PrintTime

輸出：*Time now is :*

15:45:22

Cycle

意義：*停滯草稿執行，直到下個 ISaGRAF 週期開始為止。*

語法：*Cycle*

注意：*草稿於 ISaGRAF 週期開始時執行。假如模擬器處於‘單步’模式下，“Cycle”指令將緊跟週期開始之後，其餘的草稿指令會於下個“執行一個週期”命令之後執行。*

範例：*(* ISaGRAF 的程式將 A 複製至 B *)*

A := 0

Cycle

Print B

A := 1

Print B (這個週期的程式尚未執行 / B 未設定為 1 *)*

Cycle

Print B

輸出：*B = 0*

B = 0

B = 1

Wait

意義：*停滯草稿執行，直到延遲時間經過為止。*

使用者指南

語法： **Wait** <延遲時間>

引數： <延遲時間> 表示式為 IEC 的時間常數表示法，但是“T#”或“TIME#”前置詞可以省略。延遲時間必須介於 10 毫秒至 1 小時之間。

注意： “Wait” 指定並不是很精確，準確性端視 Windows 系統而定。延遲時間的誤差為正負一個 ISaGRAF 週期時間。當執行至“Wait”指令時，ISaGRAF 會等待延遲時間經過後再繼續執行草稿。

範例： *PrintTime*

Wait 2s

Print Time

輸出： 15:45:27

15:45:29

Labels

意義： Label (標籤) 可以放置於草稿的任何地方，它可以搭配“Goto”指令一起使用，以改變草稿指令的執行順序。

語法： <標籤名稱>：

引數： <標籤名稱> 為唯一的名稱，必須根據 ISaGRAF 變數的命名規則：最多 16 個字元、以英文字母開始、其餘字元可以是英文字母、數字或底線字元。標籤名稱必須於後面緊跟著“:”字元。

注意： 標籤必須單獨一列，同一列上不可以放置任何指令。標籤名稱不可與宣告的 ISaGRAF 變數符號相同。

範例： (* 無窮迴圈的草稿例子 *)

loop:

PrintTime

Wait 1s

Goto loop

Goto

意義： 無條件跳躍至標籤

語法： **Goto** <標籤名稱>

引數： <標籤名稱> 是草稿中所定義的標籤名稱。

注意： 往前跳躍是被允許的。假如為無窮迴圈，草稿每執行一次迴圈會自動產生中斷，如此便可讓 ISaGRAF 週期持續下去。

範例： *Print 'Before Jump'*

Goto MyLabel

Print 'Within Jump' (永遠不會執行 *)*

MyLabel:

Print 'After Jump'

輸出： *Before Jump*

After Jump

If Goto

意義： 有條件跳躍至標籤。判斷條件不是兩個 ISaGRAF 變數的比較，就是變數與常數表示式的比較。

語法： **If** <變數 1> **test** <變數 2> **Goto** <標籤名稱>

If <變數 1> **test** <常數表示式> **Goto** <標籤名稱>

可使用的比較測試 (test) 指令有：

使用者指南

= 假如兩邊相等，則結果為真
<> 假如兩邊不相等，則結果為真
< 假如第一個小於第二個，則結果為真
<= 假如第一個小於或等於第二個，則結果為真
> 假如第一個大於第二個，則結果為真
>= 假如第一個大於或等於第二個，則結果為真

引數： <變數 1> <變數 2> 為宣告的變數名稱，或是使用“%”語法的變數直接表示法。

<常數表示式> 是與變數相同型態的有效常數表示式。對布林型態而言，“0”及“1”可用來代替“FALSE”或“TRUE”。對計時器型態而言，前置詞“T#”或“TIME#”可以省略。

<標籤名稱> 是草稿中所定義的標籤名稱。

注意： 往前跳躍是被允許的。假如為無窮迴圈，草稿每執行一次迴圈會自動產生中斷，如此便可讓 ISaGRAF 週期持續下去。

範例： (* MyVar 為真時，這個草稿才會結束 *)

Loop:

If MyVar = TRUE Goto TheEnd

Print MyVar

Goto LOOP

TheEnd:

End

意義： 結束草稿

語法： **End**

注意： “End”指令並不強制必須置放於草稿最後一列

範例： (* MyVar 為真時，這個草稿才會結束 *)

Loop:

If MyVar = FALSE Goto Continue

End

Continue:

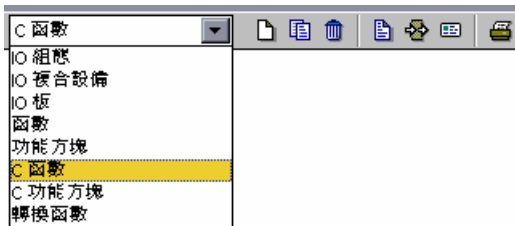
Print MyVar

Goto Loop

A.22 使用程式庫管理員

ISaGRAF 程式庫 (library) 提供一個介於自動化發展和 ISaGRAF 目標硬體系統軟硬體能力的標準界面。不同型式的界面有不同的資料庫。ISaGRAF 工作平台程式庫管理員設計給硬體供應者或者是軟體工程師來使用，他們使用程式庫管理員來描述他們所創造的 ISaGRAF 程式設計界面物件。

ISaGRAF 工作平台程式庫管理員顯示出 ISaGRAF 程式庫中的元件。視窗左邊區域是所選資料庫的**元件明細表**，視窗右邊區域是在元件表上所選元件的**技術註記** (使用者手冊)。程式庫管理員清單包含用來開啓、定義或修改程式庫元件的命令。“**檔案 / 其它程式庫**” 命令允許選取一個 ISaGRAF 程式庫。在工具列左邊上的組合窗也能夠被用來選取一個程式庫：



A.22.1 管理程式庫元件

使用“**檔案**”清單命令來開啓元件以及編輯開放程式庫中既有元件。

開啓一新元件

“**檔案**”功能表中的“**開新元件**”命令於所選程式庫中產生一個新的元件。

新元件名稱必須根據下列命名規則：

名稱最大長度為 **8** 個字元

第一個字元必須為**英文字母**

之後的字元必須為**英文字母**、**數字**或是‘**_**’字元

程式庫元件名稱不分大小寫

註解將被結合到每一個程式庫元件中。當產生新元件時，可以輸入它的註解。當新元件被創造時，下列必須被輸入：

I/O 組態的定義

I/O 板的參數

函數或功能方塊的使用者界面

當“C”轉換、“C”函數或“C”功能方塊被創造時，其原始碼將自動產生。

⇒ 編輯已存在的元件

“**檔案 / 重新命名**”命令准許使用者從元件明細表中改變所選的元件名稱或註解。“**檔案 / 複製**”命令允許使用者複製程式庫中的元件到同一程式庫的另一元件。假如目標元件已經存在，它的所有內容將會被覆蓋掉。假如目標元件並不存在，它會自動被創造。“**檔案 / 刪除**”命令從動作程式庫中去移除所選的元件。“**更名**”，“**複製**”和“**刪除**”命令將會作用在元件的下列部分：

技術註記

I/O 組態的完整定義

I/O 板或複合設備的參數

函數或功能方塊的界面定義

以 IEC 語言寫成函數或功能方塊的原始碼

C 轉換、函數或功能方塊的原始碼

⚠ 假如元件是“C”轉換、“C”函數或者“C”功能方塊，“**更名**”或“**複製**”命令並不會自動更改它的名稱。

⚠ 假如元件是一個由 IEC 語言所寫成的函數，那麼回傳參數名稱無法由“**更名**”或“**複製**”命令所更改。

⇒ 設定密碼保護

“**檔案 / 設定密碼**”命令讓使用者能夠對開放程式庫中所選的元件設定密碼保護。查閱“**密碼保護**”一節（在手冊第一部的最後），有更多有關密碼等級

使用者指南

和資料保護的資訊。密碼只和所選元件有關，它們不會影響其他 ISaGRAF 程式庫元件。

☐ 編譯函數和功能方塊

當由 IEC 語言所寫成的函數或功能方塊程式庫被選取時，“**檔案**”功能表中的“**驗證 (編譯)**”命令被用來檢查所選元件的語法及產生它的物件碼。在由 IEC 語言所寫成的函數和方塊可以被用在 ISaGRAF 專案之前，它們必須被編譯到沒有錯誤。假如另一個程式庫被選取的話，這項命令就沒有效果。

技術註記

“**編輯 / 技術註記**”命令允許使用者從目前程式庫所選取的元件中輸入技術註記。技術註記是從 ISaGRAF 文字編輯器輸入的。一個元件的技術註記就是它的**使用者指南**。ISaGRAF 專案在整合時，元件的使用者將可以查閱技術註記。技術註記在有關如何使用元件方面應該包括其本身主功能描述、程式設計界面和參數的詳細說明以及它自身內容和限制。

“**工具 / 標準註記格式**”命令准許使用者對現行所選程式庫的所有元件去定義一個標準文字格式。當對新元件編輯技術註記時，這格式就被當作是主要架構，且允許使用者去作最佳化技術註記編輯。

參數

元件的參數即是元件所提供的電腦操作和 ISaGRAF 應用程式元件的使用之間的**界面**。參數對每一程式庫元件的型態有著不同的意義。I/O 組態的參數定義組態的 I/O 板的完整集合，和用於 I/O 連結點的預設變數名稱。I/O 板或複合設備的參數定義這塊板的物理和邏輯組態。函數或功能方塊的參數是根據 ST 語言函數呼叫協定來定義元件的界面。轉換函數是沒有參數的，因為它使用到一套預先定義的標準界面。

原始碼

ISaGRAF 工作平台准許程式設計師去管理程式庫轉換、函數或功能方塊的原始碼。一個由 IEC 語言所寫成的函數或功能方塊是一串文字或是圖形。“C”

組成部份 (“C” 函數、“C” 功能方塊和轉換函數) 的原始碼被分離成兩個分割檔案：一個 **C 原始標頭檔** (包含界面的正確定義、成員的參數定義)，以及一個 **C 原始碼檔** (包含元件的執行程式)。

當新資料庫元件建立時，ISaGRAF 工作平台就會產生原始碼檔。基於參數的定義，它也不同建立和更新原始標頭檔。程式設計師可以使用 ISaGRAF 文字編輯器來完成原始碼檔。

☐ 拷貝程式庫元件

The “工具 / 拷貝工具” 功能表的命令會執行 ISaGRAF 拷貝管理員來儲存或回存程式庫元件。在執行 “拷貝命令” 之前，你第一步必須選擇一個程式庫。拷貝管理員一次只會顯示一個程式庫的元件集。

A.22.2 I/O 組態

ISaGRAF I/O 組態程式庫提供一個簡單的方式來初始化有預先定義 I/O 組態的新 ISaGRAF 案件。I/O 組態定義著：

I/O 板集合

預設的 I/O 板參數

預設的 I/O 連結點名稱

當新的 ISaGRAF 案件和程式庫 I/O 組態一同被建立時，相關的 I/O 連結將被自動宣告在專案字典內。

 使用 ISaGRAF I/O 連結工具來製作 I/O 組態定義 (案件內有著相同工具)。在手冊中的 “I/O 連結” 一節可得到更多有關如何去使用這項工具的資訊。當在組態設定插入新的 I/O 板時，這塊板的所有連結點被宣告成標準預設名稱。I/O 連結點的標準預設名稱的格式如下：

<方向><型態><槽位>_<連結點編號>

第一個字元指示 I/O 連結點的方向：

使用者指南

“I” 輸入連結點

“O” 輸出連結點

第二個字元指示 I/O 連結點的型態：

“X” 布林

“D” 類比

“M” 訊息

下面是一個標準預設名稱的一些範例：

IX0_7 布林輸入 - 板#0 - 連結點#7

OD2_4 類比輸出 - 板#2 - 連結點#4

I/O 連結編輯器中的“連結 I/O 連結點”命令被用來修改 I/O 連結點的預設名稱。

A.22.3 I/O 複合設備

一塊單板的所有連結點有著相同型態（布林，類比或訊息）和方向（輸入或輸出）。一個複合 I/O 設備表示為一個有不同型態或方向連結點的 I/O 設備。一個複合 I/O 裝置被表示成一串列的 I/O 單板，但僅用到一個插槽。



欲定義一個複合 I/O 設備，使用者必須定義所有 I/O 設備所用到的單板，也必須輸入每一單板的詳細參數。可以經由對話框來輸入所有的 I/O 單板。

按下“**新增**”按鈕允許使用者去增加新的單板到現行表列上。“**插入**”按鈕用來插入新的單板在所點選表列項目之前。“**刪除**”按鈕從表列中移除所點選的單板。“**更名**”和“**參數**”按鈕用來改變所點選單板的名稱和參數。下一節將對單板參數提出一個完整的解釋。一個複合 I/O 裝置最多由 **16** 塊單板組成。單板（於 I/O 裝置內）名稱不能超過 **8** 個字元。

A.22.4 I/O 板

ISaGRAF I/O 板程式庫定義應用程式和目標硬體之間的標準界面。在應用程式描述時，所有 I/O 變數將被連結到目標硬體的 I/O 板的連結點。ISaGRAF I/O 板由名稱及“**OEM 識別碼**”（用來識別硬體供應商）所定義。其它 I/O 板參數描述 I/O 板性質（連結點數目、連結點方向和型態），以及它自身的軟硬體組態。

I/O 板參數

I/O 板具有兩個不同的參數型態：共同參數—對 ISaGRAF 程式庫板作定義，及 OEM 參數—由硬體供應商提供，用來指定板子的執行程式。共同參數被輸入到 I/O 參數定義盒上方處。這些參數（加上 I/O 板名稱）用來識別 ISaGRAF 標準 I/O 板界面。

“**OEM 識別碼**”是一串簡單的數字，用來定義是那個**硬體供應商**。由同一硬體供應商定義的所有板子必須有相同 OEM 識別碼。OEM 識別碼是一個 **16 位元長度無正負號字組**，且為一個十六進位格式輸入值。保留給 ICS Triplex ISaGRAF Inc.的 OEM 識別碼為“**1**”。

主要參數定義 I/O 板性質。連結點數定義板子的**連結點的數目**。板子的**型態**就是能被連結到板子連結點的變數型態。**方向**定義連接到板子的變數是**輸入型態**或是**輸出型態**。

注意： 不同型態或方向的 I/O 變數不能聚集在同一 ISaGRAF I/O 板。這項性質只適用在複合 I/O 設備上。

□ OEM 參數

I/O 板參數定義盒的下方處可用來輸入 OEM 參數。這些參數由 I/O 板的硬體供應商所定義。一個 I/O 板最多可以有 **16** OEM 參數，但也可能沒有 OEM 參數。ISaGRAF 程式庫管理員允許硬體供應商定義每一參數的識別和格式，以及自動化程式設計師的輸入方式。

使用者指南

方塊左方處包含 OEM 參數表列。每一參數由一個**名稱**和一個從 **0** 到 **15** 的邏輯**數字**所組成。右方區域包含表列上所選參數的細節描述。爲了輸入參數的完整描述，你必須從表列中選取它。按下“**清除**”按鈕將重設參數描述，以及從參數表列移除它。**警告**：這項命令不能被“回復”。

於 I/O 連結時，假如這個欄位必須由自動化操作員所定義，必須用參數名稱來作輸入欄位的識別。參數名稱必須遵守下列規則：

名稱的最大長度爲 **16** 個字元

第一個字元必須爲**英文字母**

其餘的字元必須是**英文字母**，**數字**或‘**_**’字元

參數型態定義參數的**內部格式**，以及在應用程式 I/O 連結時的**輸入格式**。下面是可用的內部格式表列：

word 無正負號 16 位元字組

long 無正負號 32 位元字組

word hexa 無正負號 16 位元字組

long hexa 無正負號 32 位元字組

boolean 無正負號 16 位元字組 (只用到最低位元)

character 無正負號 16 位元字組 (只用到最低位元組)

string 包含空結束字元的 16 個字元陣列

float 單精度 32 位元浮點數值

下面是可用的輸入格式：

word 無正負號十進位字組 (word)

long 十進位長字組 (long word)

word hexa 無正負號十六進位字組

long hexa 無正負號十六進位長字組

boolean “真”或“假”

character 單一字元

string ascii 字串 (最大 15 個字元)

float 單精度浮點數值

“存取”方塊被用來定義參數如何被終端使用者 (end user) 所存取。假如“使用者定義”選項被設定，參數在 I/O 板連結時將被顯示成一可輸入的欄位。OEM 參數預設值被用作參數編輯的一項預設。假如“隱藏”選項被設定，參數將為常數值，而且不會出現在 I/O 板連結方塊上。OEM 參數預設值定義常數參數值。“唯讀”選項指示參數對使用者而言是可見的，但是不能被修改，它的內定值被用來當作一個常數值。

A.22.5 用 IEC 語言寫成的函數和方塊

ISaGRAF 可管理一個用 IEC 語言寫成的函數和方塊程式庫。用來描述如此一個函數或方塊的語言有 **FBD** (功能方塊圖)、**LD** (階梯圖)、**ST** (結構化文字) 或 **IL** (指令集)。注意 LD 和 FBD 語言能夠被用於同一個圖形中。在程式庫中 **SFC** 語言 (順序式功能圖) 不能被用來描述函數或方塊。當函數被開啓時會要求選取附加在程式庫元件的語言，而且往後都不能改變成其他語言。

一 編譯

在它們能被用於 ISaGRAF 案件之前，定義在程式庫中的函數和方塊必須被編譯 (檢驗)。在程式庫中有關函數和方塊的部分不需要改變。當在案件中使用 LD/FBD 圖形編輯器的時候，資料庫元件將自動出現在方盒中的選取清單中。

⚠ 定義在程式庫中的函數能呼叫其它在這程式庫的函數，然而，ISaGRAF 系統並不支援遞迴的函數呼叫。用 IEC 語言寫成的功能方塊不能呼叫其它功能方塊 (不論是 IEC 語言或是 “C” 語言)。

輸入原始碼

利用標準 ISaGRAF 工具：LD 或 FBD 程式的圖形編輯器、ST 或 IL 程式的文字編輯器來輸入程式庫函數或功能方塊的原始碼。可參考手冊中相關的章節以得到這些工具的更進一步的資訊。ISaGRAF 程式碼產生器能夠從圖形或文字編輯視窗中直接被呼叫來編譯程式庫函數或功能方塊的原始碼。

二 區域變數字典

使用者指南

程式庫函數或功能方塊能夠造區域變數，以及區域定義字。當編輯函數原始碼時，想要在編輯器視窗存取變數宣告，使用者必須執行“**檔案**”功能表中的“**字典**”命令。

 程式庫函數或功能方塊不能夠存取全域變數或功能方塊樣例。在函數主體部分，函數的區域變數應被初始化。

在案件中每一次使用方塊時會複製以 IEC 語言寫成的功能方塊的區域變數(樣例)。當從一個呼叫到另外一個時，樣例的區域變數會保持它們的值。

定義界面

函數或功能方塊最多可以有 **32** 個參數 (輸入或輸出)。爲了遵守 ST 語言寫法協定，函數總有一個 (而且只有一個) 回傳參數，且必須和函數名稱相同。

在視窗左上方表列內顯示這些參數，呼叫模組次序爲：首先爲呼叫參數，最後爲回傳參數。在視窗下方處顯示在表列中目前選取參數的細節描述。參數可以是任何一種 ISaGRAF 的資料型態。回傳值參數必須放在表列的所有呼叫參數之後。參數的命名方式必須遵循下列規則：

名稱長度不能超過 **16** 個字元

第一個字元必須爲英文字母

其餘的字元必須爲英文字母、阿拉伯數字或底線字元

名稱不分大小寫

“**插入**”命令用來插入新參數到所選參數之前。“**刪除**”命令用來刪除所選取的參數。“**排列**”命令自動重排 (排序) 參數，所以回傳參數將被放置在表列的最後。

A.22.6 “C”函數和功能方塊

根據 ST 語言函數呼叫界面，“C”函數和功能方塊是一種**電腦函數**，可被自動化應用程式所呼叫。

函數是**同步**的程序。在函數執行期間會暫停 ISaGRAF 目標端應用程式。功能方塊結合指令和靜態隱藏資料，舉例來說，“計數器”功能方塊代表計數指令以及計數結果。函數和功能方塊可用來讓標準自動化語言性能更加圓滿，也可以存取系統資源。



參數定義方塊用來定義每一個函數或功能方塊的呼叫或回傳參數。“編輯”功能表命令用來定義所選函數或功能方塊參數。一個函式最多 **31** 個呼叫參數，而且總是有一個傳回值參數。功能方塊最多 **32** 個參數，包含任何固定的呼叫和回傳參數。下面是 ISaGRAF 型態和 “C” 型態之間的對照：

布林	unsigned long	無正負號 32 位元字組：1=是 / 0=假
類比	long	帶正負號 32 位元字組
實數	float	單精度浮點數值
計時器	unsigned long	無正負號整數 32 位元字組 (單位為 1ms)
訊息	char *	字元字串

當從 “C” 函式或功能方塊傳出一個訊息值，不能包含空字元。字串傳給 “C” 程式碼的結束字元為空字元。

參考 ISaGRAF 目標端使用者指南有更多資訊，關於如何處理函數或功能方塊的 “C” 原始碼，以及如何去整合新元件到 ISaGRAF 目標端系統。

A.22.7 轉換函數

轉換函數是一個每次從專案中使用到輸出入轉換的類比變數，由 ISaGRAF I/O 管理員呼叫的 “C” 函數。

函數創造變數的**電位值** (從輸入感應器讀取資料) 和**物理值** (用於應用程式表示式中) 關係。函數因此分成兩個部分：輸入轉換以及輸出轉換。ISaGRAF 程式庫管理員允許使用者控制轉換函數的 “C” 原始碼。

使用者指南

轉換可用來處理**整數**或**實數**類比變數，它暗指轉換函數界面總是由浮點變數定義。對任何轉換函數而言界面都是相同的。界面的“C”定義是由“TACN0DEF.H”定義檔所定義的。

參考 ISaGRAF 目標硬體使用者指南有更多資訊關於如何管理轉換函數的“C”原始碼，以及如何整合一新成員到 ISaGRAF 目標端系統。

A.23 使用檔案拷貝工具

ISaGRAF 檔案拷貝工具 (archive) 讓使用者能夠儲存 ISaGRAF 案件和程式庫到磁碟片或備份目錄上。ISaGRAF 檔案拷貝管理員的對話盒可以從 ISaGRAF 案件管理員或程式庫管理員視窗中呼叫出來。

 爲了創造和維護可靠的檔案，建議使用下述的作法：

- 寫下儲存物件名稱及描述到磁碟貼紙上
- 不要儲存案件或程式庫到相同磁碟
- 不要儲存不同案件到相同磁碟

A.23.1 呼叫檔案拷貝管理員

“檔案”功能表的命令對一種物件型態執行備份/回存對話框。包含有案件、程式庫元件和共同資料。你可以從案件管理員視窗的“工具 / 拷貝”功能表呼叫“拷貝”對話盒，然後儲存或回存一個案件或共同資料。

你也可以從 ISaGRAF 程式庫管理員的“工具 / 拷貝工具”命令來執行“拷貝”對話盒，然後儲存或回存目前程式庫管理員視窗中所選取的程式庫元件。

▣ 案件

案件總是儲存所有的內容，案件的所有組成部分（程式原始檔、目的碼、應用程式可執行碼）一起儲存到相同檔案中。選取“**壓縮**”選項將減少案件檔案的大小。

▣ 程式庫元件

ISaGRAF 程式庫元件可以各別儲存。程式庫元件的所有組成部分（技術註解、定義、界面、原始碼...）一起儲存到相同檔案中。

□ 共同資料

案件管理員視窗的“**工具 / 拷貝 / 共同資料**”命令使使用者能夠備份或回存 ISaGRAF 工作平台的“共同範圍”資料，這項命令不作用於 ISaGRAF 程式庫。下列是可用這項命令複製之檔案的表列：

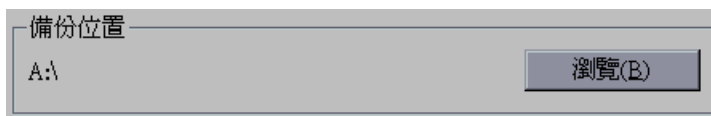
common.eqv 共同的定義字

oem.bat 使用者自訂的 MS-DOS 指令檔

這些檔案會以它們原來的形式一個一個地儲存在作用磁碟中，且不會加以壓縮。

A.23.2 選項

ISaGRAF 備份檔案的路徑會顯示在對話盒的底部。按下“**瀏覽**”按鈕可以瀏覽磁碟機並且選取另一個備份檔案的磁碟機及目錄。



當“**壓縮**”選項設定時，所有**備份**的檔案會使用壓縮的程序。這項選項對於縮減大型案件檔案的大小是非常有用的，而且只須一片磁碟片便可儲存。檔案壓縮一般來說並沒有用到程式庫的元件。當回存檔案時，ISaGRAF 檔案管理員會自動辨識此程式檔案的狀態（壓縮或不壓縮），這意喻著對“**回存**”程序而言，“**壓縮**”指令是沒有作用的。

☐ 壓縮

A.23.3 備份和回存

“**工作目錄**”表列的左邊顯示安裝在硬碟的 ISaGRAF 工作平台所存在的物件。“**備份目錄**”表列顯示儲存在指定的備份磁碟機和目錄的物件。

☐ 備份

選取**左邊**表列內的物件且按下“**備份**”按鈕 (ISaGRAF 工作平台的物件) 來儲存物件到檔案室。表列中可以選取超過一個的物件。當從**右邊**表列 (回存模式) 選取一物件時，“**備份**”按鈕將不能夠被壓下。

☐ 回存

選取**右邊**表列上的物件 (檔案室物件) 和按下“**回存**”按鈕，從備份目錄複製一物件到 ISaGRAF 工作平台。表列中可以選取超過一個的物件。當從**左邊**表列 (備份模式) 選取一物件時，“**回存**”按鈕將不能夠被壓下。

A.23.4 備份檔

ISaGRAF 檔案管理員會對每一個被儲存的物件建立一個獨立的作用檔，作用檔會與物件具有相同的名稱，而作用檔的附檔名標示出它的型態。下列是可用的附檔名：

- .pia 案件
- .bia I/O 板
- .iia IEC 語言函數
- .aia IEC 語言功能方塊
- .uia C 函數
- .fia C 功能方塊
- .cia C 轉換函數
- .ria I/O 組態
- .xia I/O 設備

A.24 列印完整文件

T ISaGRAF 文件產生器准許使用者對所選物件去建立和列印一份完整的文件。你能從案件管理員的“**案件 / 列印**”命令或程式視窗執行文件產生器視窗來列印一份完整的文件。文件產生器也能由其他 ISaGRAF 編輯器的“**列印**”命令來執行，列印一個 ISaGRAF 文件的內容。然而，不論是用哪一種方式執行文件產生器都具有相同的功能。

“**編輯**”功能表的命令用來定義必須插入在這一文件內的案件成員。爲了能作到這件事，使用者要建立起這份文件的“內容表”。任何關於此案件的資訊（程式、變數、指令、I/O 連結）都可放入這案件文件內。但是在此文件並不包含其它案件資訊或 ISaGRAF 程式庫。

 “**檔案 / 列印**”命令會根據所指定的內容表產生這份文件，而且把它送到印表機。“**列印**”的工作會花費一些時間去建立和製作這份文件的格式。在 ISaGRAF 文件產生器視窗中的“**列印工作**”完成前，在 ISaGRAF 工作平台想再下其他命令時，最好是稍微等待一下。建立整份文件可能須要硬碟的一塊大空間，假若磁碟已滿，將會出現錯誤訊息，像這樣的狀況，使用者必須刪除檔案或縮減列表的內容來增加磁碟空間。當“**列印**”命令執行時，將會出現一對話方框。它允許使用者輸入一串註解用來描述這項實際列印命令。那些註解存放在一個歷史檔中，而且將會列印到任何更新文件的第一頁（包含現在這個）。

A.24.1 自訂內容表

“**編輯**”功能表包含一些命令，用來定義這份文件的“內容表”。這些命令允許使用者去使用預設表格（包含案件中的所有元件）、去建立特定的表格（只包含一些元件）、或是去調整表列內的項目以及去修改它。

預設表列

“**編輯**”功能表中的“**預設表列**”命令定義了標準的文件內容表，包含所有案件元件。標準表組成如下：

案件描述

架構樹 (程式間的連結)

程式原始碼

程式日記

共同定義

全域定義

程式區域定義

全域變數

程式區域變數

應用程式選項

I/O 連結

變數表列

轉換表

交互參考摘要

交互參考詳述

宣告總成

網路位址圖

變更歷史

可以使用“**檔案 / 儲存**”命令將內容表儲存到磁碟中。當你從一個 ISaGRAF 編輯器執行文件產生器來列印一份單一文件時，此命令會標示成灰色。



剪下和貼上

自訂表列的次序時，使用“**編輯 / 剪下**”和“**編輯 / 貼上**”命令來移動表列內的項目。文件產生器允許一次選擇多個項目，可以同時將它們同時剪下及貼上。



清除表格

使用者指南

利用“**編輯 / 清除**”命令重設內容表，使插入單一項目時內容表可以被重建。



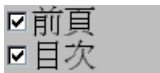
插入項目至表格

當“**編輯 / 插入**”命令執行時，下列對話方框將會出現，允許使用者插入項目 (案件元件) 至內容表。

對於項目如何關連到程式，可利用“**程式**”選取盒來選取程式名稱。按下“**加入**”按鈕來插入所選項目到內容表。在表內，相同項目只可出現一次。

A.24.2 選項

“**選項**”功能表的命令用來定義和自訂一般文件格式。你也可以用文件產生器視窗上的按鈕來直接執行其他的選項：



當設定了“**前頁**”選項，列印的文件開頭將產生一頁開頭頁，此頁包含案件標頭及列印紀錄。當此選項未設定，第一個列印項目會從第一頁開始列印。

當設定了“**目次**”選項，目次表將列印於產生的文件之後。

當從 ISaGRAF 編輯器 (程式、字典..) 下執行“**列印**”命令，開啓文件產生器時，這兩個選項初始值為未被選取的狀態。

☐ SFC 圖

“**分離 SFC 層**”選項用來指導系統去列印各個 SFC 程式，首先是 SFC 第一層 (圖及註解)，然後是第二層的程式內容。當此項選項未設定時，第一層和第二層會一同出現到列印輸出端。



單頁模式

當製作頁面格式時，“選項”功能表中的“頁面格式”用來定義文件產生器所操作的主要參數。下列參數可以被指定：

左邊界：(1 或 2 公分，或無留邊)

頁面邊框：當選取這項選項時，會在所有列印頁四周畫上邊框。

頁面標題樣本

“選項”功能表中的“頁面標題”命令用來定義列印頁底的標題方塊內容。標題方塊的標準輸出如同下列所示：

	Text1	ISaGRAF - Project 'PrName'	date
	Text2		
	Text3	User defined title	page

在主標題的第一行 (內容為 ISaGRAF 案件的名稱)，案件的內容和頁數將自動由文件管理員產生，且無法更改。

在框框左邊的三行文字及主標題的第二行，是給使用者自行定義的。使用者也能更改印在框框左邊的圖案。要使用另一個圖案，使用者須詳述點陣圖檔 (.BMP) 的路徑名稱，圖檔可以是任何的尺寸。根據列印頁的正確尺寸，也可以作伸展或收縮。點取在對話框的圖形區域，會顯示出新的指定影像。當**列印**命令執行時，影像檔必須存在於磁碟上 (在指定目錄和指定檔名)。



選取文字字型

當列印文件項目的內文和標題時，“選項”功能表中的“內文字型”和“標題字型”命令用來定義文字字型。對於內文和標題文字的大小及表示方式也可以加以選擇。字型的選取取決於視窗所定義的標準對話框。任何內文 (在對話框中命名的文字程式) 將根據所選大小、表示方式和文字字型來列印，只有標題將根據所選標題文字字型來列印。

假若沒有定義文字字型，內文中將根據印表機上的標準字型和下列表示方式列印：

對於內文和對話框名稱，使用“標準體”表示方式

使用者指南

對於標題，使用“粗體”表示方式

A.25 密碼保護

ISaGRAF 工作平台包含完整資料保護系統，使用者能夠使用密碼來保護案件和程式庫元件。程式庫元件可以是 I/O 組態、I/O 板或複合設備、函數或由 IEC 語言所寫成的功能方塊、“C” 函數、功能方塊或轉換函數。一個密碼保護資料庫用於一個案件或程式庫元件，而且不能被其他案件或程式庫分享。

▣ 保護層級

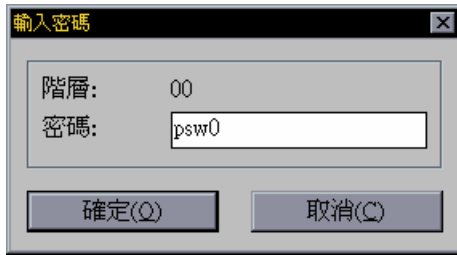
在案件或程式庫元件中，根據不同的密碼使用者可以定義到最多 **16** 個存取層級，存取層級以結構樹方式來儲存，且被標號從 **0** 到 **15**。較高存取層級為 **0**。當使用者知道一道密碼，他可以存取這存取層級的所有項目，再加上在這存取層級以下所保護的項目。每一案件或程式庫元件的初步命令或資料可以分別由不同的存取層級來保護，舉例來說，在 ISaGRAF 功能表中的“製作應用程式碼”命令可以被個別保護。初步資料可以是程式、指令、程式庫元件的技術註記等等。

▣ 定義密碼保護

ISaGRAF 功能表中的“**設定密碼**”命令用來定義案件或程式庫元件的密碼和存取層級，可以從 ISaGRAF 案件管理員 (對於案件) 或 ISaGRAF 程式庫管理員 (對於程式庫元件) 的功能表中來呼叫這個命令。當第一次執行這項命令時，不需要輸入任何密碼。在存取這項命令之前，假如密碼已經定義過，使用者必須輸入所知道的最高層級密碼，較上層級密碼和所保護項目就不能作修改。“**設定密碼**”命令使使用者能夠根據不同存取層級來定義密碼和根據定義的層級保護初步命令或資料。

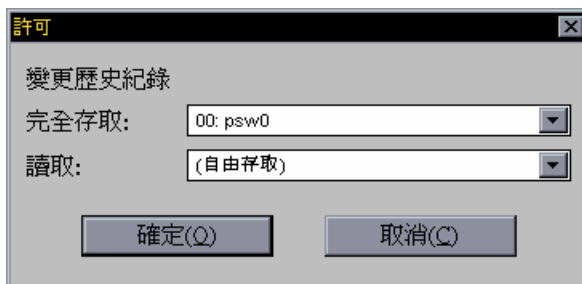
你可以在上方列表中的一行上雙按滑鼠左鍵來輸入密碼 (相對於保護的階層)。下面的對話框可以用來輸入密碼。

使用者指南



視窗下方的列表中，則列出可以被輸入密碼保護的各種不同的物件（資料或函數），及它們目前是屬於“唯讀”或“完全存取”權限的保護層級。指定保護層級給“唯讀”權限可以讓你避免沒有足夠權限的使用者開啓或列印一份文件。

在下方表列的一行上雙按滑鼠左鍵可以對選取的項目或資料設定權限。開啓如下的對話框：



上述兩種權限不是設為“自由存取”，就是設為保護層級中定義的密碼。完全存取權所依附的密碼層級不能低於讀取權所依附的密碼層級。

注意：當使用 ISaGRAF 工作平台時，某些文件（如案件描述）無法設定密碼將其設為唯讀模式，因此此文件將是可見的。

取得被保護的資料

當工作平台被開啓時，並不會被要求輸入密碼或使用名稱。在每一次使用者想要去取得一項被保護的資料或功能時，則使用者必須在對話框中輸入符合的密碼。

如果使用者輸入符合的密碼 (或是輸入連結到較高的存取階層的密碼)，則使用者可以正常的使用下去。每次使用者輸入的密碼都會被儲存於記憶體中，因此使用者一旦輸入正確的密碼後，便不必再一次輸入密碼。在每一次 ISaGRAF 工具執行另一 ISaGRAF 工具時 (例如，案件管理員執行程式管理員)，被儲存的密碼還是會被保持下來。但當最後的 ISaGRAF 視窗被關閉時，被儲存的密碼將會遺失。當案件編輯，或是使用程式庫管理員，或是使用檔案管理員時，其輸入的密碼是不能被分享的。如果使用者輸入一錯誤的密碼，他將不能使用他想使用的功能。

⇒ 連結檔案管理員

當儲存物件 (案件或程式庫元素) 到指定儲存檔案的磁碟中，將能利用名稱爲“**備份檔案**”的資料保護物件，也就相當於資料保護系統連結到工作平台中 (硬碟) 的物件一樣。如果指定儲存檔案的磁碟中，資料保護系統的物件已經存在的話，則將不會有任何的測試動作執行。在 ISaGRAF 檔案管理員中的“**拷貝**”的指令會將資料保護的資訊和物件一併儲存到指定儲存檔案的磁碟中。

當回存一個已存在於工作平台中 (硬碟) 的物件，將能利用名稱爲“**覆蓋備份檔案**”的資料保護物件，也就相當於資料保護系統連結到工作平台 (硬碟) 中的物件一樣，將不會執行任何的測試動作於指定儲存檔案磁碟中資料保護系統的物件。如果此指令已生效，則回存的資料保護資訊將更新原有已存於硬碟中的舊有資訊。

⇒ 設定變數及 I/O 連結點的個別保護

ISaGRAF 工作平台根據密碼層級提供完整的資料保護系統。變數宣告及 I/O 連結皆能做整體的密碼保護，另外 ISaGRAF 也提供你設定變數及 I/O 連結點個別保護的功能，但前提爲：

- 密碼早已於密碼定義系統中定義好 (使用案件管理員視窗中的“**案件 / 設定密碼**”命令)，所以你可以使用這些保護層級來做個別的保護。
- 你必須使用比整個變數或 I/O 保護較高層級的密碼保護來做個別保護。

當變數或 I/O 連結點做了個別保護後，你可以於字典或 I/O 連結視窗內的變數或 I/O 連結點名稱旁邊看到一個小圖示。

使用者指南

於字典或 I/O 連結視窗中，使用“**編輯**”功能表下的“**設定保護**”及“**移除保護**”命令設定或移除所選的變數或接點，這兩個命令都會要求你輸入已定義好的密碼，然後這個保護層級將依附到所選的變數或接點上。每次你要改變這些變數或連結點，你必須輸入它的保護密碼或更高層級的密碼。

警告: 假如變數或連結點以某一層級的密碼做保護，但這個密碼被移除，且沒有設定更高層級的密碼，除非重新定義足夠層級的密碼，否則這些變數或連結點將不能再做任何改變。

A.26 進階程式技巧

此章節包含了更多有關於 ISaGRAF 工作平台與目標系統的資訊。在閱讀此節之前，建議使用者先熟悉 ISaGRAF 的工具和使用程序。

A.26.1 關於 ISaGRAF 工具的更多資訊

當使用 ISaGRAF 編輯工具時，使用者可按**滑鼠右鍵**來開啓一彈出式視窗，此視窗包含了主要的編輯指令，此選單將開啓於目前滑鼠游標的位置。此功能對執行剪貼指令時，所降低的滑鼠使用量來說是非常有用的。

ISaGRAF 工具提供**多工執行**。雖然相同的工具無法開啓兩次來編輯同一文件，但以相同工具開啓不同視窗，或是以平行處理來編輯不同物件，則是可行的。

其它工具列上的指令圖示按鈕，也可以得到其資訊。在工具列空白的區域雙按滑鼠左鍵，將出現一彈出式選單，顯示工具列的內容。或是將滑鼠游標停留在圖示按鈕上以顯示和圖示相同的文字指令。

A.26.2 鎖住 I/O 點與虛擬 I/O 點

若定義 I/O 板為**虛擬**，處理程序將不連結到實際的 I/O 點上。當 I/O 板定義為虛擬時，ISaGRAF 核心的運作並不會改變，唯一的不同在於輸入值不會去讀取輸入感測器，輸出值不會去更新到輸出設備，在此模式下，可以使用 ISaGRAF 除錯器來變更輸入值。**虛擬**屬性可以使用在任何一種板子上。要使板子有虛擬的屬性，必須在使用應用程式執行碼產生器**前**，定義 I/O 板時修改其屬性。**虛擬**的屬性是一種**靜態**特性，而當應用程式停止或重新啓動時將被儲存。

使用者指南

另一種可能是**鎖住** I/O 變數。它將實際設備與相對應的 ISaGRAF I/O 變數連結關係中斷掉。變數是鎖住或解開是透過除錯器來執行。變數鎖住是一種**動態**處理，而且當程式重新啟動時是不會被記憶起來的。**鎖住**的使用只能一次對一個變數執行。下方是主要 I/O 控制摘要一覽：

虛擬屬性 鎖住屬性

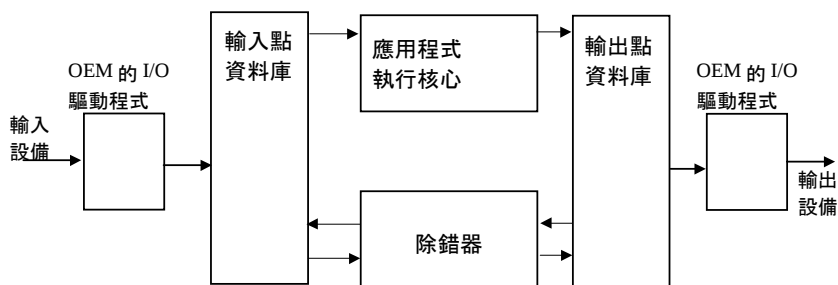
可選取的工具 I/O 板連結 除錯器

定義方式 靜態 動態

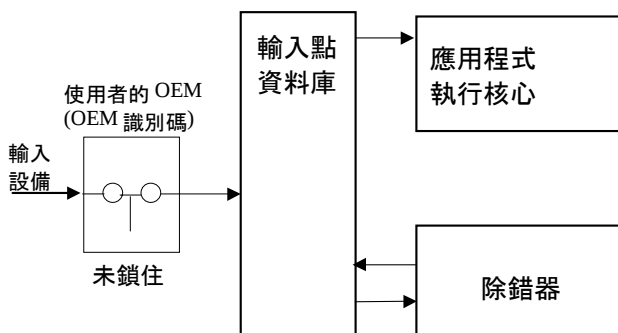
選取的模式 板子 變數

應用程式 確認與測試 保持

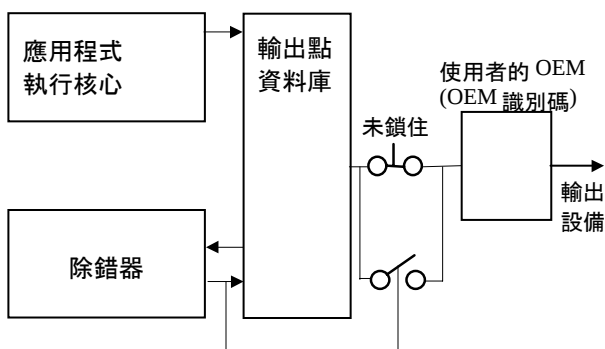
下面的方塊圖將說明 I/O 資料在 ISaGRAF 運作中的流程：



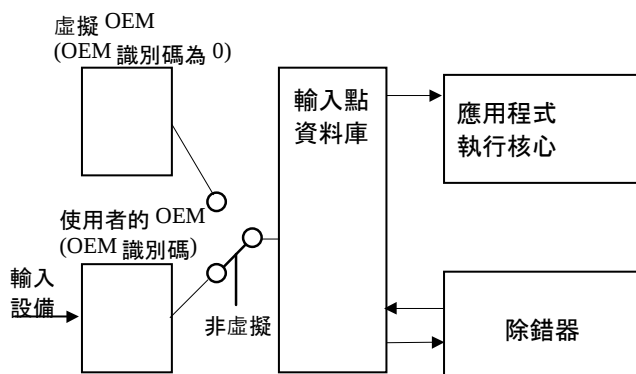
當輸入變數被鎖住時，其它通往資料庫的數個通道並沒有改變，但輸入的設備會不被連結，輸入的值可藉由除錯器更改並由 ISaGRAF 核心處理：



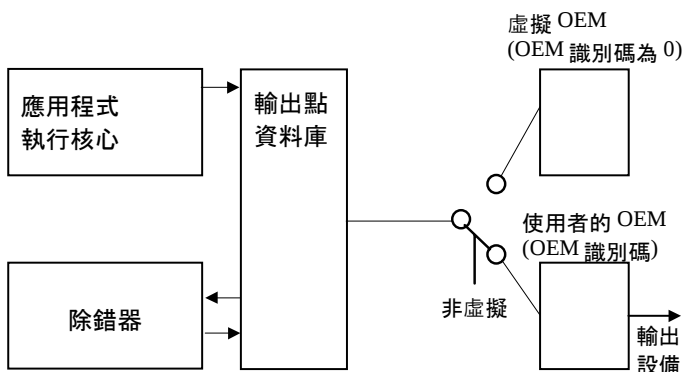
當輸出變數被鎖住時，則執行的核心與輸出驅動程式將失去連結。在這個例子中，使用 ISaGRAF 除錯器，經由輸出驅動程式仍可能對輸出設備作輸出：



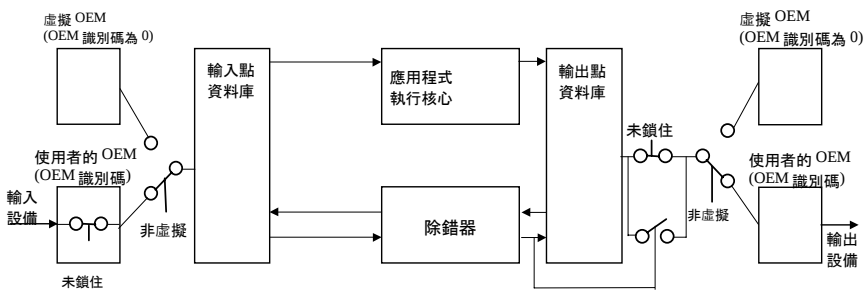
當設定輸入點屬性為虛擬時，則輸入資料庫和連結的輸入設備將失去連結，虛擬 I/O 驅動程式取代實際的。



對輸入板或是輸出板設定成虛擬屬性，其設定方式都是跟據相同的規則。對輸出板而言，ISaGRAF 核心會更新輸出資料庫。而與資料庫連結的輸出設備，不論如何將失去連結。虛擬 I/O 驅動程式將取代實際的。



總結所有的可能性：



A.26.3 PC-PLC 連結

大多數關於 ISaGRAF 工作平台與目標 PLC 間的通訊不良問題，都在除錯器視窗中被顯示成“斷線”的狀態訊息。在任何診斷測試被執行前，當目標 PLC 中沒有應用程式在執行時，其通訊的參數就應該被確認。如此一來串列的通訊連結將可由其自己確認，並能將執行時產生的影響區隔開來。

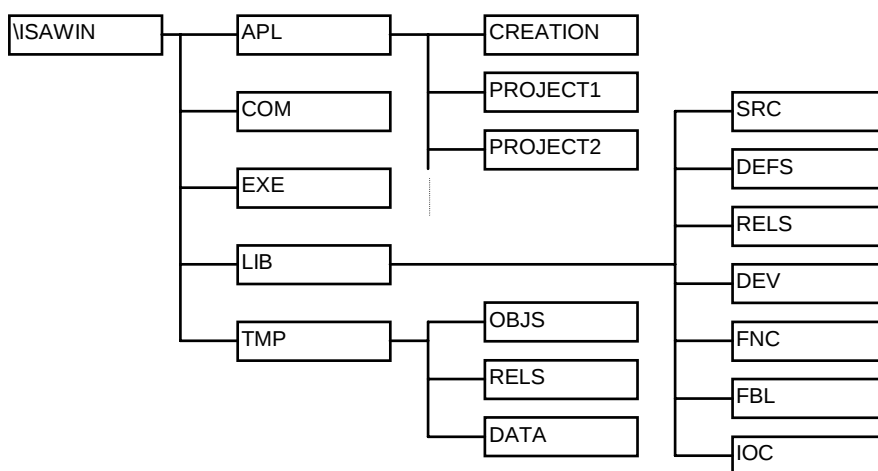
C 語言 (用來撰寫轉換函數與 C 函數的描述) 允許與目標系統直接存取。程式撰寫的錯誤可能會產生系統的錯誤，或導致 ISaGRAF 系統不正常的表現。而當 I/O 的驅動程式是以 ISaGRAF IO 發展工具所開發出來的話，則此類的程式可能會發生。舉例來說，系統錯誤可能是因為 I/O 板連結到不對的 bus 位址而造成的。下面的圖表顯示錯誤診斷的綜合列表：

狀態	情況	診斷
“斷線” (下載之前)		- 目標硬體尚未啟動 - 沒有通訊線\錯誤的通訊線 - 錯誤的通訊參數 - ISaGRAF 目標硬體安裝不正確
“斷線” (下載之後)	以單步 模式啟動	- 錯誤的 I/O 組態 - 系統當機

	以真實 模式啓 動	- 錯誤的 I/O 組態 - 系統當機 (“C” 程式的因素)
“沒有 應用程 式”		- 應用程式未下載 - 應用程式未啓動 (“C” 程式的因素) - Intel/Motorola 選取錯誤 - 錯誤的硬體版本

A.26.4 ISaGRAF 目錄

ISaGRAF 工作平台工作於單一磁碟的目錄結構。此結構的根目錄是使用者在安裝 ISaGRAF 時所定義的，而內定的 ISaGRAF 根目錄名稱爲 **ISAWIN**。下面是由安裝程式所產生出來的標準的目錄結構：



下面是標準 ISaGRAF 下層目錄結構：

路徑	內容
APL	ISaGRAF 案件所在的根目錄
	一個案件就是一個資料夾
	資料夾包含了此案件的所有資料
	你可以將其他的案件群組置放於其他的目錄底

使用者指南

	下。ISaGRAF 安裝時會產生 “SMP” 目錄，此目錄存放應用樣例
COM	“共同” 範圍的資料 能被任何案件使用的資料
EXE	ISaGRAF 的程式和說明檔案
LIB	ISaGRAF 程式庫： - 元件集 - 每個元件的參數或介面 - 技術註記
LIBI	I/O 組態的原始碼
OC	
LIBF	以 IEC 語法所撰寫的函數的原始碼
NC	
LIBF	以 IEC 語法所撰寫的功能方塊的原始碼
BL	
LIBS	轉換函數與 C 函數的原始碼
RC	
LIBD	轉換函數與 C 函數的標頭檔
EFS	
LIBR	轉換函數與 C 函數的目的碼
ELS	
LIBD	發展 “C” 程式庫的指令檔
EV	如 makefiles、link lists，等等.....
TMP	暫存檔：此目錄 TMP 是保留給 ISaGRAF 的執行碼產生器使用，因此不要去刪除它。

上述的各個下層路徑可以被搬移到其它磁碟中的其它位置。當使用者沒有一個標準的目錄結構的話，此時就應於初使檔案 **ISA.ini** 中的 **WS001** 區域的宣告一些下層目錄的參數，而此檔位於 ISaGRAF 下層目錄 **EXE** 中。下面是 **WS001** 區域的輸入項目：

Isa	ISaGRAF 目錄結構的根目錄
IsaEx	ISaGRAF 程式與說明檔案的根目錄

```

e
IsaAp      ISaGRAF 案件的根目錄
I
IsaT      ISaGRAF 暫存檔案的根目錄
mp
IsaSr      給程式庫的原始檔用的目錄
c
IsaD      給程式庫的標頭檔用的目錄
efs

```

請注意，假使你改變 **IsaTmp** 指向另一個目錄的話，則你必須在你所指目錄下增加 **OBJS**、**RELS** 和 **DATA** 等目錄。下面是個更改 **WS001** 區域中所指向的目錄，重新定義標準的 **ISaGRAF** 磁碟目錄結構：

```
;file C:\ISAWIN\EXE\ISA.ini
```

```
[WS001]
```

```
Isa=C:\isawin
```

```
IsaExe=C:\isawin\exe
```

```
IsaApl=C:\isawin\apl
```

```
IsaTmp=C:\isawin\tmp
```

```
IsaSrc=C:\isawin\lib\src
```

```
IsaDefs=C:\isawin\lib\defs
```

當你想加入“C”函數或功能方塊到 **ISaGRAF** 的目標硬體時，則目錄 **ISAWIN\LIB\DEV** 是用來儲存開發檔案如：命令檔、makefiles、maps、等等.....，而目錄 **ISAWIN\LIB\RELS** 則是用來儲存當“C”編譯時所產生的目的檔，**ISaGRAF** 的“C”程式庫也需要此目錄來作一些連結的動作。

A.26.5 應用程式符號

每一個 ISaGRAF 應用程式的物件都是藉由名稱和內部的**虛擬位址**來作為參考的依據，而名稱和位址是經由執行碼產生器所決定的。變數的虛擬位址並不是在宣告此變數時所輸入的**網路位址**。虛擬位址是用在通訊的工作，以及使用 OEM 選項所產生獨特的“C”應用程式上。當執行 ISaGRAF 執行碼產生器時，對所有案件物件（變數、程式、步驟 ...）會把名稱及虛擬位址依法定根據來產生 ASCII 檔案。經由此檔我們可輕易的由使用者的應用程式來知悉有關 ISaGRAF 靜態資料庫的資訊。此檔的名稱為“**APPLI.TST**”，位於 ISaGRAF 案件的目錄中：“\ISAWIN\APL\prname” (prname 是案件的名稱)，本章節有更多有關“**APPLI.TST**”檔案的詳細格式的描述。其中的主要符號則顯示在下方：

VA 虛擬位址

ATTR 變數的屬性

USP “C” 函數

變數可能有的值都顯示在下面。此類的值會出現在“**屬性**”欄位中：

+X 內部變數

+C 唯讀內部變數

+I 輸入變數

+O 輸出變數

除了虛擬位址外，所有的數字都是以十進位的整數來表示。虛擬位址則是以十六進位 4 個數字來表示；且數字前面加上一個字元“**!**”。例如：

123 這是一個十進位數值

!A003 這是一個十六進位數值

下面是檔案“**APPLI.TST**”的主要結構，檔案由一串的**區塊**所組成，一個**區塊**就是一串**記錄**，每個記錄都是以一行文字來表示，每一個**區塊**都是以標頭文字開始。

起始區塊
描述區塊
結束區塊

下面是一個區塊的一般語法：

```
@ <block_name> <arguments>
#record...
#record...
...
```

下面是第一個區塊的結構，包含了應用程式的主要資訊：

```
@ISA_SYMBOLS,<appli_crc>
#NAME,<appli_name>,<version>
#DATE,<creation_date>
#SIZE,G=<nbprg>,S=<nbstep>,T=<nbtra>,L=0,P=<nbpro>,V=<nbvar>
#COMMENT,ICS Triplex ISaGRAF inc.
```

appli_crc 應用程式符號檢查加總
appli_name 應用程式的名稱
version ISaGRAF 工作平台版本號碼
creation_date 應用程式產生的日期
nbprg 程式的數目
nbstep SFC 步驟的數量
nbtra SFC 轉移條件的數量
nbpro “C” 函數使用的數量
nbvar 變數的總數量

使用者指南

結束區塊的結構，用來表示已到了檔案的結尾，如下所示：

@END_SYMBOLS

下面的區塊結構是用來描述應用程式中的程式：

@PROGRAMS,<nbprg>

#<va>,<name>

#...

nbprg 此區塊中定義的程式數量

va 程式中的虛擬位址

name 程式名稱

此區塊的結構是用來描述 SFC 程式中的步驟，如下所示。注意，有一虛擬步驟是定義給每個非 SFC 程式使用的：

@STEPS,<nbsteps>

#<va>,<name>,<father>

#...

nbsteps 此區塊中定義的步驟數量

va 步驟的虛擬位址

name 步驟名稱

father 父程式的虛擬位址

此區塊的結構是用來描述 SFC 程式中的轉移條件，如下所示：

@TRANSITIONS,<nbtrans>

#<va>,<name>,<father>

#...

nbtrans 此區塊中定義的轉移條件數量

va 轉移條件的虛擬位址

name 轉移條件名稱

father 父程式的虛擬位址

此區塊的結構是用來描述應用程式中的布林變數，如下所示：

```
@BOOLEANS,<nb_boo>
#<va>,<name>,<attr>,<program>,<eq_false>,<eq_true>
#...
```

而且假如變數數量超過 4095 時：

```
X#(1.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_true>
```

nb_boo 此區塊中定義的變數數量

va 變數的虛擬位址

vamo 位址範圍 (在布林資料型態中)

name 變數名稱

attr 變數屬性

program 主程式的虛擬位址

或是以 “!0000” 表示全域變數

eq_false 值為假時的表示字串

eq_true 值為真時的表示字串

此區塊的結構是用來描述應用程式中的類比變數，如下所示：

```
@ANALOGS,<nb_ana>
#<va>,<name>,<attr>,<program>,<format>,<unit>
#...
```

而且假如變數數量超過 4095 時：

```
X#(2.<varno>),<name>,<attr>,<program>,<format>,<unit>
```

使用者指南

nb_ana 此區塊中定義的變數數量

va 變數的虛擬位址

vamo 位址範圍 (在類比資料型態中)

name 變數名稱

attr 變數屬性

program 主程式的虛擬位址

或是以 “!0000” 表示全域變數

format= “I” 為整數變數

= “F” 為實數變數

unit 單位的字串

此區塊的結構是用來描述應用程式中的計時器變數，如下所示：

```
@TIMERS,<nb_tmr>
```

```
#<va>,<name>,<attr>,<program>
```

```
#...
```

而且假如變數數量超過 4095 時：

```
X#(3.<varno>),<name>,<attr>,<program>
```

nb_tmr 此區塊中定義的變數數量

va 變數的虛擬位址

vamo 位址範圍 (在計時器資料型態中)

name 變數名稱

attr 變數屬性 (永遠為+X：內部變數)

program 主程式的虛擬位址

或是以 “!0000” 表示全為域變數

此區塊的結構是用來描述應用程式中的字串變數，如下所示：

```
@MESSAGES,<nb_msg>
```

```
#<va>,<name>,<attr>,<program>,<max_len>
#...
```

而且假如變數數量超過 4095 時：

```
X#(4.<varno>),<name>,<attr>,<program>,<max_len>
```

nb_msg 此區塊中定義的變數數量

va 變數的虛擬位址

vamo 位址範圍 (在字串資料型態中)

name 變數名稱

attr 變數屬性

program 主程式的虛擬位址

或是以 “!0000” 表示全域變數

max_len 最大的長度(取決於硬體容量)

此區塊的結構是用來描述應用程式中使用的 “C” 函數，如下所示：

```
@USP,<nb_esp>
#<va>,<name>
#...
```

nb_esp 此區塊中定義的 C 函數數量

va C 函數的虛擬位址

name C 函數名稱

此區塊的結構是用來描述應用程式中使用的 “C” 功能方塊樣例，如下所示：

```
@FBINSTANCES,<nb_fb>
#<va>,<inst_name>,<fb_name>
#...
```

nb_fb 此區塊中定義的 C 功能方塊樣例的數量

使用者指南

va C 功能方塊樣例虛擬位址

inst_name C 功能方塊樣例的名稱

fb_name C 功能方塊的名稱

A.26.6 ISaGRAF “無限點數” (WDL) 版工作平台的 限制

ISaGRAF 工作平台所用到的物件受到一些限制，當然，大多數實際的限制是起因於所用的電腦的組態（可用的記憶體與硬碟空間）以及 ISaGRAF 目標硬體的能力（可用的記憶體與軟硬體資源等 ...）。下面是所受到的最大數目限制：

⇒ 對於案件而言：

物件 最大 備註

程式 255 包含主程式、副程式及子程式

樹狀體系層次 20

安裝於工作平台上的案件數目僅受限於可用的硬碟空間大小。

⇒ 對名稱而言：

名稱： 最大 備註

案件 8 字元

程式 8 字元

變數 16 字元 註解最大 60 個字元

定義字標籤 16 字元

同意定義 255 字元 註解最大 60 個字元

轉換表 16 字元

變數列表 16 字元

函數功能方塊(程式庫) 8 字元 適用於 C 函數、C 功能方塊或以 IEC
語 言撰寫的函數

函數參數 (程式庫) 16 字元 適用於 C 函數、C 功能方塊或以 IEC

語 言撰寫的函數

IO 板 8 字元

IO 組態 8 字元

板子 oem 參數 16 字元

轉換函數 8 字元

⇒ 編輯 (對一個程式而言)：

物件 最大 備註

SFC 列 600

SFC 行 20

SFC 步驟 4095 對全部案件而言，包含步驟、初始步驟、
開始及結束步驟

SFC 轉移條件 4095 對全部應用程式而言

LD/FBD 編輯 200 行

2000 列 亦是編輯區域的大小

Quick LD 編輯 無限制 僅受限於 PC 的容量

IL 標籤 251 於同一個 IL 程式中

文字編輯 40K 位元組 或者根據系統組態來看可能會較少

⇒ 針對字典 (對一個案件而言)：

物件 最大 備註

布林變數 65535

類比變數 65535 包含整數及實數變數

計時器 65535

訊息變數 65535

定義字 4095 相同範圍所用到的

定義字 255 同一個程式所用到的

轉換表 127 於應用程式中所用到的

一個轉換表點數 32 定義於相同轉換表

布林、類比或訊息變數的最大數目限制包含內部、輸入與輸出的變數，亦包含所有編譯器所配置的隱藏的暫時變數。於字典編輯器中的變數總計 (相同

使用者指南

型態、相同範圍) 不可超過 16000 點，但是得視 PC 的組態而定，這個限制有可能小於 16000 點。假如你的目標硬體版本為 3.21 或更早的版本，若同一種型態的變數超過 4095 點，你無法加編譯後的應用程式下載至此目標硬體中。因為標準的 "Modbus" 連結限制的關係，同一種型態的變數也不能設定超過 4095 點的網路位址。

⇒ IO 連結：

物件 最大 備註

IO 板 256 定義於相同應用程式中 (包含 I/O 板及複合設備)

IO 連結點 128 於同一塊板子上

⇒ 對程式庫而言：

物件 最大 備註

函數 (IEC 語言) 255 安裝於程式庫中的數量

功能方塊 (IEC 語言) 255 安裝於程式庫中的數量

C 函數 255 安裝於程式庫中的數量

C 功能方塊 255 安裝於程式庫中的數量

功能方塊樣例 4095 於同一個應用程式中的相同型態的功能方塊

函數輸入參數 31 適用於 C 函數及以 IEC 語言撰寫的函數

功能方塊參數 32 輸出入參數總數量，但至少一個輸出參數

轉換表 128 安裝於程式庫中的數量

IO 組態 255 安裝於程式庫中的數量

IO 板 255 安裝於程式庫中的數量

複合 IO 設備 255 安裝於程式庫中的數量

板子 oem 參數 16

B. 語言參考

B.1 案件架構

ISaGRAF 案件分成數個可以撰寫的單位—稱為**程式**。案件中的程式是以樹狀結構的方式連結在一起。程式可以用任何 **SFC**、**FC (流程圖)**、**FBD**、**LD**、**ST** 或 **IL** 等以繪圖或文字形態的語言來撰寫。

B.1.1 程式

程式就是一個具邏輯性的可程式單位，敘述處理**變數**的流程。程式也能敘述成**流程性**或**週期性**的運作。週期性的程式是在每一次目標系統的循環時間執行一次，流程性的程式則是依 **SFC** 或 **FC** 語言的動態規則來執行。

程式就像樹狀架構般的被連結在一起。被擺在樹狀架構上層的程式是由系統來啟動，而下層的程式（樹狀架構中較下層的程式）則是由其自身所屬的父程式來啟動的。程式可以用下列任何以繪圖或文字形態的語言來撰寫：

循序式功能圖 (SFC) 用於高階的程式撰寫

流程圖 (FC) 用於高階的程式撰寫

功能方塊 (FBD) 用於複雜的週期性的運作

階梯圖 (LD) 只用於布林變數的運作

結構化文字 (ST) 用於任何週期性的運作

指令集 (IL) 用於低階的運作

除了 **FBD** 和 **LD** 的符號可以混合運用在同一個圖表中以外，相同的程式只能以一種語言撰寫。

B.1.2 循序的與週期的運作

程式的樹狀結構被劃分為四個主要**區域**或群組：

開始	每次目標硬體的循環時間起始時，此區程式會被執行
循序	程式依 SFC 或 FC 語言的動態規則來執行
結束	每次目標硬體的循環時間結束時，此區程式會被執行
函數	給所有程式呼叫的副程式集

在“**開始**”區域或“**結束**”區域中的程式是被敘述成週期式的處理，且不受時間的影響。在“**循序**”區的程式則被敘述成順序式的處理，其中計時器變數是明確地同步出現於各基本動作中。“**開始**”區中的主程式在每一次程式執行循環的開始都會被有系統的執行。“**結束**”區中的主程式在每一次程式執行循環的結尾都會被有系統的執行。“**循序**”區的程式則依據 **SFC** 或 **FC** 的規則執行。

“**函數**”區的程式是一種副程式，且在案件中的任何程式都可以呼叫使用它。“**函數**”區的程式可以呼叫此區內的其它程式。

循序區的主程式和子程式必須以 **SFC** 或 **FC** 語言撰寫，週期區域（**開始**和**結束**）的程式則不能以 **SFC** 或 **FC** 語言撰寫。任何區域中的任何程式都可以有一個或多個副程式，循序區的任何程式都可以有一個或多個的 **SFC** 或 **FC** 子程式（根據它所使用的程式語言），副程式不能以 **SFC** 或 **FC** 語言撰寫。

開始區的程式通常用來預先處理一些運作於輸入設備的輸入變數，這類變數會經常的被**循序**區的程式使用。**結束**區的程式通常用於把**循序**區處理過的變數，在其將值輸出到設備前，先做確認的處理。

B.1.3 SFC 及 FC 子程式

任何循序區的 **SFC** 程式皆可以控制其它的 **SFC** 程式，此種階層較低的 **SFC** 程式我們稱為 **SFC 子程式**。**SFC 子程式**是種可以被其父程式啟動、刪除、冰凍或反冰凍的並行處理程式，而父程式與子程式必須以 **SFC** 語言來撰寫。**SFC 子程式**可以使用區域變數及定義字。

語言參考

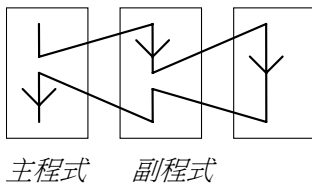
當父程式啟動 **SFC** 子程式時，它將會放置 **SFC** 的權杖記號 (token) (動作) 到 **SFC** 子程式的每個初使步驟中。啟動子程式的指令是用 **GSTART**。當父程式停止 **SFC** 子程式時，它將會清除所有位於 **SFC** 子程式步驟中的權杖記號。停止子程式的指令是用 **GKILL**。

當父程式冰凍 **SFC** 子程式時，它會暫停它的執行。然後可以用 **GRST** 指令讓被暫停的程式重新執行。

循序區中的任何 **FC** 程式可以控制其他的 **FC** 副程式。**FC** 副程式執行期間 **FC** 父程式會被凍結住 (變成等待狀態)。同時讓 **FC** 父程式及它的 **FC** 副程式執行是不可能的。

B.1.4 函數與副程式

副程式或函數的執行與否，是由其父程式決定。父程式的運作會因執行副程式或函數而暫停，直到副程式或函數執行結束為止：



任何區域中的任何程式都可以有一個或多個副程式。副程式只被一個父程式擁有，副程式可以使用區域變數及定義字。除了 **SFC** 或 **FC** 語言之外的其它語言都能用來撰寫副程式。“**函數**”區內的程式是可以被案件中的其它程式呼叫使用的一種副程式，與其它副程式不同的是“**函數**”區內的程式不需選擇父程式。“**函數**”區的程式可以呼叫此區內的其它程式，也可以被放置在程式庫中。

注意：ISaGRAF 系統在執行不支援遞迴呼叫函數，如果“**函數**”區的程式被自己或是被它自己呼叫的副程式呼叫的話，則在執行時會產生錯誤。

注意：函數或副程式並不會儲存屬於自身的區域變數值。函數或副程式並不是種樣例，因此不能呼叫功能方塊。

副程式的界面必須清楚的定義，副程式的每個呼叫或是返回的參數都要有一種**形態和唯一的名稱**。爲了要支援 **ST** 語法的規則，因此返回參數和副程式的名稱必須相同。

下面的例子顯示出，如何以不同語法，在副程式的本體中設定返回參數的值：

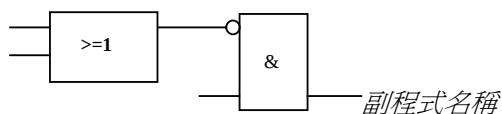
ST: 指定返回參數使用自己的名稱
(與副程式相同的名稱)：

副程式名稱 := <expression>;

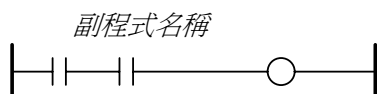
IL: 在整個程序的最後，所得到目前的值 (IL 暫存器)，會被儲存到返回的參數中：

LD10
ADD 20 (* 返回的參數值 = 30 *)

FBD: 使用自己的名稱來設定返回參數：

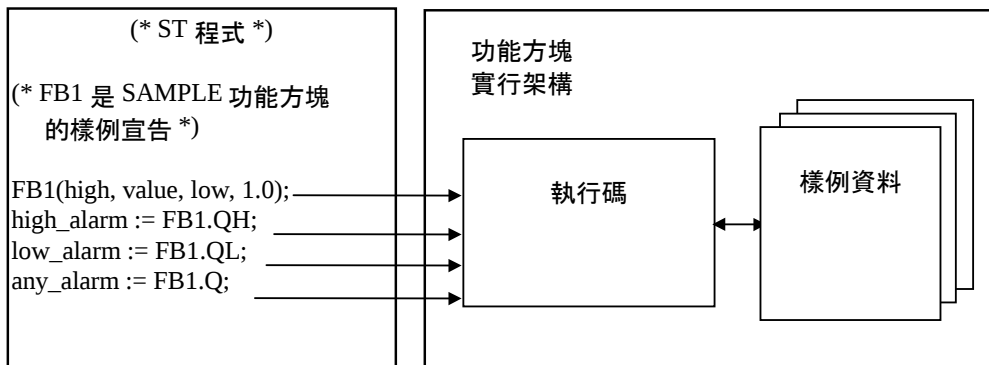


LD: 使用一名稱為返回參數的線圈符號：



B.1.5 功能方塊

功能方塊能使用的語法有：LD、FBD、ST 或 IL。功能方塊是可以做成樣例的，也就是說一個功能方塊的區域變數可以被其它樣例複製。當呼叫程式的一個方塊時，事實上你是在呼叫方塊的樣例：雖然相同的執行碼被呼叫，但會對不同的樣例配置不同的資料區間。樣例的變數值是從一個週期到另一週期時被儲存起來。



注意：

以 IEC 語言所撰寫的功能方塊，無法去呼叫另一個功能方塊：樣例的架構只能用來管理方塊本身的區域變數。下面是一些標準功能方塊表列，你無法在其內部使用一個 IEC 標準的功能方塊：

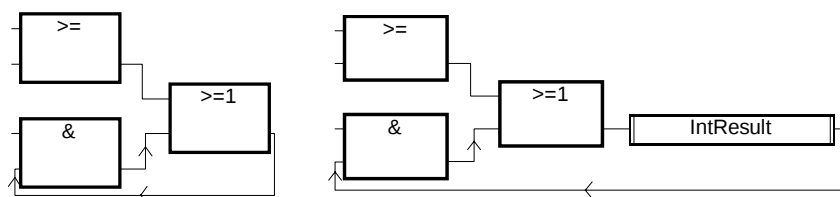
SR、RS、R_trig、F_trig、SEMA、CTU、CTUD、TON、TOF、TP、CMP、StackInt、AVERAGE、HYSTER、LIM_ALARM、INTEGRAL、DERIVATE、BLINK、SIG_GEN

基於同樣的原因，你不能使用 Positive 或 Negative 的接點或線圈，或 Set 與 Reset 線圈。

對 3.0x 版本的目標硬體來說，用 TSTART 與 TSTOP 來啟動或停止計時器的功能，不能使用於功能方塊中，但在 3.20 板的硬體上就能工作。

當你的功能方塊中需要作一個迴圈，迴圈之前你必須使用區域變數，如下例子所示：

這個將無法工作： 這個將正常工作：



B.1.6 描述語言

程式可以用下列任何以繪圖或文字形態的語言來撰寫：

循序式功能圖 (SFC) 用於高階的程式撰寫

流程圖 (FC) 用於高階的程式撰寫

功能方塊 (FBD) 用於複雜的週期性的運作

階梯圖 (LD) 只用於布林變數的運作

結構式文字 (ST) 用於任何週期性的運作

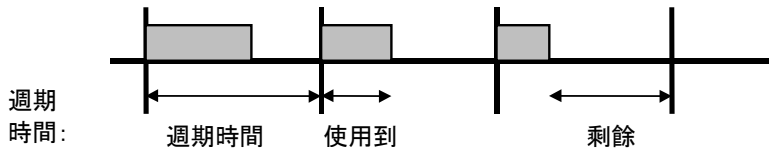
指令集 (IL) 用於低階的運作

相同的程式不能混合多種語言來撰寫。每當程式被創造時，必須選擇一種語言來描述這個程式，並且往後都不能更改，但是，FBD 和 LD 則可以混合運用在同一個程式中。

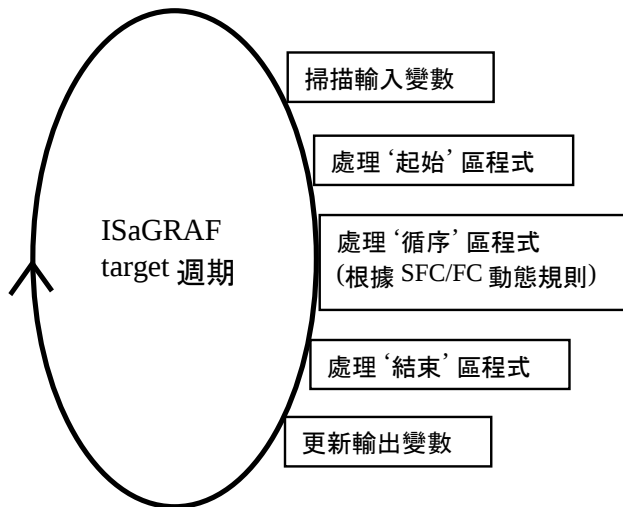
B.1.7 執行的規則

ISaGRAF 是一種同步的系統，所有的運作皆是由一計時器來觸發。基本的時間單位稱為週期時間：

語言參考



在一個硬體循環時間內的標準運作處理為：



此系統就可以：

保證輸入變數在一個週期中能保持相同的值，
保證輸出變數在一個週期中不會被重複更新，
相同的變數可以很安心的使用於不同的程式中，
估計與控制完整應用程式的回應時間。

B.2 共同物件

ISaGRAF 程式設計所使用的資料基本上有一些主要特徵和共同物件。這種物件可以使用在任何以 **SFC**、**FC**、**FBD**、**LD**、**ST** 或 **IL** 所撰寫的任何程式中。

B.2.1 基本型態

程式 (用任何語言撰寫的) 中所用的任何常數、運算式或變數都必須標示出其所屬的型態。在圖形式運算和文字敘述時，此型態必須一致。目前程式設計可以使用的基本型態如下：

布林 (BOOLEAN)： 邏輯 (真或假) 值

類比 (ANALOG)： 整數或實數 (浮點) 值

計時器 (TIMER)： 時間值

訊息 (MESSAGE)： 文字字串

注意：計時器型態只能包含少於一天的時間值，不能用來儲存日期。

B.2.2 常數表示法

常數形態也是一種型態。相同符號不能用來表示不同型態的常數形態。

B.2.2.1 布林常數表示法

只有兩種布林常數形態：

TRUE (真) 等價於整數值 1

FALSE (假) 等價於整數值 0

語言參考

“True” 及 “False” 這兩個關鍵字在拼字方面是不分大小寫的。

B.2.2.2 整數類比常數表示法

整數常數形態是指具有正負號的長整數 (32 位元) 值：從-2147483647 到 +2147483647。整數類比常數在表達上必須以下面所列的任一**基底**為基本原則。整數常數必須在數值前加上一個**前置字** (prefix) 以識別其所使用的基底：

基底	前置字	範例
十進位法 (DECIMAL)	無	-908
十六進位法 (HEXADECIMAL)	“16 #”	16#1A2B3C4D
八進位法 (OCTAL)	“8# ”	8#1756402
二進位法 (BINARY)	“2# ”	2#1101_0001_0101_ 1101

底線字元 (‘_’) 可以用來將數位的群組隔開，這作法並不是特別重要，只是用來增加常數形態的可讀性。

B.2.2.3 實數類比常數表示法

實數類比常數形態可以用**十進位法**或**科學記號表示法**來表示。其中，整數和小數的部份是用**小數點** (‘.’) 來隔開。另外，小數點必須用來辨識實數常數形態和整數常數形態的不同。在科學記號表示法中，是用 ‘E’ 或 ‘F’ 字母來隔開**尾數部份**和**指數部份**，而指數部份必須是一個有正負號的整數值，大小從-37 到+37。以下是實數類比常數形態的範例：

3.14159 -1.0E+12

+1.0 1.0F-15
-789.56 +1.0E-37

“123” 這種表示法並不能表示一個實數常數形態。它正確的實數表示法應該是 “123.0”。

B.2.2.4 時間常數表示法

時間常數形態的時間數值是從 **0 second** (0 秒) 到 **23h59m59s999ms** (23 時 59 分 59 秒 999 毫秒)。可以允許的最小單位是一個毫秒。所使用的標準時間單位如下：

時 小時的數目後必須緊接著 “h” 字母
分 分的數目後必須緊接著 “m” 字母
秒 秒的數目後必須緊接著 “s” 字母
毫秒 毫秒的數目後必須緊接著 “ms” 字母

時間常數形態的數值前必須以 “T#” 或 “TIME#” 為前置字。前置字和時間單位的字母是不分大小寫的。有些單位若沒有使用到，就可以不必標示出來。範例如下：

T#1H450MS 一小時又 450 毫秒
Time#1H3M 一小時又三分

“0” 這種表示法並不能表示一個時間數值，而是一個類比常數。

B.2.2.5 訊息字串常數表示法

字串或訊息常數形態是指文字字串。文字字串前必須有一個單引號，而且文字字串後必須緊接著另一個單引號。例如：

‘THIS IS A MESSAGE’

語言參考

警告：在一個字串常數形態中，不能使用單引號 “'” 字元。撰寫程式原始碼時，若想使用字串常數形態，必須將它撰寫在同一列上。而它的長度不能超過 255 個字元 (包括空白字元)。

可以用兩個單引號來表示空字串常數形態，而這兩個單引號之間不可以包含任何的空白或跳位字元，如下：

'' (*這是一個空字串*)

若在一些特殊的字元前加上一個錢號字元 ('\$')，就可以將其視為非列印字元，使用在字串常數形態中。各種情形如下所示：

	含意	ASC II (十 六進 位)	範例
\$ \$	'\$' 字元	16# 24	'I paid \$\$5 for this'
\$,	單引號	16# 27	'Enter \$'Y\$' for YES'
\$ L	換行	16# 0a	'next \$L line'
\$ R	歸位到 行首	16# 0d	' llo \$R He'
\$ N	新的 一行	16# 0d0 a	'This is a line \$N'
\$ F	新的 一頁	16# 0c	'lastline \$P first line'

\$ T	跳位符 號	16# 09	'name \$T size \$T date'
\$ h h (*)	任何字 元	16# hh	'ABCD \$41\$42\$43\$44' =

(*) “hh” 是將所要表達的字元用 ASCII 碼的十六進位法數值表示出來。

B.2.3 變數

對一個程式來說，其所使用的變數可以是**區域的**或**全域的**。區域變數只能在一個程式使用。而全域變數可以在案件的任何程式中使用。變數名稱必須遵循下面的規則：

字數不可以超過十六個字元

第一個字元必須是一個**英文字母**

除了第一個字元之外，其他的字元可以是**英文字母**、**阿拉伯數字**或**底線字元**

B.2.3.1 保留字

下面列出了 ISaGRAF 的保留字。這些保留字 (關鍵字) 不能是另一個程式、變數、‘C’函數或功能方塊的名稱：

ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,

語言參考

{ BCD_TO_BOOL , BCD_TO_INT , BCD_TO_REAL ,
BCD_TO_STRING , BCD_TO_TIME , BOO , BOOL ,
BOOL_TO_BCD , BOOL_TO_INT , BOOL_TO_REAL ,
BOOL_TO_STRING , BOOL_TO_TIME , BY , BYTE ,
(CAL , CALC , CALCN , CALN , CALNC , CASE , CONCAT ,
CONSTANT , COS ,
[DATE , DATE_AND_TIME , DELETE , DINT , DIV , DO , DT ,
DWORD ,
{ ELSE , ELSIF , EN , END_CASE , END_FOR , END_FUNCTION ,
END_IF , END_PROGRAM , END_REPEAT ,
END_RESSOURCE , END_STRUCT , END_TYPE , END_VAR ,
END_WHILE , ENO , EQ , EXIT , EXP , EXPT ,
[FALSE , FEDGE , FIND , FOR , FUNCTION ,
(GE , GFREEZE , GKILL , GRST , GSTART , GSTATUS , GT ,
[IF , INSERT , INT , INT_TO_BCD , INT_TO_BOOL ,
INT_TO_REAL , INT_TO_STRING , INT_TO_TIME ,
(JMP , JMPC , JMPCN , JMPN , JMPNC ,
[LD , LDN , LE , LEFT , LEN , LIMIT , LINT , LN , LOG , LREAL ,
LT , LWORD ,
[MAX , MID , MIN , MOD , MOVE , MSG , MUL , MUX ,
[NE , NOT ,
(OF , ON , OPERATE , OR , OR_MASK , ORN ,
[PROGRAM ,
[R , READ_ONLY , READ_WRITE , REAL , REAL_TO_BCD ,
REAL_TO_BOOL , REAL_TO_INT , REAL_TO_STRING ,
REAL_TO_TIME , REDGE , REPEAT , REPLACE ,
RESSOURCE , RET , RETAIN , RETC , RETCN , RETN , RETNC ,
RETURN , RIGHT , ROL , ROR ,

```

{      S , SEL , SHL , SHR , SIN , SINT , SQRT , ST , STN , STRING ,
      STRING_TO_BCD , STRING_TO_BOOL , STRING_TO_INT ,
      STRING_TO_REAL , STRING_TO_TIME , STRUCT , SUB ,
      SYS_ERR_READ , SYS_ERR_TEST , SYS_INITALL ,
      SYS_INITANA , SYS_INITBOO , SYS_INITTMR ,
      SYS_RESTALL , SYS_RESTANA , SYS_RESTBOO ,
      SYS_RESTTMR , SYS_SAVALL , SYS_SAVANA ,
      SYS_SAVBOO , SYS_SAVTMR , SYS_TALLOWED ,
      SYS_TCURRENT , SYS_TMAXIMUM , SYS_TOVERFLOW ,
      SYS_TRESET , SYS_TWRITE , SYSTEM ,
;      TAN , TASK , THEN , TIME , TIME_OF_DAY , TIME_TO_BCD ,
      TIME_TO_BOOL , TIME_TO_INT , TIME_TO_REAL ,
      TIME_TO_STRING , TMR , TO , TOD , TRUE , TSTART , TSTOP ,
      TYPE ,
(      UDINT , UINT , ULINT , UNTIL , USINT ,
\      VAR , VAR_ACCESS , VAR_EXTERNAL , VAR_GLOBAL ,
      VAR_IN_OUT , VAR_INPUT , VAR_OUTPUT ,
\      WHILE , WITH , WORD ,
)      XOR , XOR_MASK , XORN

```

只要關鍵字 (keyword) 的字首是一個底線 (‘_’) 字元，它就是內部關鍵字，不可以在文字敘述的指令中使用。

B.2.3.2 變數的直接表示法

ISaGRAF 可以在程式的原始碼中使用**變數的直接表示法**，來表達空置的連結點。空置的連結點是指未連結至一個已宣告的 I/O 點變數的連結點。變數的直接表示法總是用一個 “%” 字元作為前導字元，以便識別。

對於單板上的連結點，其變數直接表示法的命名規則如下。“s” 是板子上的插槽編號，“c” 是板子上的連結點編號。

語言參考

%IXs.c 布林輸入板上的空置連結點
%IDs.c 整數輸入板上的空置連結點
%ISs.c 字串輸入板上的空置連結點
%QXs.c 布林輸出板上的空置連結點
%QDs.c 整數輸出板上的空置連結點
%QSs.c 訊息輸出板上的空置連結點

對於複合設備的連結點，其變數直接表示法的命名規則如下。“s”是設備的插槽編號，“b”是複合設備中單板的指標，“c”是複合設備上單板的連結點編號。

%IXs.b.c 布林輸入板上的空置連結點
%IDs.b.c 整數輸入板上的空置連結點
%ISs.b.c 訊息輸入板上的空置連結點
%QXs.b.c 布林輸出板上的空置連結點
%QDs.b.c 整數輸出板上的空置連結點
%QSs.b.c 訊息輸出板上的空置連結點

範例如下：

%QX1.6 板子#1 的第六個連結點 (布林輸出)
%ID2.1.7 設備#2 上，板子#1 的第七個連結點 (整數輸入)

變數的直接表示法不可以有“實數”資料型態。

B.2.3.3 布林變數

布林意味邏輯。此變數可以是布林數值 **TRUE** (真) 或 **FALSE** (假)。布林變數通常會在布林形態中使用。它可以具有的屬性如下：

內部： 由程式更新數值的記憶體變數
常數： 具有初始值的唯讀記憶體變數

輸入： 連接至一個輸入裝置的變數 (由系統更新數值)

輸出： 連接至一個輸出裝置的變數

警告：當宣告布林變數的時候，可以定義在除錯期間取代 ‘true’ 和 ‘false’ 數值的字串。假如用適合於程式的語言將字串宣告成 ‘**定義字**’ (defined words)，就能在程式中使用這些字串。

B.2.3.4 類比變數

類比意味**連續**，此變數是具有正負號的整數或實 (浮點) 數。類比變數可以具有的格式如下：

整數 32 位元帶符號的整數：從-2147483647 到+2147483647

實數 標準 IEEE 32 位元浮點數值 (單精度)

正負號的一個位元+假數的 23 個位元+指數的 8 個位元

實數類比變數的指數數值不能少於-37 或大於+37。類比變數可以具有的**屬性**如下：

內部： 由程式更新數值的記憶體變數

常數： 具有初始值的唯讀記憶體變數

輸入： 連接至輸入裝置的變數 (由系統更新數值)

輸出： 連接至輸出裝置的變數

注意：當實數變數被連接至 I/O 裝置時，相對應的 I/O 驅動器會將之轉為等價的整數值。

警告：在同一個類比運算式中，不能將整數和實數類比變數或常數形態混和在一起使用。

B.2.3.5 計時器變數

計時器意味**時鐘**或**計數器**。此變數具有時間數值，而且通常在時間表示式中使用。時間數值不能超過**23h59m59s999ms** (23 小時 59 分 59 秒 999 毫秒)，而且不能是負值。計時器變數是以 32 位元的字組單位來儲存。它的內部表示法是一個以毫秒為單位的正數。

計時器變數可以具有的**屬性**如下：

內部： 由程式管理的記憶體變數，而且由 ISaGRAF 系統更新數值

常數： 具有初始值的唯讀記憶體變數

警告：計時器變數不可以具有輸入或輸出的屬性。

ISaGRAF 系統會自動更新計時器變數的數值。當計時器變數處於**動作** (active) 狀態時，它的數值會根據目標硬體系統的即時時鐘自動地遞增。可以用下面的 **ST** 語言敘述式來控制一個時間：

TSTART 開始計時器的自動更新

TSTOP 停止計時器的自動更新

B.2.3.6 訊息字串變數

訊息或字串變數包含文字字串。在程序運作期間，可以改變字串的長度。當宣告變數的時候，訊息變數的長度不能超過特定的容量 (最大長度)。訊息的容量限制是 255 個字元。訊息變數可以具有的**屬性**如下：

內部： 由程式更新數值的記憶體變數

常數： 有一個初始值的唯讀記憶體變數

輸入： 連接至一個輸入裝置的變數 (由系統更新數值)

輸出： 連接至一個輸出裝置的變數

字串變數可以包含標準 ASCII 表格 (ASCII 碼從 0 到 255) 中的任何字元。文字字串中可以包含空 (null) 字元。在標準的 ISaGRAF 程式庫中，有一些“C” 函數不能正確地操作包含空 (ASCII 碼為 0) 字元的訊息。

B.2.4 註解

在文字敘述的語言 (例如 **ST** 和 **IL** 語言) 中，可以自由的插入註解。一個註解必須啓始於特定的字元 “(” 和終止於 “)” 。在 **ST** 語言的程式中，註解可以插在任何地方，而且可以不必寫在同一列。

以下是註解的範例：

```
counter := ival; (* 指定主計數器 *)  
(* 這是一個二行的  
註解 *)  
c := counter (* 你可以於任何地方置放註解 *) + base_value + 1;
```

註解裡不能插入另一個註解，也就是說，在一個註解裡不能使用 “(” 字元。

警告：在 **IL** 語言中，只能在一個指令列的最後加上註解。

B.2.5 定義字

ISaGRAF 系統允許使用者對常數形態，真和假布林形態，關鍵字或複雜的 **ST** 語言運算式再定義。達成這個目的的方式，就是必須指定一個**識別字** (identifier) 名稱給對應的表示式。例如：

```
YES 是 TRUE  
PI 是 3.14159  
OK 是 (auto_mode AND NOT (alarm))
```

語言參考

當定義這樣的等式關係之後，就可以在 **ST** 程式的任何地方使用此**識別字**，以取代附屬於此識別字的表示式。下面是使用識別字撰寫的**ST** 程式範例：

If OK Then

angle := PI / 2.0;

isdone := YES;

End_if;

對一個程式來說，其所使用的定義字的範圍可以是**區域的**、**全域的**或**共同的**。
區域的定義字只能在一個程式中使用。

全域的定義字可以在案件的任何程式中使用。

共同的定義字可以在任何案件的任何程式中使用。

使用者需注意到一件事，就是共同的定義字可以由拷貝 (Archive) 管理員分開地儲存。

警告：當使用者用不同的 **ST** 等價式將同樣的識別字定義兩次時，只有最後的定義式會成立。例如：

定義： **OPEN** 是 **FALSE**
 OPEN 是 **TRUE**

意味： **OPEN** 是 **TRUE**

定義字命名時必須遵循下面的規則：

名稱不可以超過**16** 個字元

第一個字元必須是一個**英文字母**

若名稱的字數不止一個字元，那麼除了第一個字元之外，其他的字元可以是**英文字母**、**阿拉伯數字**或**底線** ('_') 字元

警告：不可以用一個定義字去定義另外一個定義字。例如，不能有下面的情形：

PI 是 **3.14159**
—— **PI2** 是 **PI*2** ——

只能用常數或變數和運算式來撰寫完整的等價式子：

PI2 是 **6.28318**

B.3 SFC 語言

循序式功能圖 (Sequential Function Chart, SFC) 是一種用來描述**循序操作**的**繪圖式語言**。在流程圖中，流程是用一連串定義明確的**步驟 (steps)** 來表示，步驟間以**轉移條件 (transition)** 連結起來。每一個轉移條件都有一個**布林判斷條件 (boolean condition)**。步驟中的**行為**可以用其它語言 (ST、IL、LD 和 FDB) 來詳述。

B.3.1 SFC 圖表的主要格式

SFC 程式是以一連串的**步驟**和**轉移條件**圖形來表示，而步驟與轉移條件之間是用**有方向性的線 (oriented links)** 連結在一起。多連結線可以用來表示發散和收斂。完整程式的某些部份可以被分隔出來，然後用一個所謂**巨集步驟 (macro steps)** 的單一符號表示在主要圖形裡。SFC 的基本**繪圖規則**如下：

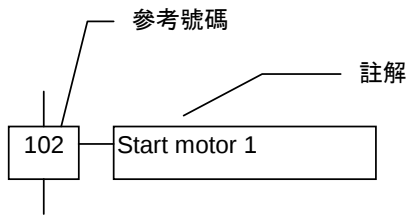
- 步驟之後不能緊接著另外一個步驟
- 轉移條件之後不能緊接著另外一個轉移條件

B.3.2 SFC 基本元件

SFC 語言的基本元件 (圖形符號) 是：步驟 (steps) 和初始步驟 (initial steps)、轉移條件 (transitions)、有方向性的線 (oriented links) 及跳躍至步驟 (jumps to a step)。

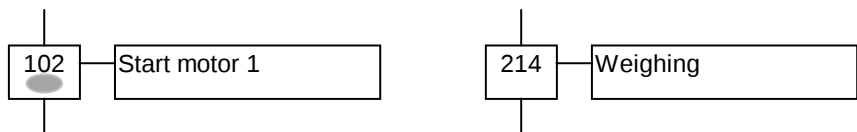
B.3.2.1 步驟和初始步驟

步驟是用一個單一方格來表示。每一個步驟的方格符號中會有一個**參考號碼**。步驟的註解會撰寫在連結於步驟符號的一個長方形中。註解的寫法**沒有固定的格式** (不是程式設計語言的部份)。以上這些信息稱為步驟的**第一層 (Level 1)**，圖示如下：



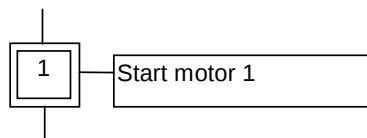
程式執行時，假如步驟中有一個**權杖記號 (token)**，就表示這個步驟是正在動作 (active) 中，圖示如下：

正在動作的步驟： 沒有動作的步驟：



SFC 程式的**初始情形**是用**初始步驟**來表示。初始步驟是一個**雙重邊框**的圖形符號。當程式開始執行的時候，權杖符號會自動地出現在每一個初始步驟中。初始步驟的圖示如下：

初始步驟：



一個 SFC 程式**至少**必須包含一個初始步驟。

步驟具有一些屬性。這種屬性可以在任何其他語言中使用：

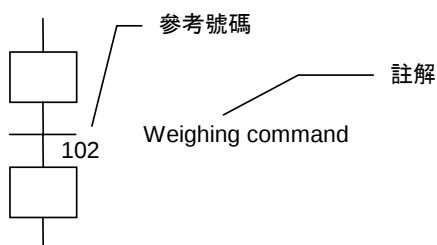
GSnnn.x 步驟動作的情形 (布林值)

GSnnn.t 步驟動作的持續時間 (時間值)

(其中，**nnn** 是步驟的參考號碼)

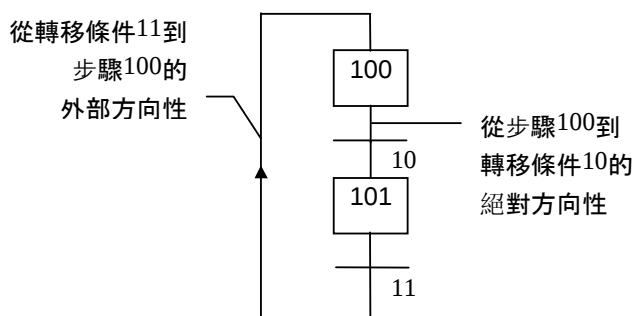
B.3.2.2 轉移條件

轉移條件是用交叉於連結線的一個小水平棒來表示。每一個轉移條件有一個**參考號碼**，其被寫在轉移條件符號的右下角。轉移條件的註解會寫在轉移條件符號的右邊。註解的寫法**沒有固定格式**（不是程式設計的部份）。以上這些信息稱為轉移條件的**第一層 (Level 1)**，圖示如下：



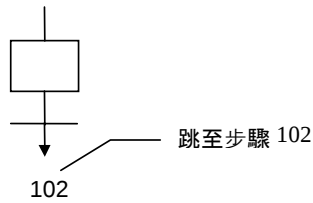
B.3.2.3 有方向性的連結

步驟和轉移條件之間是用單一線連結起來，而這些線就是有方向性的線。當線的方向性沒有明確給定的時候，其是由上到下。

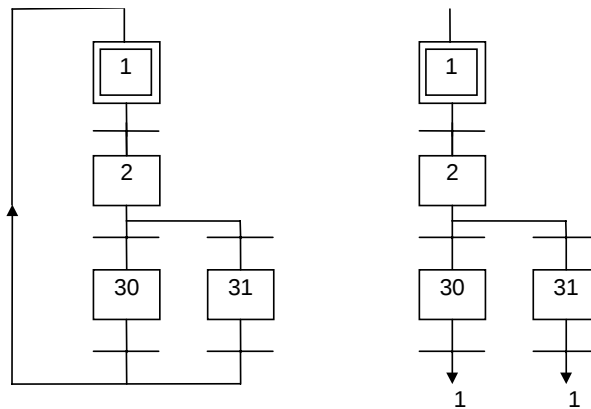


B.3.2.4 跳躍至步驟

跳躍符號可以用來表示從一個轉移條件到一個步驟的連接線，這樣就不一定要畫連接線了。跳躍符號必須指向目的步驟的參考號碼，圖示如下：



跳躍符號不能用來表示從一個步驟到一個轉移條件的連結線。下面是跳躍的範例 — 兩個圖形是相等的：



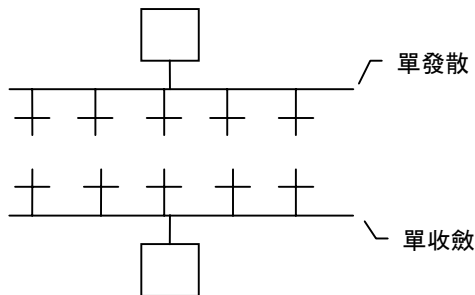
B.3.3 發散和收斂

發散是指從一個 SFC 符號 (步驟或轉移條件) 到很多其他 SFC 符號的**多連接線** (multiple connection links)。收斂是指從不止一個的 SFC 符號到一個其他符號的多連接線。發散和收斂有單或雙的不同情形。

B.3.3.1 單發散 (single divergences)

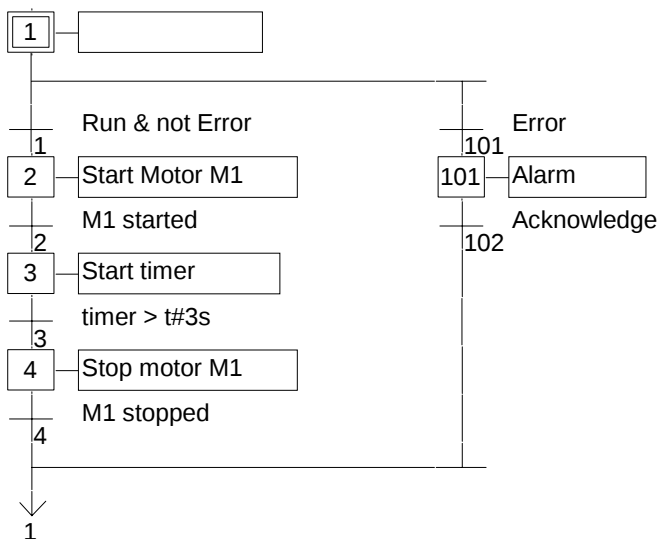
單發散是指從一個步驟到很多轉移條件的多連結線，它允許正在行動的權杖符號進入許多分支的其中一支。單收斂 (single convergence) 是指從很多的轉移條件到同一個步驟的多連結線，它通常用來將開始於一個單發散的 SFC 分支聚集在一起。單發散和單收斂是以單一的水平線來表示，如下圖所示：

語言參考



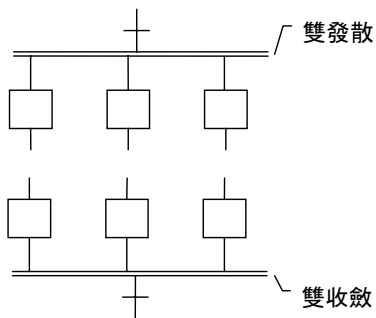
警告：在一個單發散開始的不同轉移條件中的判斷條件並不是絕對互斥的 (not implicitly exclusive)。而轉移條件的判斷條件中必須明確地詳述這種互斥情形，以便確保程式執行時，發散的一個分支中只有一個權杖符號經過。單發散和單收斂的範例如下：

(* 單發散和單收斂的 SFC 程式 *)



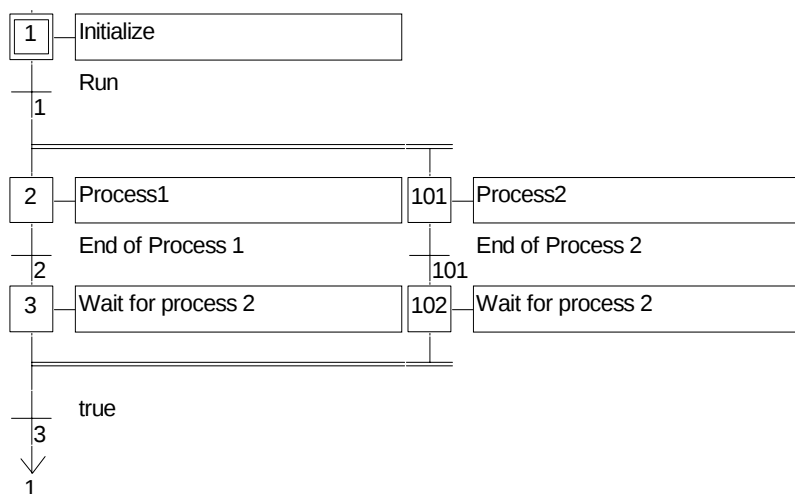
B.3.3.2 雙發散 (Double divergences)

雙發散是指從一個轉移條件到很多步驟的多連結線，它相當於程序的平行操作。雙收斂 (double convergence) 是指從很多步驟到相同轉移條件的多連結線，它通常用來將開始於一個雙發散的 SFC 分支聚集在一起。雙發散和雙收斂是以雙重水平線來表示，如下圖所示：



雙發散和雙收斂的範例如下：

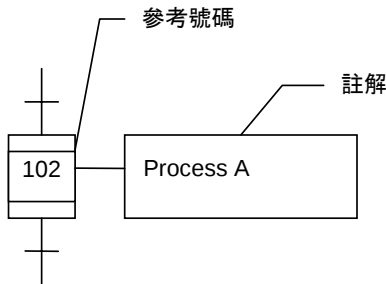
(* 雙發散和雙收斂的 SFC 程式 *)



B.3.4 巨集步驟 (Macro steps)

巨集步驟是另一些單獨造出的步驟及轉移條件的獨特表示方式。巨集步驟的主體會被另外地描述在相同 SFC 程式的其他地方。它會以一個單一符號出現在 SFC 的主流程中。巨集步驟的符號如下所示：

語言參考



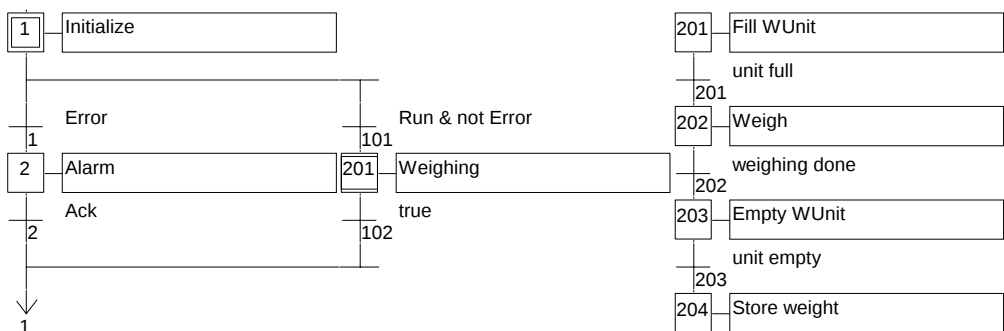
巨集步驟符號中的參考號碼是巨集步驟程式主體的第一個步驟的參考號碼。巨集步驟的程式主體必須開始於**開始步驟** (beginning step)，並且終結於**結束步驟** (ending step)。其圖形必須是獨立自足的 (self-contained)，也就是說，開始步驟沒有上連結線（沒有之前的轉移條件），結束步驟沒有下連結線（沒有之後的轉移條件）。巨集步驟的程式主體中可以使用另一個巨集步驟符號。

警告：因為巨集步驟是另一些**單獨**造出的步驟及轉移條件，所以在一個 SFC 程式中，同樣的巨集步驟只能用一次。

巨集步驟的範例如下：

(* 使用巨集步驟的 SFC 程式 *)

(* 主流程 *) (* 巨集步驟的程式主體 *)



B.3.5 步驟中的行為

步驟活動期間所執行的行為 (actions) 必須詳述在 SFC 步驟的第二層 (level 2) 中。此描述是用 SFC 的文字敘述特徵和其他語言 (例如 ST 敘述式語言) 所撰寫的。行為的基本形態如下：

布林行為 (Boolean actions)

以 ST 語言撰寫的脈衝行為 (Pulse actions)

以 ST 語言撰寫的非儲存行為 (Non-stored actions)

SFC 行為 (SFC actions)

同一個的步驟中可以描述很多行為 (具有相同或不同的形態)。步驟還具有使用任何其他語言的特殊特徵：

呼叫副程式

使用 IL (Instruction List) 語言

B.3.5.1 布林行為

布林行為會隨著步驟的動作而指定布林變數。此布林變數可以是輸出變數或是內部變數，而且只有當步驟開始或停止活動時才會被指定。基本布林行為的語法如下：

<布林變數> (N) ; 將步驟動作的狀態指派給變數

<布林變數> ; 與上一個語法具有相同的效果 (因為 N 可以不表示出來)

!<布林變數> ; 將步驟動作狀態的反相指派給變數

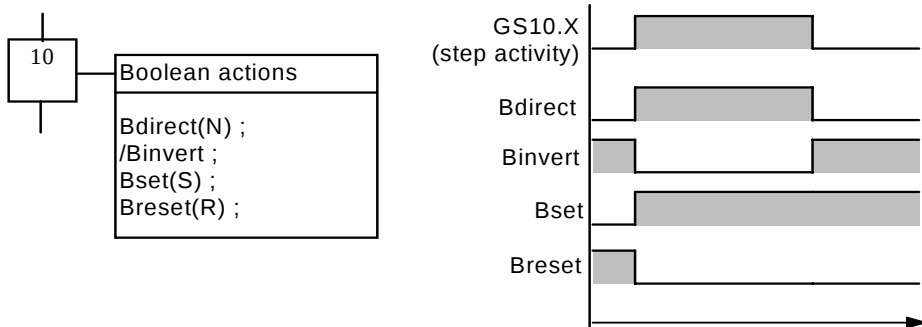
當步驟正在動作的時候，還可以將布林變數設定 (set) 或重置 (reset) 成其他狀態。設定及重置布林行為的語法如下：

<布林變數> (S) ; 當步驟動作的狀態變成真 (TRUE) 的時候，將變數設定成真 (TRUE)

<布林變數> (R) ; 當步驟動作的狀態變成真 (TRUE) 的時候，將變數重置成假 (FALSE)

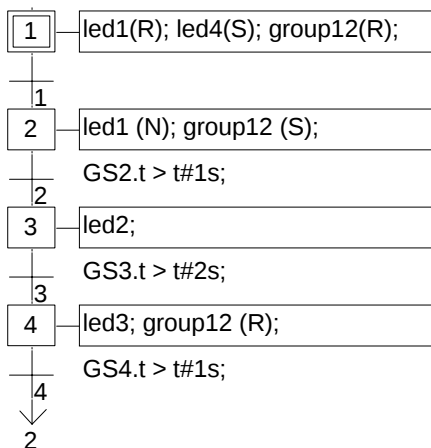
語言參考

此布林變數必須是輸出變數或是內部變數。左下方的 **SFC** 程式設計會引導出右下方的布林行為：



布林行為的範例如下：

(* 使用布林行為的 **SFC** 程式 *)

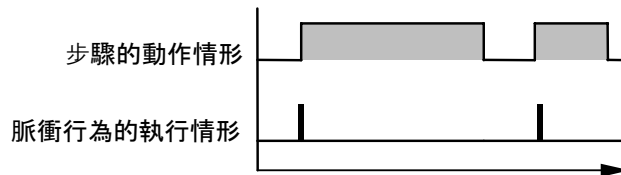


B.3.5.2 脈衝行為

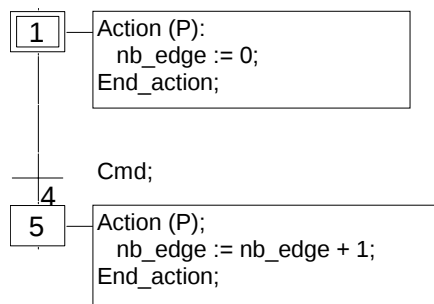
脈衝行為是用 **ST** 或 **IL** 指令撰寫的，它在步驟活動時只會執行一次。其指令是根據下面的 **SFC** 語法撰寫的：

ACTION (P) :
 (* ST 敘述 *)
END_ACTION ;

下面是一個脈衝行為的結果：



脈衝行為的範例如下：



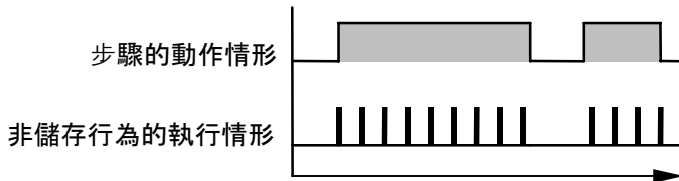
B.3.5.3 非儲存行為

非儲存 (標準) 行為是用 ST 或 IL 指令撰寫的。在步驟整個活動期間的每一個週期 (each cycle)，非儲存行為都會執行一次。其指令是根據下面的 SFC 語法撰寫的：

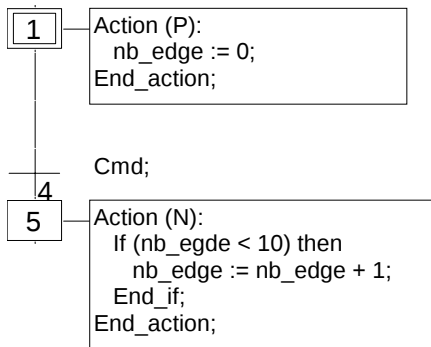
ACTION (N) :
 (* ST 敘述 *)
END_ACTION ;

下面是一個非儲存行為的結果：

語言參考



非儲存行為的範例如下：



B.3.5.4 SFC 行為

SFC 行為是一個 SFC 子程序，它會根據步驟動作狀態的改變而開始或終止。

SFC 行為有 **N** (非儲存, Non stored) , **S** (設定, Set) , 或 **R** (重置, Reset) 三種限定語法。基本 SFC 行為的語法如下：

<子程式> (N); 當步驟一開始動作，就啓始子程序，而且當步驟一停止動作，就終止子程序。

<子程式>; 與上一個語法有相同的效果 (因為 **N** 可以不表示出來)

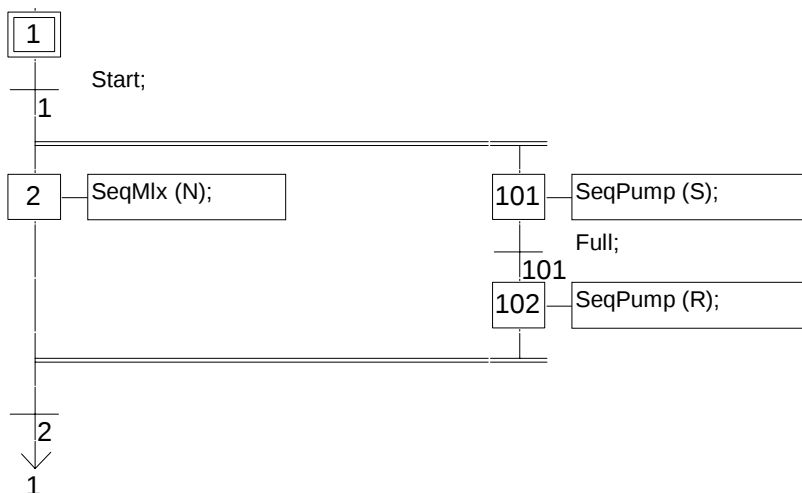
<子程式> (S); 當步驟一開始動作，就啓始子程序。但是當步驟停止動作時，不會有任何影響。

<子程式> (R); 當步驟一開始動作，就終止子程序。但是當步驟停止動作時，不會有任何影響。

指定行為的 SFC 程序必須是目前正在編輯的程式的 **SFC 子程式**。可以注意到的是，在 SFC 行為中使用 **S** 或 **R** 語法的效果，絕對會跟在 ST 脈衝行為程式中使用 **GSTART** 和 **GKILL** 敘述所產生的效果一樣。

下面是一個 SFC 行為的範例。其中，SFC 主程式的名稱是 **Father**，它有兩個 SFC 子程式，名稱分別是 **SeqMlx** 和 **SeqPump**。SFC 父程式的程式設計部分如下圖所示：

(* 使用 SFC 行為的 SFC 程式 *)



B.3.5.5 在行為中呼叫函數和功能方塊

假如想在 SFC 行為方塊中直接呼叫副程式，函數或功能方塊（以 ST、IL、LD 或 FBD 語言撰寫的）或 “C” 函數和 “C” 功能方塊，必須遵循下面的語法：

呼叫副程式、函數和 “C” 函數時：

```

ACTION (P) :
result := sub_program ( ) ;
END_ACTION;

```

或

```

ACTION (N) :

```

語言參考

```
result := sub_program ( );  
END_ACTION;
```

呼叫用“C” 或用 ST、IL、LD、FBD 語言所寫的功能方塊時：

```
ACTION (P) :  
Fbinst(in1, in2);  
result1 := Fbinst.out1;  
result2 := Fbinst.out2;  
END_ACTION;
```

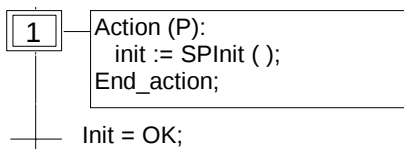
或

```
ACTION (N) :  
Fbinst(in1,in2);  
result1 := Fbinst.out1;  
result2 := Fbinst.out2;  
END_ACTION;
```

若想知道更詳細的語法，請參考 ST 語言的章節。

在行為方塊中呼叫一個副程式的範例如下：

(* 在行為方塊中呼叫一個副程式的 SFC 程式 *)



B.3.5.6 使用 IL 語言

假如想在 SFC 行為方塊中直接用 IL 語言撰寫程式，必須遵循下面的基本語法：

ACTION (P): (* 或用 N *)

#info=IL

<指令>

<指令>

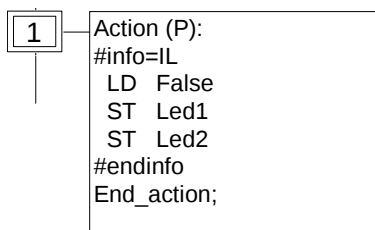
....

#endinfo

END_ACTION;

“#info=IL” 和 “#endinfo” 這兩個特別的關鍵字必須依照上面的方式撰寫，而且只能寫成**小寫**。另外，關鍵字的中間、後面或前面不能插入空白或跳位字元。在行為方塊中撰寫 IL 程式的範例如下：

(* 在行為方塊中撰寫 IL 程序的 SFC 程式 *)



B.3.6 轉移條件的判斷條件

每一個轉移條件的判斷條件都會附上一個**布林型態**。判斷條件通常是用 ST 語言或 LD 語言 (LD 快速編輯器) 撰寫的。判斷條件的撰寫是屬於轉移條件的**第二層 (Level 2)**。然而，還有其他結構可以使用：

ST 語言規則

LD 語言規則

IL 語言規則

在轉移條件中呼叫函數

警告：當轉移條件沒有附上任何運算式時，它的判斷條件為**真 (TRUE)**。

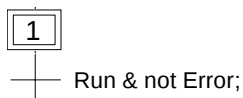
B.3.6.1 ST 語言規則

轉移條件的**判斷條件**可以用 **ST 結構化文字** 語言來撰寫。而判斷條件的完整運算式必須具有**布林**型態，而且必須終結於**分號** (;)，語法如下：

< 布林運算式 >;

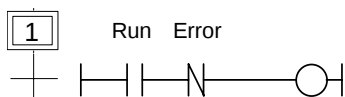
此運算式可以是真 (TRUE) 或假 (FALSE) 的常數形態、單一輸入或內部布林變數、或結果為布林值的變數運算。在轉移條件中用 ST 語言撰寫程式的範例如下：

(* 在轉移條件中用 ST 語言撰寫程式的 SFC 程式 *)



B.3.6.2 LD 語言規則

轉移條件的**判斷條件**可以用 **LD 階梯圖** 語言來撰寫。其 LD 圖形僅能由一個線圈和一個導軌組成的。線圈值就是轉移條件的值。在轉移條件中用 LD 語言撰寫程式的範例如下：



B.3.6.3 IL 語言規則

SFC 轉移條件可以直接用 **IL 指令集** 語言來撰寫，語法如下：

#info=IL
<指令>

<指令>

....

#endinfo

當 IL 程式執行完畢後，存在 IL 登錄器 (IL register) 中的**目前結果**就是轉移條件判斷條件的值：

目前結果等於 0 → 判斷條件是假 (FALSE)

目前結果不等於 0 → 判斷條件是真 (TRUE)

“#info=IL” 和 “#endinfo” 這兩個特別的關鍵字必須依照上面的方式撰寫，而且只能寫成**小寫**。兩個關鍵字的中間，後面或前面不能插入空白或跳位字元。在轉移條件中撰寫 IL 程式的範例如下：

(*在轉移條件中撰寫 IL 程式的 SFC 程式*)

```

1
├── #info=IL
│   LD Run
│   &N Error
│   #endinfo

```

B.3.6.4 在轉移條件中呼叫函數

在轉移條件中可以呼叫任何副程式或函數 (以 FBD、LD、ST 或 IL 語言撰寫的)，或 “C” 函數來決定判斷條件的結果，語法如下：

< 副程式 > ();

副程式或函式的回傳值必須是布林型態，而且就是轉移條件判斷條件的值：

回傳值等於 FALSE → 判斷條件是 FALSE

回傳值等於 TRUE → 判斷條件是 TRUE

語言參考

在轉移條件中呼叫副程式的範例如下：

(*在轉移條件中呼叫副程式的 SFC 程式*)



B.3.7 SFC 動態規則

SFC 語言的**五項**動態規則為：

初始情形

初始情形就是開始操作時處於動作狀態的**初始步驟**。每一個 SFC 程式**至少**要有一個初始步驟。

清除轉移條件

轉移條件是**有效的** (enabled) 或**無效的** (disabled) 。也就是說，當緊接轉移條件的上一層步驟是**正在動作**時，它就是有效的，否則是無效的。轉移條件有以下的情形時，才可以被**清除**：

轉移條件是有效的時候，而且

轉移條件的判斷條件是真 (true) 的時候

改變動作中步驟的狀態

清除轉移條件的同時會讓緊接轉移條件的下一個步驟開始動作，並且使緊接轉移條件的上一個步驟停止動作。

同時清除轉移條件

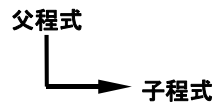
雙重線可以用來顯示必須被同時清除的轉移條件。假如這種轉移條件被分開顯示時，轉移條件上一個的步驟的動作情形 (GSnnn.x) 就可以用來當成是這些轉移條件的判斷條件。

同時讓步驟變成動作和無動作

在操作期間，假如同時讓一個步驟變成動作中和無動作的狀態，那麼最後的結果是讓此步驟變成動作。

B.3.8 SFC 程式的組織

ISaGRAF 系統可以撰寫垂直架構的 SFC 程式。SFC 程式的架構是一棵**組織樹** (hierarchy tree)。每一個 SFC 程式可以控制 (開始、刪除 ...) 其他的 SFC 程式。受 SFC 程式控制的這種程式稱爲此 SFC 程式的**子程式** (children)。所有的 SFC 程式會以“**父程式與子程式**” (father - child) 的關係連結成一棵主要的**組織樹**：



組織的架構暗示了幾項基本規則：

沒有父程式的 SFC 程式稱爲 SFC 主程式 (main SFC programs)

當應用程式開始時，ISaGRAF 系統會將 SFC 的主要程式執行起來

一個程式可以有一些子程式

一個子程式只能有一個父程式

子程式只能由其父程式控制

程式不能控制其子程式的子程式

SFC 父程式對其子程式可以採取的基本控制行爲：

開始 (Start) (GSTART) 啓始子程式：讓子程式的初始步驟執行起來。

此時，子程式的子程式並沒有自動的啓始。

刪除 (Kill) (GKILL) 將子程式目前執行的所有步驟皆終止掉。而子程式的所有子程式也會被終止掉。

冰凍 (Freeze) (GFREEZE) 冰凍程式的執行 (讓所有正在動作的步驟皆暫停下來，並且暫停轉換條件執行)，而且將這些步驟的狀態紀錄下來，以便將來再重新執行。子程式的所有子程式也會被暫停下來。

語言參考

重置 (Restart) (**GRST**) 將暫停的 SFC 程式重新啓始起來，而且是從暫停的步驟開始執行。程式的子程式並不會自動的啓始起來。

讀取狀態 (Get status) (**GSTATUS**) 取得子程式的目前狀態 (正在動作，停止動作或暫停動作) 。

B.4 流程圖語言

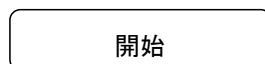
流程圖 (Flow Chart, **FC**) 是一種圖形式語言，用來敘述**順序式的操作**。流程圖由**動作 (Action)** 及**測試 (Test)** 所組成，動作及測試之間以**有方向性的連結線**表示資料流。多連結可用來表示發散及收斂。動作及測試也可以以 ST、LD 或 IL 語言來描述，除了 SFC 的任何語言所撰寫的函數及功能方塊可以被動作及測試所呼叫，流程圖程式也可以呼叫另一個流程圖程式，這個被呼叫的 FC 程式稱為呼叫程式的**副程式**。

B.4.1 FC 元件

以下是流程圖語言的圖形元件：

▣ FC 圖形的開始

“**開始**” (begin) 符號必須出現於流程圖程式的最前面，它必須是唯一的，且不能被刪除，表示流程圖程式動作時的初始狀態。下面是“開始”符號的表示法：



“開始”符號必須連結（底端）到其他圖形物件，假如開始符號未連結到其他物件，則這個流程圖程式將是無效的。

▣ FC 圖形的結束

“**結束**” (end) 符號必須出現於流程圖程式的結束處，它必須是唯一的且不能被刪除。可能沒有任何連結線連到此“結束”符號（迴圈流程），但是依然必須將“結束”符號放置於流程圖最底端。它表示圖形的最終狀態。下面是“結束”符號的表示法：

結束

“結束”符號一般來講會有一個連結線（於頂端），連到其他圖形元件，但是也可以沒有連結線（迴圈流程），不過依然可以出現於圖形底端。

FC 流程連結

流程連結以線來表示，代表圖形兩點之間的流向。連結終點以箭頭表示。以下是流程連結的圖形。



兩個連結不能從同一個連結點開始。

FC 動作

動作 (action) 符號表示所執行的動作，上面會標上編號及名稱作為識別。下面是“動作”符號的圖示：

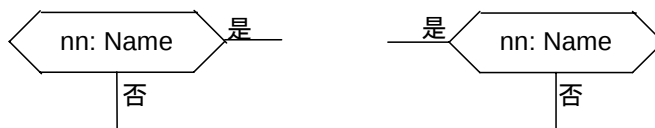
nn: Name

同一個流程圖中的不同物件不可以具有相同的名稱或邏輯號碼。動作的設計語言可以是 ST、LD 或 IL。動作總是與連結線連結，一個連結至它，一個從它開始連結至其他物件。

FC 條件

條件 (condition) 表示布林測試 (test)，上面會標上編號及名稱作為識別。根據所依附的 ST、LD 或 IL 表示式的結果，連結流向可以指向“是”或“否”兩個路徑。下面是條件符號可能的畫法：





同一個流程圖中的不同物件不可以具有相同的名稱或邏輯號碼。測試的程式可以為：

ST 的表示式。

單一的 LD 導軌，但是沒有符號連至唯一的線圈上。

IL 的指令。IL 的暫存器 (或現在的結果) 可用來當作條件判斷。

當以 ST 文字來設計時，表示式最後可以加上分號，也可以不加。當以 LD 來設計時，唯一的線圈表示條件值。若條件等於：

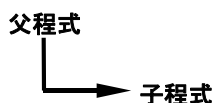
0 或 FALSE 將會走否的路徑

1 或 TRUE 將會走是的路徑

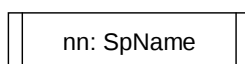
測試總是連結到連結線的尾端，且必須定義兩個連結的走向。

FC 副程式

系統可以以 FC 程式的垂直架構來表示，這些 FC 程式以**樹狀結構方式**組成，每一個 FC 程式可以呼叫其他的 FC 程式。被呼叫的 FC 程式稱為呼叫程式的**子程式**，而呼叫的 FC 程式稱為**父程式**。FC 程式使用“父 - 子”的關係連結成結構樹：



流程圖中的**副程式**符號表示呼叫流程圖副程式的關係。呼叫的 FC 程式會產生中斷，直到副程式執行完成之後才會再繼續執行。FC 副程式以號碼及名稱作為識別 (函數及功能方塊亦同)。下面是“副程式呼叫”符號的圖示：



語言參考

相同圖形中的不同物件不可以具有相同的邏輯號碼。FC 樹狀結構隱含基本的規則，如下：

沒有父程式的 FC 程式稱為 FC 主程式。

當應用程式開始時，FC 主程式會自動由系統呼叫起來執行。

子程式不能擁有兩個以上的父程式。

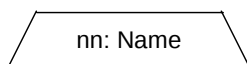
子程式僅能由它的父程式所呼叫。

程式不能呼叫它的子程式的子程式

於父程式圖形中，相同的副程式可以出現多次。FC 副程式呼叫表示副程式流程完整的執行。子流程執行時將會中斷父流程的執行。

FC I/O 特定動作

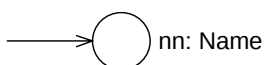
I/O 特定動作 (specific action) 符號表示指定的動作。如其他的動作一樣，I/O 特定動作以編號及名稱作為識別。I/O 特定動作與標準動作的意義一樣，只是 I/O 特定動作的目的是使流程圖形更具有可讀性，且用於圖形的非可攜性部分。I/O 特定動作是選擇性的元件。下面是“I/O 特定動作”符號的圖例：



I/O 特定方塊的行為與標準動作一樣，也就是具有相同的特性，都可使用 ST、LD 或 IL 來設計程式，以及有相同的連線規則。

FC 連結器

連結器用於表示圖形中兩點的連結關係，而不需直接繪出它們之間的連結線。連結器以圓圈表示，且連結到圖形的來源處。連結器必須加上目標點做為識別（一般為目標符號名稱，此名稱必須根據資料流的方向，放置於適當的位置）。下面為連結器的標準畫法：



連結器總是以已定義的流程圖符號為目標，這個目標符號以它的邏輯號碼為識別。

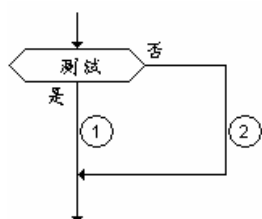
FC 註解

註解方塊包含一些與圖形語意無關的文字，它會被插入於流程圖視窗的任何空白處，且作為這個程式的文件。下面是“註解”符號的圖例：

comment text can
be on several lines...

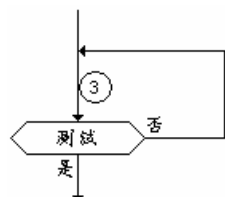
B.4.2 FC 複合結構

這個章節顯示定義於 FC 程式中的**複合結構**，這些結構為基本物件連結的組合。



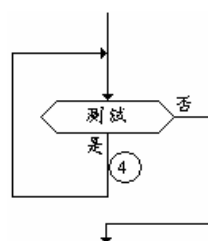
IF / THEN / ELSE

- (1) 放置插入的“THEN”動作
- (2) 放置插入的“ELSE”動作



REPEAT / UNTIL

- (3) 放置插入的重覆動作



WHILE / DO

- (4) 放置插入的重覆動作

B.4.3 FC 動態行為

FC 圖形的執行規則如下：

- 開始符號花費一個目標硬體週期
- 結束符號花費一個目標硬體週期，且結束流程圖的執行。這個符號到達後不會再執行任何的動作。
- 每一次碰到項目（動作、決定）時，流程會產生中斷，若這些項目早已完成，則於下一個週期，流程將會繼續下去。

注意：比照 SFC 來說，動作並不是穩定的狀態，當動作符號執行時並沒有重覆性的指令。

B.4.4 FC 檢查

除了依附的 ST、LD 或 IL 程式外，流程圖形本身必須遵循如下的主規則：

- 所有符號的所有“連結”點必須加以連線（連到“結束”符號的連結線可以去除）。
- 所有符號都必須連結在一起（沒有單獨存在的物件）。
- 所有的連結器都必須具有有效的目的。

其他次要的語法錯誤如下：

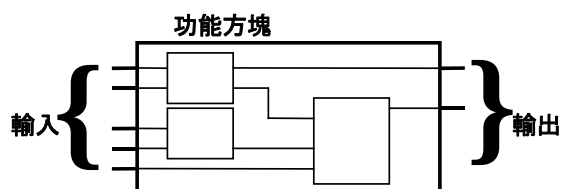
- 空的動作（沒有程式）於執行期排程時會被認為是步驟。
- 空的測試（沒有程式）會被考慮成“永遠成真”。

B.5 FBD 語言

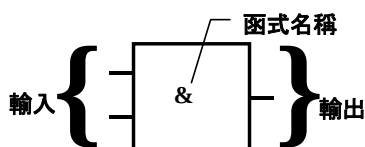
功能方塊圖 (Functional Block Diagram, FBD) 是一種繪圖式語言。在方塊圖編輯區裡，使用者可以利用 ISaGRAF 程式庫中的內建**函數**來撰寫複雜的程序，並且將這些程序用線連在一起。

B.5.1 FBD 方塊圖的主要格式

FBD 方塊圖是用來撰寫介於**輸入變數**和**輸出變數**之間的函數，此函數是以一連串**基本的功能方塊**所組成的。輸入和輸出變數與功能方塊是以**連接線**連接起來。功能方塊的輸出可以是另一個功能方塊的輸入。



FBD 程式中的完整函數都是用 ISaGRAF 程式庫中的標準的基本功能方塊所撰寫的。每一個功能方塊都有固定數目的**輸入連接點**和固定數目的**輸出連接點**。功能方塊是以單一的**長方形**來表示，**左邊**連接輸入，**右邊**連接輸出，而長方形符號中是功能方塊執行的單一**函數**名稱。功能方塊的每一筆輸入和輸出**型態**都已事先定義好了。



語言參考

FBD 程式的輸入變數必須連接到功能方塊的輸入連接點，而且每一個變數的型態都必須與功能方塊定義好的輸入型態相同。FBD 程式的輸入可以是**常數**形態、任何**內部 (internal)** 變數、**輸入 (input)** 變數或**輸出 (output)** 變數。

FBD 程式的輸出變數必須連接到功能方塊的輸出連接點，而且每一個變數的型態都必須與功能方塊定義好的輸出型態相同。FBD 程式的輸出可以是任何**內部 (internal)** 變數、**輸出 (output)** 變數或**程式名稱**（此情形只適用於 FBD 副程式）。當輸出被指定為目前編輯的副程式名稱時，此輸出就是此副程式的回傳值（傳給呼叫程式）。

輸入和輸出變數，功能方塊的輸入和輸出是以**連接線**連接在一起。連結線可以將以下所述的任兩個邏輯點**連接**起來：

輸入變數與功能方塊的輸入點

功能方塊的輸出點與另一個功能方塊的輸入點

功能方塊的輸出點與輸出變數

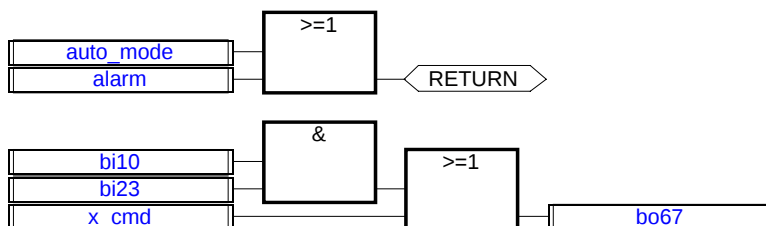
這種連接線是**有方向性的**，也就是說，程式的執行方向是從連接線的左端到連接線的右端。而且連接線兩端的資料**型態**必須**相同**。

多數右連接線可以將左端點的訊號傳送給所有右端點，而這些右端點必須具有相同型態。

B.5.2 跳回 (RETURN) 敘述

“<RETURN>” 關鍵字可以是一個圖表的輸出。它必須連接到功能方塊的布林輸出連接點。RETURN 敘述表示程式**有條件的結束**：假如連接到 RETURN 敘述的功能方塊輸出布林值是**真 (TRUE)** 時，則方塊圖的剩餘部份就不會執行。

(*使用 RETURN 敘述的 FBD 程式*)



(* ST 相等式 : *)

If auto_mode OR alarm Then

Return;

End_if;

bo67 := (bi10 AND bi23) OR x_cmd;

B.5.3 跳躍和標籤 (Jumps and labels)

標籤和跳躍可以用來控制方塊圖的執行。跳躍和標籤符號的右邊不可以連接任何物件。使用下面的記號：

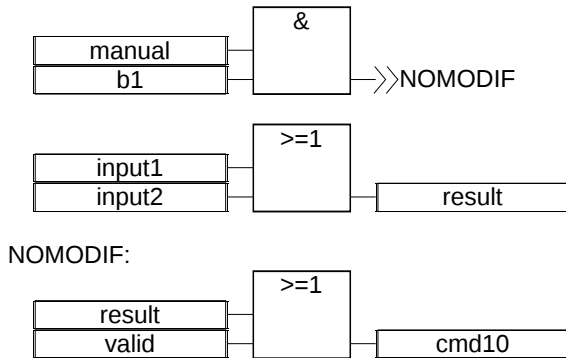
>>LAB 跳躍至一個標籤 (標籤名稱是 “LAB”)

LAB: 標籤的定義 (標籤名稱是 “LAB”)

假如跳躍符號**左邊**連接線的布林狀態是**真 (TRUE)** 時，程式就會直接跳躍到對應的標籤符號程式中執行。

(* 使用標籤和跳躍的 FBD 程式 *)

語言參考



(* IL 相等式 : *)

```
ld  manual
   and    b1
   jmpc   NOMODIF
ld  input1
or  input2
st  result
NOMODIF: ld  result
or  valid
st  cmd10
```

B.5.4 布林反相 (Boolean negation)

連接於功能方塊輸入的連接線可以是**布林反相**。反相是用一個小圓圈來表示。當使用布林反相時，連接線的左右兩端必須具有**布林**型態。

(* 使用布林反相的FBD 程式 *)



(* ST 相等式 : *)

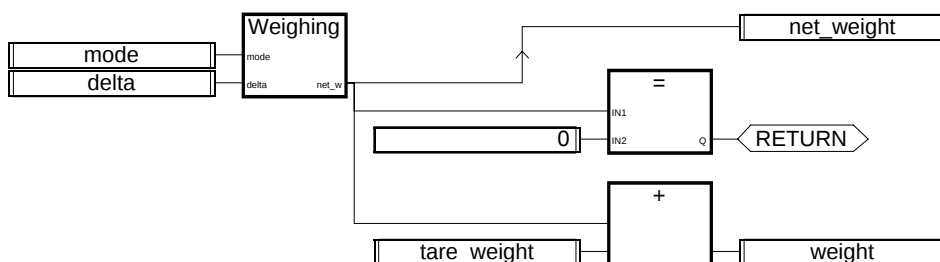
```
output1 := input1 AND NOT (input2);
```

B.5.5 在 FBD 程式中呼叫函式或功能方塊

FBD 語言可以呼叫副程式、函式或功能方塊。而副程式、函式或功能方塊是以函式方塊來表示，名稱標示於方塊中。

若呼叫的是副程式或函式，回傳值就是函式方塊唯一的輸出。若呼叫的是功能方塊，函式方塊的輸出就不只一個了。

(* 使用副程式方塊的 FBD 程式 *)



(* ST 相等式 : *)

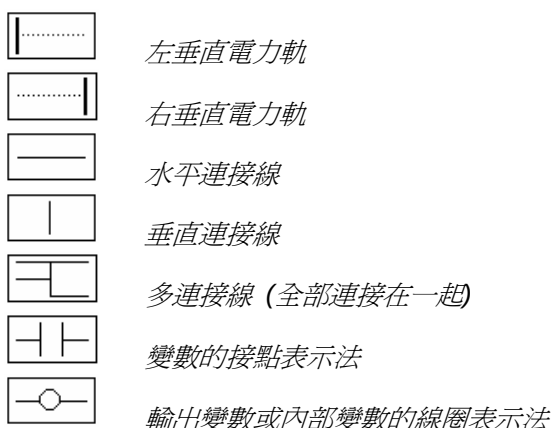
`net_weight := Weighing (mode, delta);` (* 呼叫副程式 *)

`If (net_weight = 0) Then Return; End_if;`

`weight := net_weight + tare_weight;`

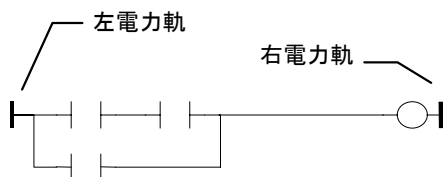
B.6 LD 階梯圖語言

階梯圖 (Ladder Diagram, LD) 是布林式子的圖形表示法，它將**接點** (contact, 輸入參數) 和**線圈** (coil, 輸出結果) 結合在一起使用。LD 語言藉著將**圖形符號**放入程式圖中做測試敘述和**布林**資料的修改。就像畫一張電力接點圖 (electric contact diagram) 一樣，階梯圖是由 LD 圖形符號組織起來的。LD 階梯圖的左邊和右邊都與垂直**電力軌** (power rail) 連接。LD 階梯圖的基本圖形元件如：

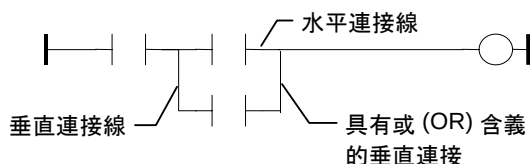


B.6.1 電力軌和連接線

LD 階梯圖的左右邊界是垂直線，名稱分別是**左電力軌**和**右電力軌**。



LD 階梯圖的圖形符號是以**連接線**與電力軌或其他符號連接起來。連接線有水平和垂直兩種。



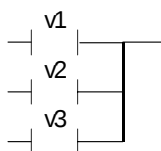
每一線段都具有布林狀態，值為**真 (TRUE)** 或**假 (FALSE)**。直接連結在一起的所有線段的布林狀態是一樣的。連接至左**垂直電力軌**的任何水平線的狀態是**真 (TRUE)**。

B.6.2 多連接

水平連接線的左端和右端有相同的布林狀態。將水平和垂直連接線結合在一起使用就可以製造出**多連接**。多連接端點的布林狀態遵循邏輯規則。

左多連接是一條垂直線的左邊連接了不止一條的水平線，而且右邊連接了一條水平線。則此多連接右端的布林狀態是左端所有布林狀態的**邏輯或 (LOGICAL OR)** 運算值。

(* 左多連接範例 *)

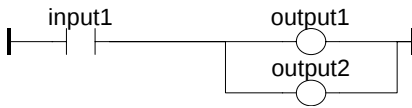


(* 右端狀態是 (v1 OR v2 OR v3) *)

右多連接是一條垂直線的左邊連接了一條水平線，而且右邊連接了不止一條的水平線。則此多連接左端的布林狀態會傳送給每一個右端。

(* 右多連接範例 *)

語言參考



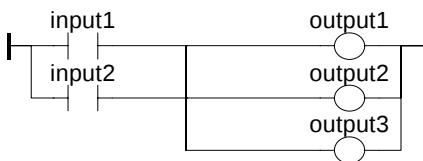
(* ST 相等式 : *)

output1 := input1;

output2 := input1;

左右多連接是一條垂直線的左邊連接了不止一條的水平線，而且右邊也連接了不止一條的水平線。則此多連接的每一個右端的布林狀態是左端所有布林狀態的邏輯或 (LOGICAL OR) 運算值。

(* 左右多連接範例 *)



(* ST 相等式 : *)

output1 := input1 OR input2;

output2 := input1 OR input2;

output3 := input1 OR input2;

B.6.3 基本的LD 接點和線圈

對於輸入接點，可以使用下列的符號：

直接接點 (Direct contact)

反相接點 (Inverted contact)

邊緣偵測接點 (Contacts with edge detection)

對於輸出線圈，可以使用下列的符號：

直接線圈 (Direct coil)

反相線圈 (Inverted coil)

設定線圈 (SET coil)

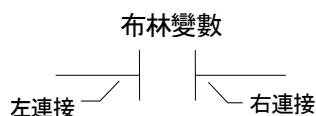
重置線圈 (RESET coil)

邊緣偵測線圈 (Coils with edge detection)

變數名稱會標示在任何圖形符號的上面：

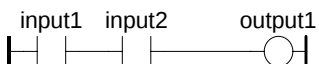
直接接點

直接接點可以將**連接線**的狀態與布林**變數值**做**布林運算**。



接點的右連接線狀態是左連接線狀態與接點上布林變數值的**邏輯且** (LOGICAL AND)。

(* 使用直接接點的範例 *)

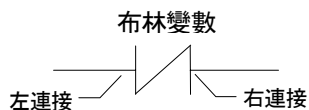


(* ST 相等式 : *)

`output1 := input1 AND input2;`

反相接點

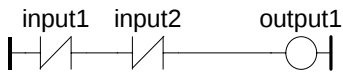
反相接點可以將**連接線**的狀態與布林**變數值**的布林反相做**布林運算**。



接點的右連接線狀態是左連接線狀態與接點上布林變數值的**布林反相的邏輯且** (LOGICAL AND)。

語言參考

(* 使用反相接點的範例 *)

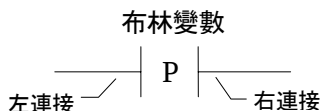


(* ST 相等式 : *)

`output1 := NOT (input1) AND NOT (input2);`

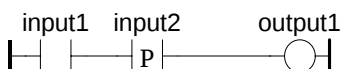
上升邊緣偵測接點

此接點 (正的) 可以將連接線的狀態與布林變數的上升邊緣值做布林運算。



當接點左邊連接線的狀態是**真 (TRUE)**，而且接點上的布林變數狀態正好由假 (FALSE) 變成**真 (TRUE)** 時，接點右邊連接線的狀態就設定成真 (TRUE)。在任何其他的情況下，接點右邊連接線的狀態會重新設定成假 (FALSE)。

(* 使用上升邊緣偵測接點的範例 *)



(* ST 相等式 : *)

`output1 := input1 AND (input2 AND NOT (input2prev));`

(* input2prev 是 input2 在前一個週期時的值 *)

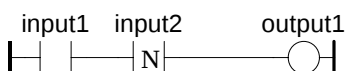
下降邊緣偵測接點

此接點 (負的) 可以將連接線的狀態與布林變數的下降邊緣值做布林運算。



當接點左邊連接線的狀態是**真 (TRUE)**，而且接點上的布林變數狀態正好由**真 (TRUE)** 變成假 (FALSE) 時，接點右邊連接線的狀態就設定成**真 (TRUE)**。在任何其他的情況下，接點右邊連接線的狀態會重新設定成假 (FALSE)。

(* 使用下降邊緣接點的範例 *)



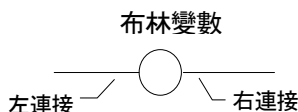
(* ST 相等式 : *)

`output1 := input1 AND (NOT (input2) AND input2prev);`

(* input2prev 是 input2 在前一個週期時的值 *)

直接線圈

直接線圈可以將**連接線**的布林狀態做**布林輸出**。



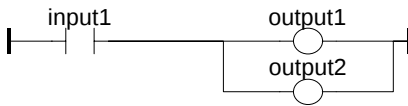
線圈上的變數值是**左連接線的布林狀態**。左連接線的狀態會傳送給右連接線。右連接線可以連接至右垂直電力軌。

線圈上的變數必須是**輸出變數**或**內部變數**。

線圈上的變數名稱可以是程式的名稱 (只適用於**副程式**)。這就相當於將副程式的回傳值指定給變數。

(* 使用直接線圈的範例 *)

語言參考



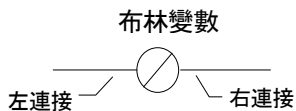
(* ST 相等式 : *)

```
output1 := input1;
```

```
output2 := input1;
```

反相線圈

反相線圈會根據**連接線**狀態的布林**反相**做**布林輸出**。

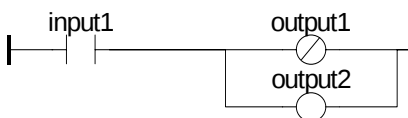


線圈上的變數值是**左連接線**狀態的布林**反相**。左連接線的狀態會傳送給右連接線。右連接線可以連接至右垂直電力軌。

線圈上的變數必須是**輸出變數**或**內部變數**。

線圈上的變數名稱可以是程式的名稱 (只適用於**副程式**)，這就相當於將副程式的回傳值指定給變數。

(* 使用反相線圈的範例 *)



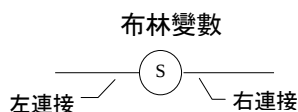
(* ST 相等式 : *)

```
output1 := NOT (input1);
```

```
output2 := input1;
```

設定線圈

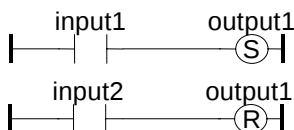
“設定” 線圈可以將**連接線**的布林狀態做**布林輸出**。



當左連接線的布林狀態變成真 (TRUE) 時，線圈上的變數值就設定成真 (TRUE)。在遇到“重置”線圈的反相規則之前，此輸出變數會一直維持這個數值。左連接線的狀態會傳送給右連接線。右連接線可以連接至右垂直電力軌。

線圈上的變數必須是輸出變數或內部變數。

(* 使用“設定”和“重置”線圈的範例 *)

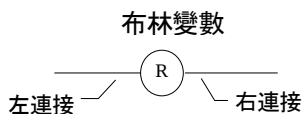


(* ST 相等式 : *)

```
IF input1 THEN
output1 := TRUE;
END_IF;
IF input2 THEN
output1 := FALSE;
END_IF;
```

重置線圈

“重置”線圈可以將連接線的布林狀態做布林輸出。

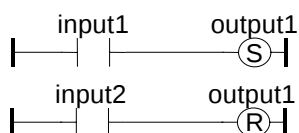


語言參考

當左連接線的布林狀態變成真 (TRUE) 時，線圈上的變數值就重新設定成假 (FALSE)。在遇到“設定”線圈的反相規則之前，此輸出變數會一直維持這個數值。左連接線的狀態會傳送給右連接線。右連接線可以連接至右垂直電力軌。

線圈上的變數必須是輸出變數或內部變數。

(* 使用“設定”和“重置”線圈的範例 *)

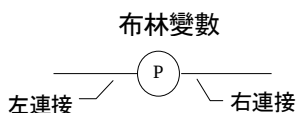


(* ST 相等式 : *)

```
IF input1 THEN
output1 := TRUE;
END_IF;
IF input2 THEN
output1 := FALSE;
END_IF;
```

□ 上升邊緣偵測線圈

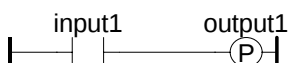
“正”線圈可以將連接線的布林狀態做布林輸出。這種型態的線圈只能在 Quick LD 編輯器中使用。



當左連接線的布林狀態由假 (FALSE) 變成真 (TRUE) 時，線圈上的變數值就設定成真 (TRUE)。在所有其他的情況時，線圈上的變數值會重新設定成假 (FALSE)。左連接線的狀態會傳送給右連接線。右連接線可以連接至右垂直電力軌。

線圈上的變數必須是**輸出變數**或**內部變數**。

(* 使用“正”線圈的範例 *)



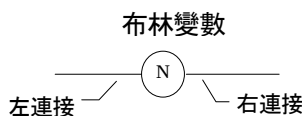
(* ST 相等式 : *)

```

IF (input1 and NOT(input1prev)) THEN
output1 := TRUE;
ELSE
output1 := FALSE;
END_IF;
(* input1prev 是 input1 在前一個週期時的值 *)
  
```

▣ 下降邊緣偵測線圈

“負”線圈可以將**連接線**的布林狀態做**布林輸出**。這種型態的線圈只能在 Quick LD 編輯器中使用。

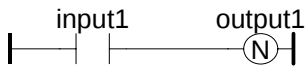


當**左連接線**的布林狀態由真 (TRUE) 變成假 (FALSE) 時，線圈上的變數值就設定成**真 (TRUE)**。在所有其他的情況時，線圈上的變數值會重新設定成假 (FALSE)。左連接線的狀態會傳送給右連接線。右連接線可以連接至右垂直電力軌。

線圈上的布林變數必須是**輸出變數**或**內部變數**。

(* 使用“負”線圈的範例 *)

語言參考



(* ST 相等式 : *)

IF (NOT(input1) and input1prev) THEN

output1 := TRUE;

ELSE

output1 := FALSE;

END_IF;

(* input1prev 是 input1 在前一個週期時的值 *)

B.6.4 跳回 (RETURN) 敘述

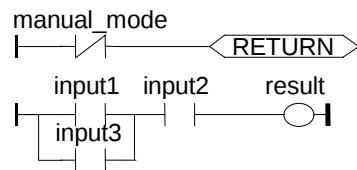
RETURN 標籤可以做為程式有條件終止時的輸出。**RETURN** 符號的右邊不可以連接任何東西。



假如 **RETURN** 符號的左連接線布林狀態為**真** (TRUE) 時，程式就會直接結束，不會再執行程式其他未執行的部分。

注意：當 LD 程式是一個副程式時，必須有一個輸出線圈的變數名稱與副程式名稱相同，以作為此副程式的回傳值（傳給呼叫程式）。

(* 使用 RETURN 符號的範例 *)



(* ST 相等式 : *)

If Not (manual_mode) Then RETURN; End_if;

result := (input1 OR input3) AND input2;

B.6.5 跳躍 (Jumps) 和標籤 (labels)

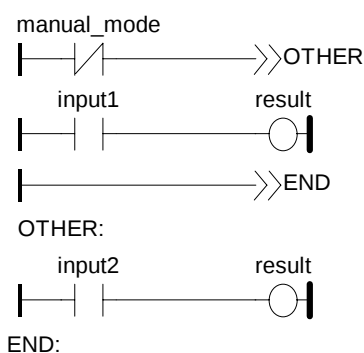
標籤、條件和非條件跳躍符號，可以用來控制階梯圖的執行。標籤和跳躍符號的右側不可以連接任何東西。使用下面的符號：

>>LAB 跳躍至標籤為“LAB”的地方

LAB: 名稱為“LAB”的標籤的定義

假如跳躍符號左連接線的布林狀態為真 (TRUE) 時，程式就會跳躍至相對應的標籤程式去執行。

(* 使用跳躍和標籤符號的範例 *)



(* IL 相等式 : *)

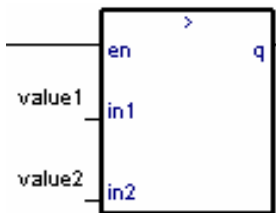
	<i>ldn</i>	<i>manual_mode</i>
	<i>jmpc</i>	<i>other</i>
	<i>ld</i>	<i>input1</i>
	<i>st</i>	<i>result</i>
	<i>jmp</i>	<i>END</i>
OTHER:	<i>ld</i>	<i>input2</i>
	<i>st</i>	<i>result</i>
END:	(* 程式結束 *)	

B.6.6 在LD 中使用方塊

使用 Quick LD 編輯器時，使用者必須將函式方塊連接在布林狀態的連接線上。事實上，函式方塊中可以是運算、功能方塊或函式。假如在階梯圖中插入沒有布林輸入和（或）布林輸出的方塊時，編輯器會自動為方塊加上 EN、ENO 參數。假如使用者使用的是 FBD 或 LD 編輯器，而且連接至方塊的變數是方塊所需的型態，那麼編輯器就不會在方塊上加上 EN 和 ENO 參數。

□ “EN” 輸入

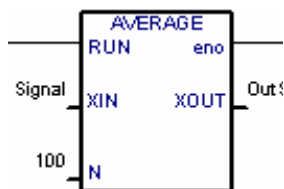
在運算、函式或功能方塊中，它的第一筆輸入不具有布林資料型態。因為第一筆輸入總是必須連接在階梯圖的導軌上，所以編輯器就會自動將另一個輸入插在方塊輸入的第一個位置，名稱爲“EN”。只有當輸入值 EN 是真 (TRUE) 時，方塊中的運算才會執行。下面是一個比較大小的運算和等價的 ST 程式例子：



```
IF rung_state THEN
    q := (value1 > value2);
ELSE
    q := FALSE;
END_IF;
(* q 的狀態會傳送給方塊右邊的導軌 *)
```

□ “ENO” 輸出

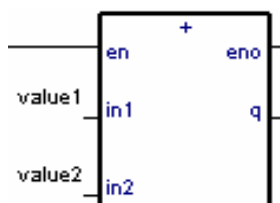
在一些運算、函式或功能方塊中，它的第一筆輸出不具有布林資料型態。因為第一筆輸出總是必須連接在階梯圖的導軌上，所以編輯器會自動將另一個輸出插在方塊輸出的第一個位置，名稱爲“ENO”。輸出值 ENO 總是與方塊的第一筆輸入值有相同狀態。下面是使用平均值功能方塊的範例，和等價的 ST 程式：



```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* eno 的狀態會傳送給方塊右邊的導軌
*)
```

“EN” 和 “ENO” 參數

在某些情形，同時需要使用 **EN** 和 **ENO**。下面是一個算數運算的範例，和等價的 ST 程式：



```
IF rung_state THEN
    result := (value1 + value2);
END_IF;
eno := rung_state;
(* eno 的狀態會傳送給方塊右邊的導軌
*)
```

B.7 ST (結構化文字) 語言

ST (Structured Text) 是一種為自動化流程所設計的高階結構化語言。此語言主要是用來執行無法用繪圖式語言簡單表示出來的複雜程序。在 **SFC** 語言中，步驟的行為描述和轉移條件的判斷條件是以 **ST** 語言為內定語言。

B.7.1 ST 主要語法

ST 程式是一連串的 **ST 敘述**，每一個敘述以一個分號 (“;”) 做為結尾。在 **ST** 程式原始碼中所使用的名稱 (變數識別字、常數、語言的關鍵字...) 是以 **無作用分隔符號** (空白字元、列結束或跳位符號) 或是以已定義好含意的 **有作用分隔符號** (例如，“>” 分隔符號是做 “大於” 比較的意思) 來分隔開。文字中可以自由的加上註解，而註解必須啓始於 “(*)”，且終止於 “(*)”。每一個敘述終結於分號 (“;”) 字元。**ST** 敘述的基本形式如下：

指定敘述 (變數 := 表示式)

副程式或函數呼叫

功能方塊呼叫

選擇敘述 (IF, THEN, ELSE, CASE...)

遞迴敘述 (FOR, WHILE, REPEAT...)

控制敘述 (RETURN, EXIT...)

與其他語言 (例如 **SFC**) 連結的特別敘述

無作用分隔符號可以自由的用在有作用分隔符號、常數型態和識別字之間。

ST 的無作用分隔符號是：空白 (space) 字元、跳位 (tabs) 符號和列結束字元。不像每一列有固定格式的語言 (例如 **IL**)，列結束可以在程式的任何地方使用。當使用無作用分隔符號來增加 **ST** 程式的可讀性時，應遵循下面的規則：

一列只寫一個敘述

使用跳位符號做為複雜敘述的縮排方式

加上註解來增加每一行和每一個段落的可讀性

B.7.2 表示式和括弧

ST 表示式是由 ST **運算符號**和變數或常數**運算子**結合起來的。對於每一個單一的表示式 (用 ST 運算符號將運算子結合起來)，所有運算子的**型態**必須相同。單一表示式的型態會與它的運算子型態相同，而且可以被用在更複雜的表示式中。例如：

(boo_var1 AND boo_var2) 具有布林型態

not (boo_var1) 具有布林型態

(sin (3.14) + 0.72) 具有實數類比型態

(t#1s23 + 1.78) 是一個無效的表示式

括弧可以用來隔離表示式的子部分，並且明確的整理出它們操作的優先順序。當一個複雜的表示式中沒有任何括弧時，操作順序是以 ST 運算符號之間的內定**優先順序**為準。例如：

$2 + 3 * 6$ 等於 $2+18=20$ 因為相乘運算符號有較高的優先順序

$(2+3) * 6$ 等於 $5*6=30$ 括弧有較高的優先順序

警告：在一個表示式中，槽狀括弧的最大數目是 **8** 階層的括弧。

B.7.3 函式或功能方塊呼叫

標準的 ST 函式呼叫可以使用在下面的每一個物件中：

副程式

程式庫函式和用 IEC 語言撰寫的功能方塊

“C” 函式和功能方塊

型態轉換函式

□ 呼叫副程式或函數

名稱： 被呼叫的副程式的名稱

或是用 IEC 語言或 “C” 語言撰寫的程式庫函數

含意： 呼叫一個 ST、IL、LD 或 FBD 副程式或函數或一個 “C” 函數，並且得到它的回傳值

語法： <變數> := <副程式> (<參數 1>, ... <參數 N>);

運算子： 回傳值的型態和呼叫參數必須遵循副程式的定義

回傳值： 由副程式傳回來的值

副程式呼叫可以使用在任何的表示式中，也可以使用在 SFC 的轉移條件中。

範例一：副程式呼叫

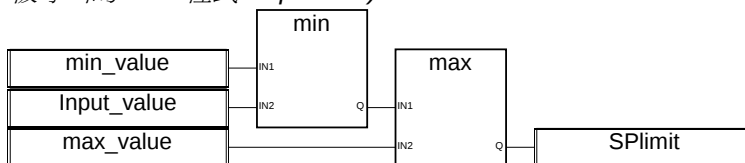
(* ST 主程式 *)

(* 得到一個類比數值並且將它轉換成一個有限制的時間數值 *)

```
ana_timeprog := SPLimit ( tprog_cmd );
```

```
appl_timer := tmr (ana_timeprog * 100);
```

(* 被呼叫的 FBD 程式 ‘SPlimit’ *)



範例二：函數呼叫

(* 在複雜的表示式中使用標準的 “C” 函數: min, max, right, mlen 和 left *)

```
limited_value := min(16, max (0, input_value) );
```

```
rol_msg := right (message, mlen (message) – 1) + left (message, 1);
```

□ 呼叫功能方塊

名稱： 功能方塊樣例的名稱

含意： 從 ISaGRAF 程式庫或從使用者的程式庫呼叫一個功能方塊，並且使用它的回傳參數

語法： (* 呼叫功能方塊 *)

<方塊名稱> (<參數 1>, <參數 2> ...);

(* 得到它的回傳參數 *)

<結果> := <方塊名稱>. <回傳參數 1>;

...

<結果> := <方塊名稱>. <回傳參數 N>;

運算子： 參數是配合功能方塊指明的參數型態的表示式

回傳值： 參考語法部份以得到回傳參數

請查閱 ISaGRAF library，以便瞭解功能方塊中每一個參數的含意和型態。

功能方塊樣例 (功能方塊的複製名稱) 必須在字典中宣告。

範例：

(* 呼叫功能方塊的 ST 程式 *)

(* 在字典宣告方塊樣例 : *)

(* trigb1 : block R_TRIG – 上升邊界偵測 *)

(* 用 ST 語言執行功能方塊 *)

trigb1 (b1);

(* 使用回傳參數的 *)

If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;

B.7.4 ST 的特定布林運算符號

下面是 ST 語言的特定布林運算符號：

-REDGE 上升邊緣偵測

-FEDGE 下降邊緣偵測

語言參考

ST 語言可以使用其他的標準布林運算符號，如下：

- NOT 布林反相
- AND (&) 邏輯和
- OR 邏輯或
- XOR 邏輯互斥或

這些運算符號的相關說明，請參考‘標準運算符號，功能方塊和函式’章節。

▢ “REDGE” 運算

名稱： **REDGE**

含意： 求得完整布林表示式的上升邊緣

語法： <邊緣> := REDGE (<布林表示式>,<記憶體變數>);

運算子： 第一個運算子是任何的布林變數或複雜的表示式

 第二個運算子是一個用來儲存表示式最後狀態的內部布林變數

回傳值： 當表示式從假 (FALSE) 變成真 (TRUE) 時，回傳值為真 (TRUE)。而其他所有的狀況，回傳值皆為假 (FALSE)。

使用 REDGE 運算符號時，在相同的執行週期中，一個表示式的上升邊緣只能偵測到一次。REDGE 運算符號可以用來描述 SFC 轉移條件的判斷條件。

警告：用來儲存表示式最後狀態的“記憶體”布林變數不可以用來做為不同表示式的邊緣觸發器。

假如想在 REDGE 表示式中使用名稱爲“xxx”的布林變數，必須先宣告一個名稱爲“EDGE_xxx”的獨特內部變數，然後在 REDGE 表示式中使用 EDGE_xxx 變數。這個方式可以使得 REDGE 在估值期間，不會覆蓋掉其它的記憶體變數。

範例：

(* 使用 REDGE 運算符號的 ST 程式 *)

(* 這個程式是用來計數布林輸入的上升邊緣 *)
 (* Bi120 是一個輸入布林變數 *)
 (* Edge_Bi120 是用來儲存 Bi120 變數的狀態 *)

```
If REDGE (Bi120, Edge_Bi120) Then
Counter := Counter + 1;
End_if;
```

注意:IEC1131-1 標準沒有包括 REDGE 運算符號。使用者可以使用 R_TRIG 標準方塊。REDGE 運算符號是因為相容性的理由而被保留下來的。

□ “FEDGE” 運算符號

名稱： FEDGE

含意： 求得布林表示式的下降邊緣

語法： <邊緣> := FEDGE (<布林表示式>,<記憶體變數>);

運算子： 第一個運算子是任何的布林變數或複雜的表示式
 第二個運算子是一個用來儲存表示式最後狀態的內部布林變數

回傳值： 當表示式從真 (TRUE) 變成假 (FALSE) 時，回傳值為真 (TRUE)。而其他所有的狀況，回傳值皆為假 (FALSE)。

使用 FEDGE 運算符號時，在相同的執行週期中，一個表示式的下降邊緣只能偵測到一次。FEDGE 運算符號可以用來描述 SFC 轉移條件的判斷條件。

警告：用來儲存表示式最後狀態的“記憶體”布林變數不可以用來做為不同表示式的邊緣觸發器。

假如想在 FEDGE 表示式中使用名稱爲“xxx”的布林變數，必須先宣告一個名稱爲“**EDGE_xxx**”的獨特內部變數，然後在 FEDGE 表示式中使用 EDGE_xxx 變數。這個方式可以使得 REDGE 在估值期間，不會覆蓋掉其它的記憶體變數。

範例：

語言參考

(* 使用 FEDGE 運算符號的 ST 程式 *)

(* 這個程式是用來計數一個布林輸入的下降邊緣 *)

(* Bi120 是一個輸入布林變數 *)

(* Edge_Bi120 是用來儲存 Bi120 變數的狀態 *)

If FEDGE (Bi120, Edge_Bi120) Then

Counter := Counter + 1;

End_if;

注意:IEC1131-1 標準沒有包括 FEDGE 運算符號,使用者可以使用 F_TRIG 標準方塊。FEDGE 運算符號是因為相容性的理由而被保留下來的。

B.7.5 ST 基本敘述

下面是 ST 語言的基本敘述：

指定

RETURN 敘述

IF-THEN-ELSIF-ELSE 結構

CASE 敘述

WHILE 反覆敘述

REPEAT 反覆敘述

FOR 反覆敘述

EXIT 敘述

▢ 指定

名稱： :=

含意： 將表示式結果指定給變數

語法： <變數> := <任何表示式>;

運算子： 變數必須是內部變數或輸出變數

變數和表示式必須具有相同的型態

表示式可以是副程式呼叫或是從 ISaGRAF 程式庫中呼叫函式的敘述。

範例：

(* 使用指定敘述的 ST 程式 *)

(* 變數 <=< 變數 *)

bo23 := bo10;

(* 變數 <=< 表示式 *)

bo56 := bx34 OR alm100 & (level >= over_value);

result := (100 * input_value) / scale;

(* 指定為副程式的回傳值 *)

rc := PSelect ();

(* 指定為函式呼叫 *)

limited_value := min (16, max(0, input_value));

▣ 跳回 (RETURN) 敘述

名稱： **RETURN**

含意： 終止目前程式執行

語法： **RETURN ;**

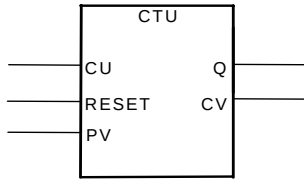
運算子： 無

在 SFC 行為方塊中，RETURN 敘述僅表示此方塊執行結束。

範例：

(* 以 FBD 規格撰寫的程式：可程式計數器 *)

語言參考



(* 使用 ST 的 RETURN 敘述來撰寫此程式 *)

```
If not(CU) then
Q := false;
  CV := 0;
  RETURN; (* 終止程式 *)
end_if;
```

```
if R then
CV := 0;
else
if (CV < PV) then
  CV := CV + 1;
end_if;
end_if;
Q := (CV >= PV);
```

□ IF-THEN-ELSIF-ELSE 敘述

名稱： IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF

含意： 選擇其中一組 ST 敘述來執行

選擇是根據布林表示式的數值決定的

語法： IF <布林表示式> THEN

<敘述>;

<敘述>;

...

ELSIF <布林表示式> THEN

<敘述>;

<敘述>;

```
...  
    ELSE  
    <敘述>;  
    <敘述>;  
...  
END_IF;
```

ELSE 和 ELSIF 敘述是選擇性的。假如程式中沒有 ELSE 敘述，則當判斷條件是假 (FALSE) 的時候，不會執行任何指令。

範例：

(* 使用 IF 敘述的 ST 程式 *)

```
IF manual AND not (alarm) THEN  
    level := manual_level;  
    bx126 := bi12 OR bi45;  
ELSIF over_mode THEN  
    level := max_level;  
ELSE  
    level := (lv16 * 100) / scale;  
END_IF;
```

(* 沒有 ELSE 的 IF 結構 *)

```
If overflow THEN  
    alarm_level := true;  
END_IF;
```

⇨ CASE 敘述

名稱： **CASE ... OF ... ELSE ... END_CASE**

含意： 選擇其中一組 ST 敘述來執行
 選擇是根據整數表示式決定的

語言參考

語法： **CASE** <整數表示式> **OF**

 <數值> : <敘述>;

 <數值> , <數值> : <敘述>;

 ...

ELSE

 <敘述>;

END_CASE;

CASE 的數值必須是整數常數形態。以逗點分開的一些數值可以使用相同的一組敘述。**ELSE** 敘述是選擇性的。

範例：

(* 使用 **CASE** 敘述的 ST 程式 *)

```
CASE error_code OF
255:   err_msg := 'Division by zero';
       fatal_error := TRUE;
1:    err_msg := 'Overflow';
2, 3: err_msg := 'Bad sign';
ELSE
err_msg := 'Unknown error';
END_CASE;
```

▣ **WHILE** 敘述

名稱： **WHILE ... DO ... END_WHILE**

含意： 一組 ST 敘述的遞迴結構

 做任何遞迴動作之前，會先評估是否“繼續”遞迴的判斷條件

語法： **WHILE** <布林表示式> **DO**

 <敘述>;

 <敘述>;

 ...

END_WHILE;

警告：因為 ISaGRAF 是一個**同步**系統，所以在 WHILE 的反覆執行期間，輸入變數不會被更新。輸入變數狀態的改變不可用來描述一個 WHILE 敘述的判斷條件。

範例：

(* 使用 WHILE 敘述的 ST 程式 *)

(* 這個程式使用了特殊的 “C” 函式在一個序列埠上讀取字串 *)

string := ‘’; (* 空字串 *)

nbchar := 0;

WHILE ((nbchar < 16) & ComIsReady ()) DO

string := string + ComGetChar ();

nbchar := nbchar + 1;

END_WHILE;

▣ REPEAT 敘述

名稱： REPEAT ... UNTIL ... END_REPEAT

含意： 一組 ST 敘述的遞迴結構

做任何遞迴動作之後，會評估是否“繼續”遞迴的判斷條件

語法： REPEAT

<敘述>;

<敘述>;

...

UNTIL <布林判斷條件>

END_REPEAT;

語言參考

警告：因為 ISaGRAF 是一個**同步**系統，所以在 REPEAT 的反覆執行期間，輸入變數不會被更新。輸入變數狀態的改變不可用來描述一個 REPEAT 敘述的結束判斷條件。

範例：

(* 使用 REPEAT 敘述的 ST 程式 *)

(* 這個程式使用了特殊的“C”函式在一個序列埠上讀取字串 *)

```
string := ''; (* 空字串 *)
nbchar := 0;
IF ComIsReady ( ) THEN
REPEAT
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
    UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
END_REPEAT;
END_IF;
```

FOR 敘述

名稱： FOR ... TO ... BY ... DO ... END_FOR

含意： 使用整數類比索引變數來執行一個限制數目的遞迴動作

語法： FOR <索引> := <最小值> TO <最大值> BY <增量> DO
 <敘述>;
 <敘述>;
END_FOR;

運算子： 索引： 在任何迴路中遞增的內部類比變數

最小值： 索引的初始值 (第一個迴路之前)

最大值： 索引的最大允許值

增量： 每一個迴路的索引增量

[BY 增量] 敘述是選擇性的，假如沒有特別指定，增量為 1。

警告：因為 ISaGRAF 是一個**同步**系統，所以在 FOR 的遞迴執行期間，輸入變數不會被更新。

下面是用 “while” 語法來表達 FOR 敘述：

```
索引 := 最小值;
while (索引 <= 最大值) do
    <敘述>;
    <敘述>;
    索引 := 索引 + 增量;
end_while;
```

範例：

```
(* 使用 FOR 敘述的 ST 程式 *)
(* 這個程式用來抽取字串的數位字元 *)
```

```
length := mlen (message);
target := ''; (* 空字串 *)
FOR index := 1 TO length BY 1 DO
code := ascii (message, index);
    IF (code >= 48) & (code <= 57) THEN
        target := target + char (code);
    END_IF;
END_FOR;
```

⇨ 離開 (EXIT) 敘述

名稱：EXIT

含意：跳離 FOR、WHILE 或 REPEAT 的遞迴敘述

語法：EXIT;

語言參考

運算子： 無

EXIT 通常用在 IF 敘述中，或是一個 FOR、WHILE 或 REPEAT 區塊中。

範例：

(* 使用 EXIT 敘述的 ST 程式 *)

(* 這個程式會從字串中尋找一個字元 *)

```
length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
IF (code = searched_char) THEN
    found := YES;
EXIT;
END_IF;
END_FOR;
```

B.7.6 ST 延伸函數

下面是 ST 語言的延伸函式：

- TSTART - TSTOP：計時器控制

下面的敘述和函式可以用來控制 SFC 子程式的執行。它們可以用在 SFC 步驟的 ACTION(): ... END_ACTION; 區塊中。

- GSTART 開始 SFC 程式
- GKILL 刪除 SFC 程式
- GFREEZE 冰凍 SFC 程式
- GRST 重新開始被冰凍的 SFC 程式
- GSTATUS 得到 SFC 程式的目前執行狀態

警告： IEC 1131-3 標準不包括以上這些函式。

撰寫 SFC 步驟時，使用者可以發現 GSTART 和 GKILL 的等價語法，如下所示：

`child_name(S); (* 與 GSTART(child_name);意思等價 *)`

`child_name(R); (* 與 GKILL(child_name);意思等價 *)`

下面的欄位可以用來存取 SFC 步驟的狀態：

GSnnn.x 用來表示步驟動作狀態的布林值

GSnnn.t 從步驟最後活動到目前的時間

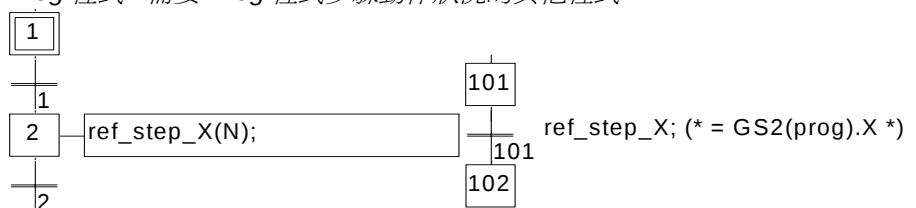
(“nnn” 是 SFC 步驟的參考號碼)

藉著使用下面的語法，使用者也有可能測試另一個 SFC 程式中的步驟動作情形：

GSnnn(程式名稱).x

警告： 參考另一個程式的步驟的語法並沒有包括在 IEC 1131-3 標準裡。說到 IEC 規則，一個簡單方式可以達到相同效果，就是在字典裡宣告一個全域的布林變數來表示被測試的步驟的活動情形 (例如 `ref_step_X`)。然後使用者在步驟中插入這個變數的 N 修飾字 (譬如：`ref_step_X(N);`)。則在測試步驟動作狀態的程式中，使用者就可以使用這個變數。

Prog 程式 需要 Prog 程式步驟動作狀況的其他程式



▢ TSTART 敘述

名稱： TSTART

含意： 開始計時器變數計時

TSTART 指令不會修改時間數值，換言之，計時會從時間目前數值開始。

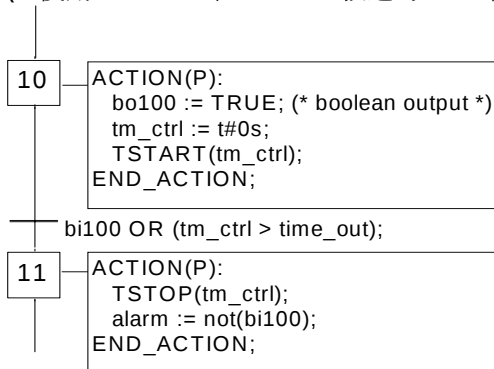
語法： TSTART (<計時器變數>);

運算子： 任何未動作的計時器變數

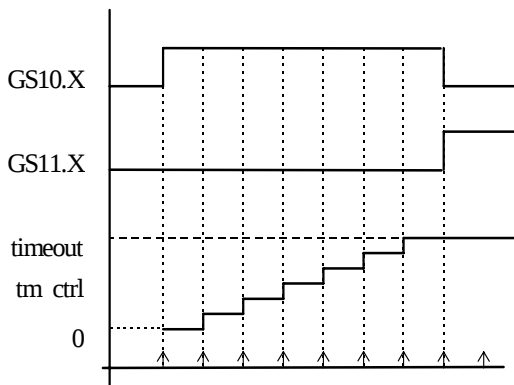
回傳值： 無

範例：

(* 使用 TSTART 和 TSTOP 敘述的 SFC 程式 *)



假如 bi100 總是假 (FALSE) 時，時間的變化圖如下：



在一個週期期間，時間一直保持相同的數值。

▢ TSTOP 敘述

名稱： TSTOP

含意： 停止更新一個計時器變數

TSTOP 指令不會修改時間數值

語法： TSTOP (<計時器變數>);

運算子： 任何正動作中的計時器變數

回傳值： 無

範例：請參考 TSTART 範例。

▣ GSTART 敘述

名稱： GSTART

含意： 開始一個 SFC 子程式，並且在每一個初始步驟放進一個權杖符號

語法： GSTART (<子程式>);

運算子： 敘述中的子程式必須是這個程式的子程式

回傳值： 無

GSTART 敘述不會自動啓始子程式的子程式。

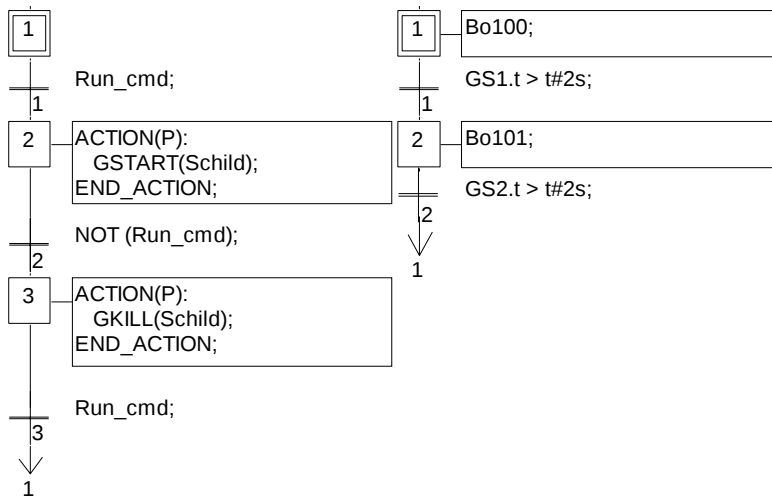
注意：因為 IEC 1131-3 標準並不包括 GSTART，所以可以使用 S 修飾字，
下面的語法啓始一個 SFC 子程式：

Child_name(S);

範例：使用 GSTART 和 GKILL

(* 'Sfather' 程序 *) (* 'Schild' 程序 *)

語言參考



＝ GKILL 敘述

名稱： **GKILL**

含意： 藉著移除存在於一個 SFC 子程式之目前步驟的權杖符號來終止此 SFC 子程式

語法： **GKILL (<子程式>);**

運算子： 敘述中的子程式必須是這個程式的子程式

回傳值： 無

GKILL 敘述會自動終止子程式的子程式。

注意：因為 IEC 1131-3 標準並不包括 **GKILL**，所以可以使用 **R** 修飾字，以下面的語法終止一個 SFC 子程式：

Child_name(R);

範例：請參考 **GSTART** 範例。

＝ GFREEZE 敘述

名稱： **GFREEZE**

含意： 冰凍 SFC 子程式執行。被冰凍的程式可以用 **GRST** 敘述來重新啓始。

語法： **GFREEZE (<子程式>);**

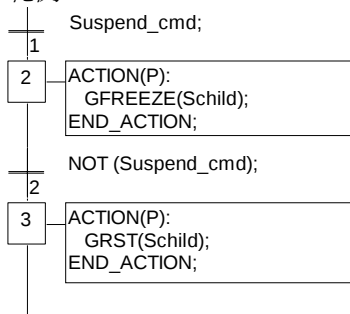
運算子： 敘述中的子程式必須是這個程式的子程式

回傳值： 無

GKILL 敘述會自動暫停子程式的子程式。

注意：IEC 1131-3 標準不包括 GFREEZE。

範例：



⇨ GRST 敘述

名稱： GRST

含意： 重新啓始被 GFREEZE 敘述冰凍的 SFC 子程式

語法： GRST (<子程式>);

運算子： 敘述中的子程式必須是這個程式的子程式

回傳值： 無

GRST 敘述會自動重新啓始子程式的子程式。

注意：IEC 1131-3 標準不包括 GRST。

範例：請參考 GFREEZE 範例。

⇨ GSTATUS 敘述

名稱： GSTATUS

含意： 傳回 SFC 程式目前的執行狀態

語法： <ana_var> := GSTATUS (<子程式>);

運算子： 敘述中的子程式必須是這個程式的子程式

回傳值： 0 = 程式是非執行的狀態 (終止的)

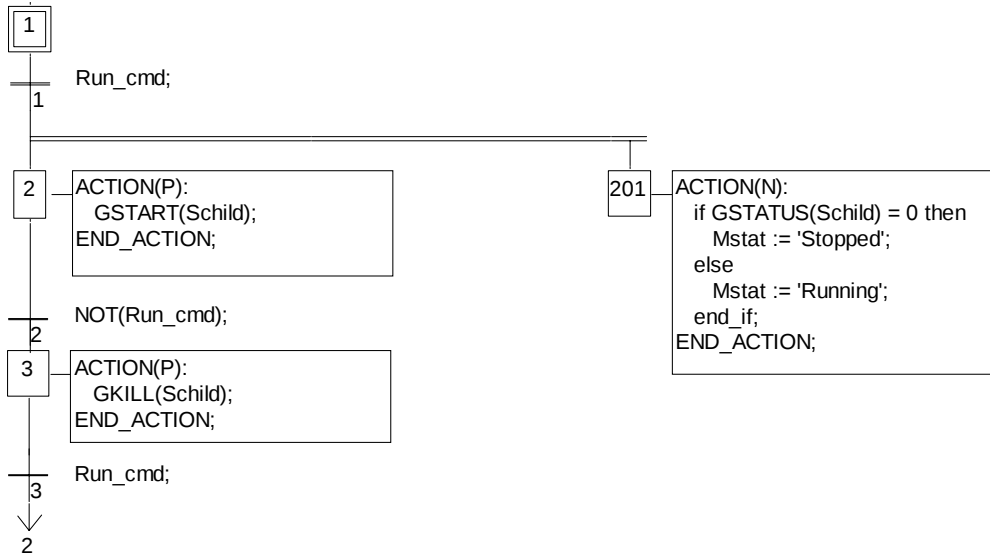
1 = 程式是正在執行的狀態 (啓始的)

語言參考

2 = 程式是被冰凍的

注意：IEC 1131-3 標準不包括 *GSTATUS*。

範例：



B.8 IL 指令集語言

IL (Instruction List，指令集語言) 是一種低階語言，它的指令一定關聯於目前結果 (或 IL 暫存器，IL register)。運算元顯示了目前數值與運算子之間的運算操作，而結果會再儲存在目前結果。

B.8.1 IL 主要語法

IL 程式是一連串的指令。每一個指令必須撰寫在每一列的開始，而且必須包含一個運算元 (對於比較特別的運算必須包含修飾字)、一個或多個運算子 (以逗號 ‘,’ 分隔開)。標籤及其冒號必須撰寫在指令前。假如指令有註解時，註解必須寫在一列的最後，而且必須啓始於 ‘(’，終止於 ‘)’。指令之間可以插入空的列。空的列可以加入註解。下面是指令列的範例：

```

標籤 運算符號 運算子  註解
Start: LD  IX1 (* 按下按鍵 *)
      ANDN  MX5 (* 指令沒有被禁止 *)
      ST   QX2 (* 馬達開始運轉 *)

```

▣ 標籤

標籤 (label) 後必須緊接一個冒號 ‘:’，並且放在指令之前。標籤可以放在空的列。標籤可以當成是一些運算元的運算子，例如跳躍。標籤名稱必須遵循下面規則：

不能超過 16 個字元

第一個字元必須是英文字母

除了名稱的第一個字元之外，其他的字元必須是英文字母、阿拉伯數字或底線 (‘_’) 字元

相同的 IL 程式中，一個名稱只能給一個標籤使用。標籤和變數的名稱可以相同。

□ 運算元的修飾字

下面列出一些運算元的修飾字 (modifier) 。有些運算元的名稱必須包含修飾字元，而且它們之間不包含空白字元：

N 運算子的布林反相

(延遲執行

C 有條件的執行

‘N’ 修飾字顯示運算子的布林反相。例如，ORN IX12 指令可以解釋成：
result := result OR NOT (IX12) 。

當指令遇見括弧 ‘(’ 修飾字時，就會被延遲執行；直到遇見關閉的 ‘)’ 括弧運算符號時，指令才會再繼續執行。

‘C’ 修飾字顯示相關的指令只有在目前結果是布林數值真 (TRUE，非布林型態時為非 0 數值) 時才必須執行。‘C’ 修飾字可以和 ‘N’ 修飾字結合起來，表示指令只有在目前結果是布林數值假 (FALSE，非布林型態時為 0) 時才必須執行。

□ 延遲運算

因為只有一個 IL 暫存器 (目前結果)，所以某一些操作必須被延遲，以便執行順序或指令可以被改變。括弧可以用來顯示延遲執行：

‘(’	是一個修飾字	表示被延遲的操作
)’	是一個運算元	執行被延遲的操作

開啓括弧 ‘(’ 修飾字顯示指令必須被延遲執行，直到遇見關閉括弧 ‘)’ 運算符號時才再繼續執行。例如下面的程序：

```

AND(    IX12
OR      IX35
)

```

可以解釋為：

```
result := result AND ( IX12 OR IX35 )
```

B.8.2 IL 運算元

下面表格集合了IL 語言標準的運算元：

運算元	修飾字	運算子	描述
LD	N	變數，常數	載入運算子
ST	N	變數	儲存目前結果
S		布林變數	設定為真
R		布林變數	重置為假
AND	N	布林	布林且 (boolean AND)
&	(	布林	布林且 (boolean AND)
OR	(	布林	布林或 (boolean OR)
XOR	(	布林	互斥或 (exclusive OR)

語言參考

AD	(	變數，常數	加
D	(	變數，常數	減
SU	(	變數，常數	乘
B	(	變數，常數	除
MU			
L			
DIV			
GT	(	變數，常數	測試：大於 (>)
GE	(	變數，常數	測試：大於等於 (>=)
EQ	(	變數，常數	測試：等於 (=)
LE	(	變數，常數	測試：小於等於 (<=)
LT	(	變數，常數	測試：小於 (<)
NE	(	變數，常數	測試：不等於 (<>)
CA	C	功能方塊的樣	呼叫功能方塊
L	N	例名稱	跳躍至標籤
JM	C	標籤	從副程式返回
P	N		
RE	C		
T	N		
)			執行延遲運算

在下一個部分，只有IL 語言的特殊運算符號會被描述，其他標準運算符號可以參考“標準運算符號、功能方塊和函式”部分。

LD 運算符號

運算： 載入一個數值給目前結果

允許的修飾字： N

運算子： 常數形態

內部、輸入或輸出變數

範例：

(* LD 運算範例 *)

```
LDex: LD false (* 目前結果 := 假布林常數 *)
LD true (* 目前結果 := 真布林常數 *)
LD 123 (* 目前結果 := 整數常數 *)
LD 123.1 (* 目前結果 := 實數常數 *)
LD t#3ms (* 目前結果 := 時間常數 *)
LD boo_var1 (* 目前結果 := 布林變數 *)
LD ana_var1 (* 目前結果 := 類比變數 *)
LD tmr_var1 (* 目前結果 := 計時器變數 *)
LDN boo_var2 (* 目前結果 := NOT ( 布林變數 ) *)
```

⇨ ST 運算元

運算： 將目前結果存入一個變數

此運算不會修改目前結果

允許的修飾字： N

運算子： 內部或輸出變數

範例：

(* ST 運算範例 *)

```
STboo: LD false
ST boo_var1 (* boo_var1 := FALSE *)
STN boo_var2 (* boo_var2 := TRUE *)
STana: LD 123
ST ana_var1 (* ana_var1 := 123 *)
STtmr: LD t#12s
ST tmr_var1 (* tmr_var1 := t#12s *)
```

⇨ S 運算元

運算： 假如目前結果是布林值真，就將布林值真存入一個布林變數。假

如目前結果是布林值假，不會執行任何運算。此運算不會修改目前結果。

允許的修飾字： 無

語言參考

運算子： 輸出或內部布林變數

範例：

```
(* S 運算範例 *)  
SETex: LD true (* 目前結果 := TRUE *)  
S boo_var1 (* boo_var1 := TRUE *)  
        (* 目前結果不會被修改 *)  
LD false (* 目前結果 := FALSE *)  
S boo_var1 (* 不會做任何事 – boo_var1 不改變 *)
```

⇨ R 運算符號

運算： 假如目前結果是布林值真，就將布林值假存入一個布林變數。假如目前結果是布林值假，不會執行任何運算。此運算不會修改目前結果。

允許的修飾字： 無

運算子： 輸出或內部布林變數

範例：

```
(* R 運算範例 *)  
RESETex: LD true (* 目前結果 := TRUE *)  
R boo_var1 (* boo_var1 := FALSE *)  
          (* 目前結果不會被修改 *)  
ST boo_var2 (* boo_var2 := TRUE *)  
LD false (* 目前結果 := FALSE *)  
R boo_var1 (* 不會做任何事 – boo_var1 不改變 *)
```

⇨ JMP 運算符號

運算： 跳躍至指定的標籤

允許的修飾字： C N

運算子： 標籤定義在同一個 IL 程式中

範例：

(* 下面的範例是用來測試類比選擇器 (0 或 1 或 2) 的數值 *)
 (* 以便設定三個輸出布林值的其中之一。由 JMPc 運算元來決定 *)
 (* 測試 “等於 0” 的結果 *)

```
JMPex:  LD  selector  (* selector 是 0 或 1 或 2 *)
        BOO          (* 轉換成布林型態 *)
        JMPc  test1  (* if selector = 0 then *)
        LD  true
        ST  bo0  (* bo0 := true *)
        JMP  JMPend  (* 程式結束 *)

test1: LD  selector
        SUB 1        (* 將 selector 減為 0 或 1 *)
        BOO          (* 轉換成布林型態 *)
        JMPc  test2  (* if selector = 0 then *)
        LD  true
        ST  bo1  (* bo1 := true *)
        JMP  JMPend  (* 程式結束 *)

test2: LD  true  (* 最後可能 *)
        ST  bo2  (* bo2 := true *)

JMPend:                                (* IL 程式結束 *)
```

RET 運算元

運算： 結束目前 IL 指令。假如 IL 程序是一個副程式，目前結果就回傳給呼叫程式。

允許的修飾字： C N

運算子： 無

範例：

(* 下面的範例是用來測試類比選擇器 (0 或 1 或 2) 的數值 *)

語言參考

(* 以便設定三個輸出布林值的其中之一。由 JMPc 運算元來決定 *)

(* 測試 “等於0” 的結果 *)

JMPex: LD selector (* selector 是 0 或 1 或 2 *)

BOO (* 轉換成布林型態 *)

JMPC test1 (* if selector = 0 then *)

LD true

ST bo0 (* bo0 := true *)

RET (* 程式結束 – 傳回 0 *)

(* 減少 selector *)

test1: LD selector

SUB 1 (* selector 現在是 0 或 1 *)

BOO (* 轉換成布林型態 *)

JMPC test2 (* if selector = 0 then *)

LD true

ST bo1 (* bo1 := true *)

LD 1 (* 載入實際 selector 值 *)

RET (* 程式結束 – 傳回 1 *)

(* 最後可能 *)

test2: RETNC (* 假如 selector 有一個有效的數值就回傳 *)

LD true

ST bo2 (* bo2 := true *)

LD 2 (* 載入實際 selector 值 *)

(* IL 程式結束 – 傳回 2 *)

□ “)” 運算符號

運算： 執行一個延遲執行。此延遲執行是以 ‘(’ 通知的。

允許的修飾字： 無

運算子： 無

範例：

```
(* 下面程式插入了數個延遲執行： *)
(* res := a1 + (a2 * (a3 - a4) * a5) + a6; *)

Delayed: LD  a1  (* 目前結果 := a1; *)
          ADD(  a2  (* 延遲 ADD，目前結果 := a2; *)
          MUL(  a3  (* 延遲 MUL，目前結果 := a3; *)
          SUB a4  (* 目前結果 := a3 - a4; *)
          )      (* 執行延遲 MUL，目前結果 := a2 * (a3-a4); *)
          MUL a5  (* 目前結果 := a2 * (a3 - a4) * a5; *)
          )      (* 執行延遲 ADD *)
          (* 目前結果 := a1 + (a2 * (a3 - a4) * a5); *)
          ADD a6  (* 目前結果 := a1 + (a2 * (a3 - a4) * a5) + a6; *)
          ST  res (* 儲存目前結果到變數 res *)
```

□ 呼叫副程式或函式

假如在 IL 語言中呼叫副程式或函數 (以任何 IL、ST、LD、FBD 或 “C” 語言撰寫的) 時，使用副程式或函數的名稱作為運算元來使用。

運算： 執行副程式或函式 – 副程式或函式的回傳值會存入 IL 的目前結果

允許的修飾字： 無

運算子： 呼叫前，必須將第一個呼叫參數存入目前結果。剩下的參數被表示在運算子欄位中，並且以逗號分隔開。

範例：

```
(* 呼叫程式：將類比數值轉換成時間數值 *)

Main: LD  bi0
      SUBPRO bi1,bi2  (* 呼叫副程式來得到類比數值 *)
      ST  result      (* 副程式將數值傳回為目前結果 *)
      GT  vmax        (* 測試數值是否溢位 *)
```

語言參考

RETC (* 假如溢位就回傳 *)
LD result
MUL 1000 (* 將秒轉換成毫秒 *)
TMR (* 轉換成時間 *)
ST tmval (* 儲存轉換值給計時器 *)

(* 被呼叫的副程式名稱爲 'SUBPRO' : 將副程式的三個布林輸入參數
(* in0 , in1 , in2 視爲二元數值形態的類比數值，並且估算之*)

LD in2
ANA (* 目前結果 := ana (in2); *)
MUL 2 (* 目前結果 := 2*ana (in2); *)
ST temporary (* temporary := 目前結果 *)
LD in1
ANA
ADD temporary (* 目前結果 := 2*ana (in2) + ana (in1); *)
MUL 2 (* 目前結果 := 4*ana (in2) + 2*ana (in1); *)
ST temporary (* temporary := 目前結果 *)
LD in0
ANA
ADD temporary (* 目前結果 := 4*ana(in2)+2*ana(in1)+ana(in0); *)
ST SUBPRO (* 回傳目前結果給呼叫程式 *)

□ 呼叫功能方塊：CAL 運算元

運算： 呼叫功能方塊

允許的修飾字： C N

運算子： 功能方塊樣例的名稱。

使用LD/ST 運算程序呼叫方塊之前，方塊的輸入參數必須先指定。

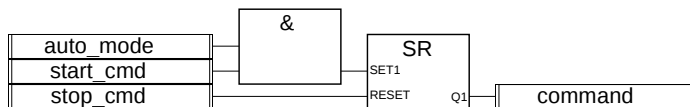
假如要使用輸出參數，必須先知道參數名稱。

範例一：

(* 呼叫功能方塊 SR : SR1 是一個 SR 樣例 *)

```
LD  auto_mode
AND start_cmd
ST  SR1.set1
LD  stop_cmd
ST  SR1.reset
CAL SR1
LD  SR1.Q1
ST  command
```

(* FBD 相等式 : *)



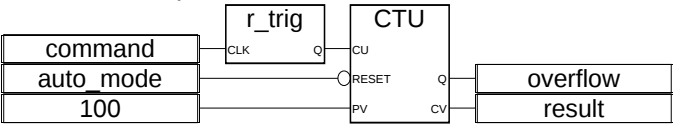
範例二：

(* 假設 R_TRIG1 是一個 R_TRIG 方塊樣例, CTU1 是一個 CTU 方塊樣例 *)

```
LD  command
ST  R_TRIG1.clk
CAL R_TRIG1
LD  R_TRIG1.Q
ST  CTU1.cu
LDN auto_mode
ST  CTU1.reset
LD  100
ST  CTU1.pv
CAL CTU1
LD  CTU1.Q
ST  overflow
LD  CTU1.cv
ST  result
```

語言參考

(* FBD 相等式 : *)



B.9 標準運算元、功能方塊及函數

B.9.1 標準運算元

下列是 IEC 語言中的標準運算元。

資料處理 指定、類比值反相

布林運算元 布林且 (boolean AND)

布林或 (boolean OR)

布林互斥或 (exclusive OR)

算術運算元 加法

減法

乘法

除法

邏輯運算元 類比逐位元且遮罩 (AND mask)

類比逐位元或遮罩 (OR mask)

類比逐位元互斥或遮罩 (Exclusive OR mask)

逐位元反相

比較測試 小於

小於等於

大於

大於等於

等於

不等於

資料轉換 轉換成布林

轉換成整數類比

轉換成實數類比

轉換成時間

轉換成訊息

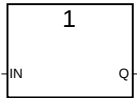
其他 字串的連結

語言參考

直接操作

處理 I/O 接點

1 gain



引數：

IN 任何形態

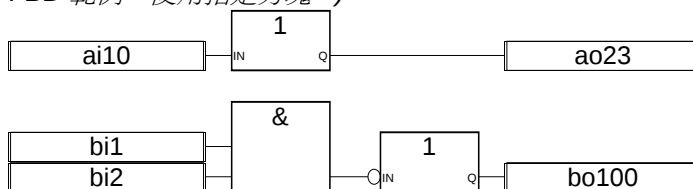
Q 任何形態

描述：

將一變數指定到另一個變數。

此方塊對於圖形輸入與圖形輸出間的直接連結來說是非常有用的。它也可以用來反轉一個用線連結到圖形輸出的狀態（使用布林的反相線）。

(* FBD 範例：使用指定方塊 *)



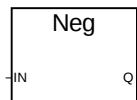
(* ST 相等式：*)

ao23 := ai10;

bo100 := NOT (bi1 AND bi2);

(* IL 相等式：*)

```
LD      ai10
ST      ao23
LD      bi1
AND     bi2
STN     bo100
```

NEG

引數：

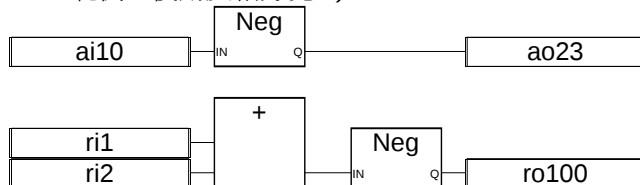
IN 整數，實數 輸入與輸出必須是相同的格式

Q 整數，實數

描述：

將變數設定成反相 (正變負，負變正)。

(* FBD 範例：使用反相方塊 *)



(* ST 相等式 : *)

$ao23 := - (ai10);$

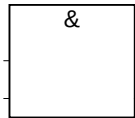
$ro100 := - (ri1 + ri2);$

(* IL 相等式 : *)

```
LD      ai10
MUL     -1
ST      ao23
LD      ri1
ADD     ri2
MUL     -1.0
ST      ro100
```

& AND

語言參考



註：此運算元輸入的數量可以擴展到多於兩個。

引數：

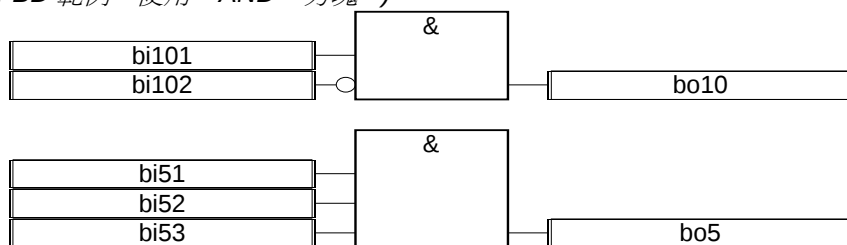
(輸入) 布林

輸出 布林 輸入的布林且 (boolean AND) 運算

描述：

將兩個或多於兩個的條件做布林且 (AND)。

(* FBD 範例：使用 “AND” 方塊 *)



(* ST 相等式：*)

bo10 := bi101 AND NOT (bi102);

bo5 := (bi51 AND bi52) AND bi53;

(* IL 相等式：*)

LD bi101 (* 目前結果 := bi101 *)

ANDNbi102 (* 目前結果 := bi101 AND not(bi102) *)

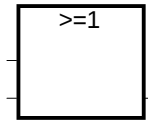
ST bo10(* bo10 := 目前結果 *)

LD bi51 (* 目前結果 := bi51 *)

& bi52 (* 目前結果 := bi51 AND bi52 *)

& bi53 (* 目前結果 := (bi51 AND bi52) AND bi53 *)

ST bo5 (* bo5 := 目前結果 *)

>=1 OR

註：此運算元輸入的數量可以擴展到多於兩個。

引數：

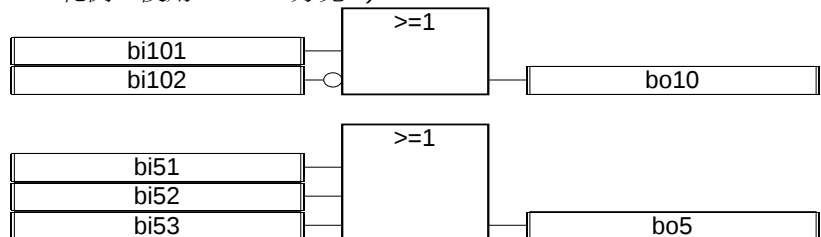
(輸入) 布林

輸出 布林 輸入條件的布林或 (OR)

描述：

將兩個或多於兩個的條件做布林或 (OR)。

(* FBD 範例：使用 “OR” 方塊 *)



(* ST 相等式 : *)

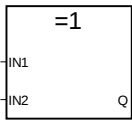
$bo10 := bi101 \text{ OR NOT } (bi102);$

$bo5 := (bi51 \text{ OR } bi52) \text{ OR } bi53;$

(* IL 相等式 : *)

```
LD      bi101
ORN     bi102
ST      bo10
LD      bi51
OR      bi52
OR      bi53
ST      bo5
```

=1 XOR



引數：

IN1 布林

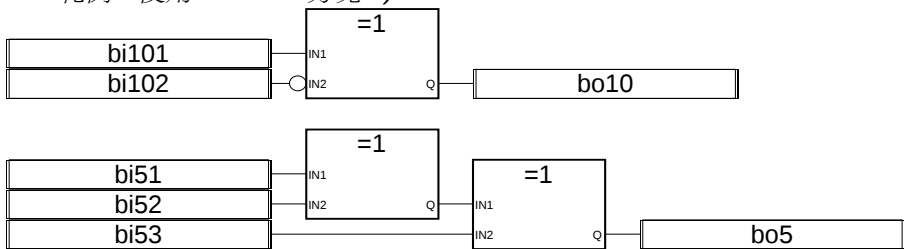
IN2 布林

Q 布林 兩個輸入條件的布林互斥或 (XOR)

描述：

將兩個條件做布林互斥或 (XOR)。

(* FBD 範例：使用 “XOR” 方塊 *)



(* ST 相等式：*)

$bo10 := bi101 \text{ XOR NOT } (bi102);$

$bo5 := (bi51 \text{ XOR } bi52) \text{ XOR } bi53;$

(* IL 相等式：*)

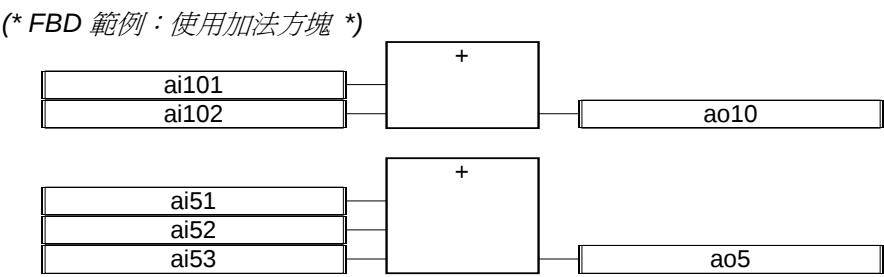
```
LD      bi101
XORN    bi102
ST      bo10
LD      bi51
XOR     bi52
XOR     bi53
ST      bo5
```



註：此運算元輸入的數量可以擴展到多於兩個。

引數：
(輸入) 整數，實數 可以是整數或實數
(所有的輸入必須有相同的格式)
輸出 整數，實數 輸入條件相加

描述：
將兩個或多於兩個的類比變數加起來。



(* ST 相等式 : *)
 $ao10 := ai101 + ai102;$
 $ao5 := (ai51 + ai52) + ai53;$

(* IL 相等式 : *)
LD ai101
ADD ai102
ST ao10
LD ai51
ADD ai52
ADD ai53

語言參考

ST ao5



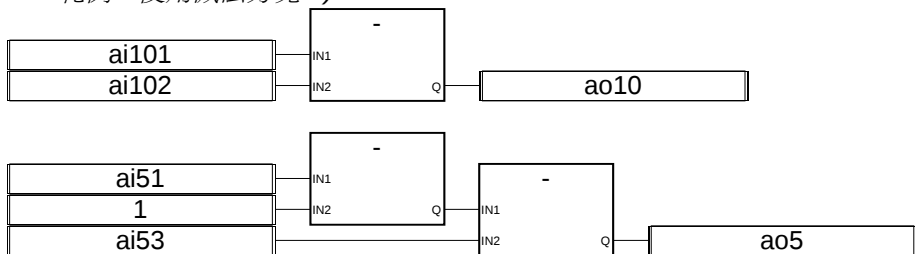
引數：

IN1 整數，實數 可以是整數或實數
IN2 整數，實數 (IN1 與 IN2 必須有相同的格式)
Q 整數，實數 相減 (IN1-IN2)

描述：

將兩個類比變數相減 (第一引數減第二引數)。

(* FBD 範例：使用減法方塊 *)



(* ST 相等式：*)

ao10 := ai101 - ai102;
ao5 := (ai51 - 1) - ai53;

(* IL 相等式：*)

```
LD      ai101
SUB     ai102
ST      ao10
LD      ai51
SUB     1
SUB     ai53
```

ST ao5



註：此運算元輸入的數量可以擴展到多於兩個。

引數：

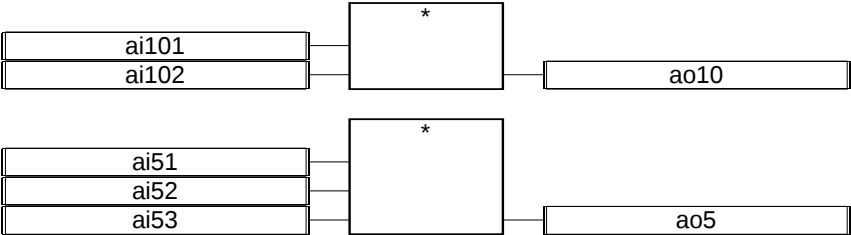
(輸入) 整數，實數 可以是整數或實數
(所有的輸入必須有相同的格式)

輸出 整數，實數 輸入條件的相乘

描述：

將兩個或多於兩個的類比變數相乘。

(* FBD 範例：使用乘法方塊 *)



(* ST 相等式 : *)

ao10 := ai101 * ai102;
ao5 := (ai51 * ai52) * ai53;

(* IL 相等式 : *)

```
LD            ai101
              *
MUL   ai102
              *
ST     ao10

LD            ai51
```

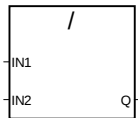
語言參考

MUL ai52

MUL ai53

ST ao5

/



引數：

IN1 整數，實數 可以是整數或實數

IN2 整數，實數 非零類比數值

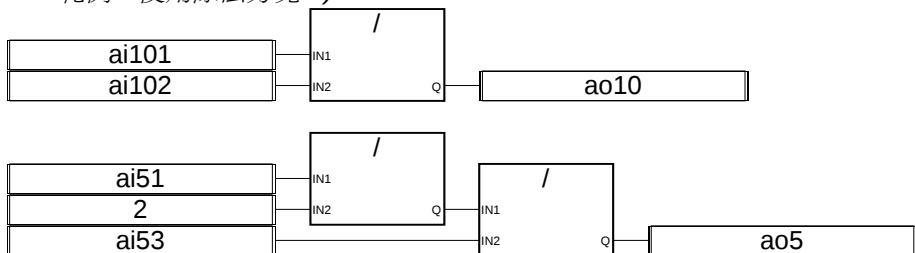
(IN1 與 IN2 必須是相同的格式)

Q 整數，實數 IN1 除以 IN2 整數或實數之值

描述：

將兩個類比變數相除 (第一個引數被第二個引數除)。

(* FBD 範例：使用除法方塊 *)



(* ST 相等式 : *)

ao10 := ai101 / ai102;

ao5 := (ai5 / 2) / ai53;

(* IL 相等式 : *)

LD ai101

DIV ai102

ST ao10

```

LD    ai51
DIV   2
DIV   ai53
ST          a05

```

AND_MASK



引數：

IN 整數 必須是整數的格式

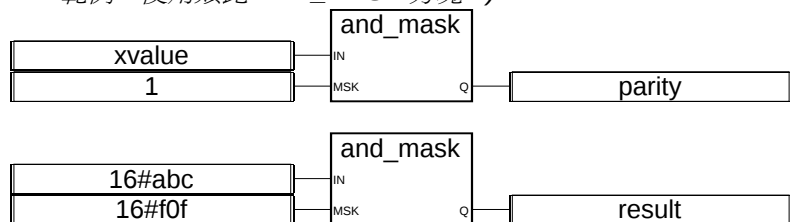
MSK 整數 必須是整數的格式

Q 整數 IN 與 MSK 間做位元對位元的邏輯且 (AND)

描述：

整數類比值位元對位元且遮罩。

(* FBD 範例：使用類比 AND_MASK 方塊 *)



(* ST 相等式：*)

parity := AND_MASK (xvalue, 1); (* 假如 xvalue 為奇數，值為 1 *)

result := AND_MASK (16#abc, 16#f0f); (* 等於 16#a0c *)

(* IL 相等式：*)

```

LD          xvalue
AND_MASK    1
ST    parity
LD          16#abc

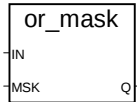
```

語言參考

AND_MASK 16#f0f

ST result

OR_MASK



引數：

IN 整數 必須是整數的格式

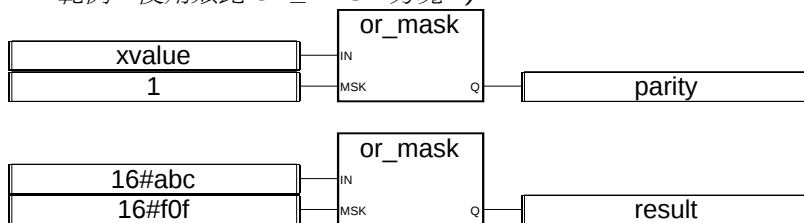
MSK 整數 必須是整數的格式

Q 整數 IN 與 MSK 間做位元對位元的邏輯或 (OR)

描述：

整數類比值位元對位元或遮罩。

(* FBD 範例：使用類比 OR_MASK 方塊 *)



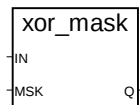
(* ST 相等式：*)

is_odd := OR_MASK (xvalue, 1); (* 使 is_odd 永遠是奇數值 *)

result := OR_MASK (16#abc, 16#f0f); (* 等於 16#fbf *)

(* IL 相等式：*)

```
LD      xvalue
OR_MASK 1
ST      is_odd
LD      16#abc
OR_MASK 16#f0f
ST      result
```

XOR_MASK

引數：

IN 整數 必須是整數的格式

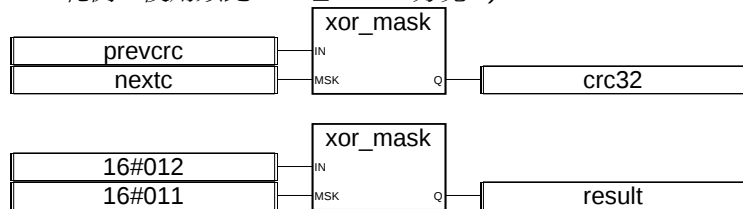
MSK 整數 必須是整數的格式

Q 整數 IN 與 MSK 間做位元對位元的邏輯互斥或 (XOR)

描述：

整數類比值位元對位元互斥或遮罩。

(* FBD 範例：使用類比 XOR_MASK 方塊 *)



(* ST 相等式 : *)

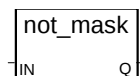
`crc32 := XOR_MASK (prevcrc, nextc);`

`result := XOR_MASK (16#012, 16#011); (* 等於 16#003 *)`

(* IL 相等式 : *)

```

LD      prevcrc
XOR_MASK  nextc
ST      crc32
LD      16#012
XOR_MASK  16#011
ST      result
  
```

NOT_MASK

語言參考

引數：

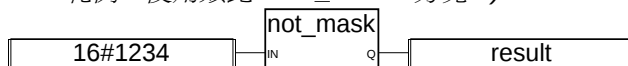
IN 整數 必須是整數的格式

Q 整數 對 IN 的 32 位元做位元對位元的補數運算

描述：

整數類比值位元對位元補數運算遮罩。

(* FBD 範例：使用類比 NOT_MASK 方塊 *)



(* ST 相等式：*)

result := NOT_MASK (16#1234); (* result := 16#FFFF_EDCB *)

(* IL 相等式：*)

```
LD      16#1234
NOT_MASK
ST      result
```

<



引數：

IN1 整數，實數，計時器，訊息

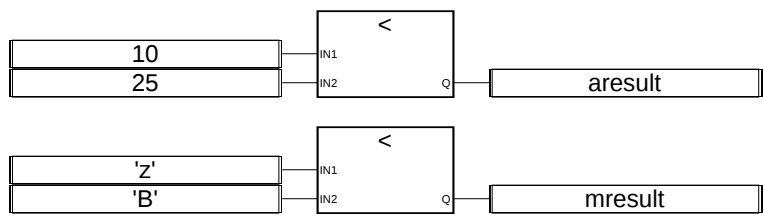
IN2 整數，實數，計時器，訊息 IN1 與 IN2 必須是相同的格式

Q 布林 如果 IN1 < IN2 則為真

描述：

測試是否一值小於另一值（對類比，計時器或訊息變數）。

(* FBD 範例：使用“小於”方塊 *)



(* ST 相等式 : *)
aresult := (10 < 25); (* aresult 為真 *)
mresult := ('z' < 'B'); (* mresult 為假 *)

(* IL 相等式 : *)
LD 10
LT 25
ST aresult
LD 'z'
LT 'B'
ST mresult

<=

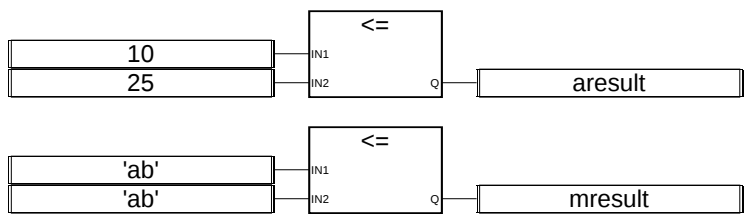


引數：
IN1 整數，實數，訊息
IN2 整數，實數，訊息 IN1 與 IN2 必須是相同的格式
Q 布林 如果 IN1 <= IN2 則為真

描述：
測試是否一值小於或等於另一值 (對類比或訊息變數)。

(* FBD 範例：使用 “小於或等於” 方塊 *)

語言參考



(* ST 相等式 : *)

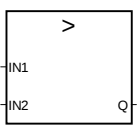
areult := (10 <= 25); (* areult 為真 *)

mresult := ('ab' <= 'ab'); (* mresult 為假 *)

(* IL 相等式 : *)

LD 10
LE 25
ST areult
LD 'ab'
LE 'ab'
ST mresult

>



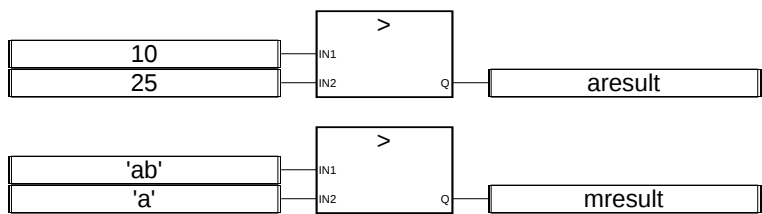
引數：

- IN1 整數，實數，計時器，訊息
- IN2 整數，實數，計時器，訊息 IN1 與 IN2 必須是相同的格式
- Q 布林 如果 IN1 > IN2 則為真

描述：

測試是否一值大於另一值 (對類比、計時器或訊息變數)。

(* FBD 範例：使用 “大於” 方塊 *)



(* ST 相等式 : *)
aresult := (10 > 25); (* aresult 為假 *)
mresult := ('ab' > 'a'); (* mresult 為真 *)

(* IL 相等式 : *)
LD 10
GT 25
ST aresult
LD 'ab'
GT 'a'
ST mresult

>=



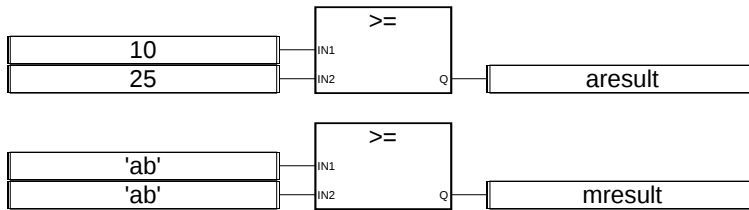
引數：

- IN1 整數，實數，訊息
- IN2 整數，實數，訊息 IN1 與 IN2 必須是相同的格式
- Q 布林 如果 IN1 >= IN2 則為真

描述：
測試是否一值大於或等於另一值 (對類比或訊息變數)。

(* FBD 範例：使用 “大於或等於” 方塊 *)

語言參考



(* ST 相等式 : *)

areresult := (10 >= 25); (* *areresult* 為假 *)

mresult := ('ab' >= 'ab'); (* *mresult* 為真 *)

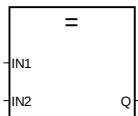
(* IL 相等式 : *)

```

LD      10
GE      25
ST      areresult
LD      'ab'
GE      'ab'
ST      mresult

```

=



引/數：

IN1 整數，實數，訊息

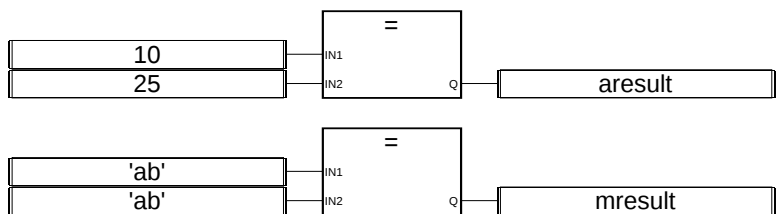
IN2 整數，實數，訊息 **IN1** 與 **IN2** 必須是相同的格式

Q 布林 如果 **IN1** = **IN2** 則為真

描述：

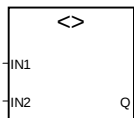
測試是否一值等於另一值 (對類比或訊息變數)。

(* FBD 範例：使用 “等於” 方塊 *)



(* ST 相等式 : *)
areresult := (10 = 25); (* areresult 為假 *)
mresult := ('ab' = 'ab'); (* mresult 為真 *)

(* IL 相等式 : *)
LD 10
EQ 25
ST areresult
LD 'ab'
EQ 'ab'
ST mresult

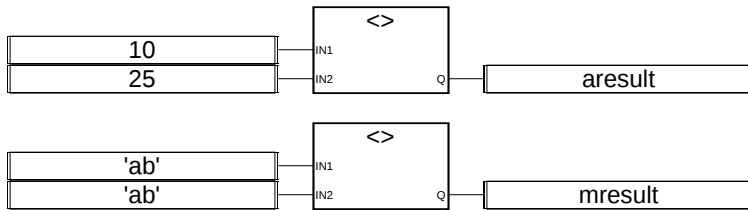


引數：
IN1 整數，實數，訊息
IN2 整數，實數，訊息 IN1 與 IN2 必須是相同的格式
Q 布林 如果 IN1 <> IN2 則為真

描述：
測試是否一值不等於另一值 (對類比或訊息變數)。

(* FBD 範例：使用 “不等於” 方塊 *)

語言參考



(* ST 相等式 : *)

areresult := (10 <> 25); (* areresult 為真 *)

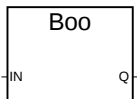
mresult := ('ab' <> 'ab'); (* mresult 為假 *)

(* IL 相等式 : *)

```

LD      10
NE      25
ST      areresult
LD      'ab'
NE      'ab'
ST      mresult
  
```

BOO



引數：

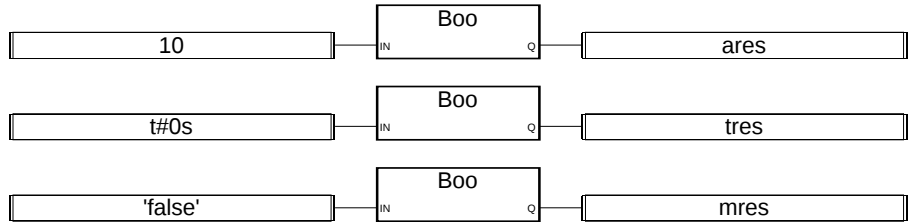
IN 任何變數 任何非布林值

Q 布林 如果是非零的數值則得真
 如果是為零的數值則得假
 如果是 'TRUE' 的字串則得真
 如果是 'FALSE' 的字串則得假

描述：

將任何變數轉換成布林形態。

(* FBD 範例：使用 “轉換成布林” 方塊 *)



(* ST 相等式 : *)

ares := BOO (10); (* ares 為真 *)

tres := BOO (t#0s); (* tres 為假 *)

mres := BOO ('false'); (* mres 為假 *)

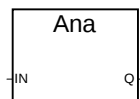
(* IL 相等式 : *)

```

LD      10
BOO
ST      ares
LD      t#0s
BOO
ST      tres
LD      'false'
BOO
ST      mres

```

ANA



引數：

IN 任何變數 任何非整數的類比值

Q 整數 如果 IN 是假則得 0 / 如果 IN 是真則得 1

計時器變數轉換為以千分之一秒為單位的數值

實數類比值抽取出整數部分

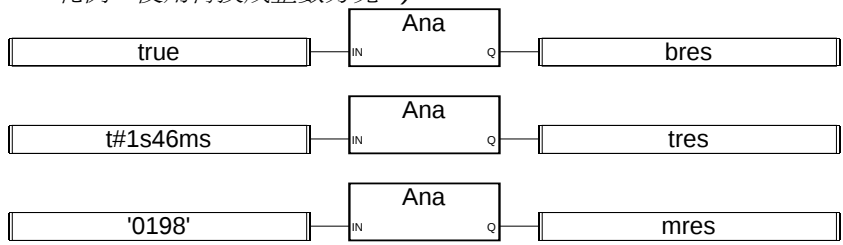
字串轉為 10 進位整數

語言參考

描述：

將任何變數轉換成整數形態。

(* FBD 範例：使用轉換成整數方塊 *)



(* ST 相等式 : *)

bres := ANA (true); (* bres 為 1 *)

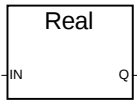
tres := ANA (t#1s46ms); (* tres 為 1046 *)

mres := ANA ('0198'); (* mres 為 198 *)

(* IL 相等式 : *)

LD true
ANA
ST bres
LD t#1s46ms
ANA
ST tres
LD '0198'
ANA
ST mres

REAL



引數：

IN 布林，整數，計時器 任何非實數的類比值 (不允許字串)

Q 實數 如果 IN 是假則得 0.0 / 如果 IN 是真則得 1.0

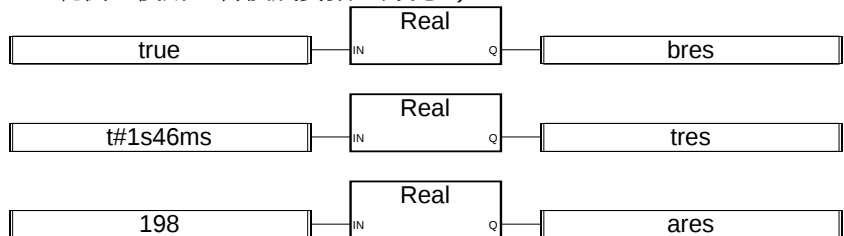
計時器變數轉為以千分之一秒為單位的數值

整數類比值轉為同等數值

描述：

將任何變數轉換成實數形態。

(* FBD 範例：使用“轉換成實數”方塊 *)



(* ST 相等式 : *)

bres := REAL (true); (* bres 為 1.0 *)

tres := REAL (t#1s46ms); (* tres 為 1046.0 *)

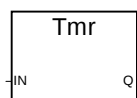
ares := REAL (198); (* ares 為 198.0 *)

(* IL 相等式 : *)

```

LD      true
REAL
ST      bres
LD      t#1s46ms
REAL
ST      tres
LD      198
REAL
ST      ares
  
```

TMR



語言參考

引數：

IN 整數，實數 任何非計時器變數的值

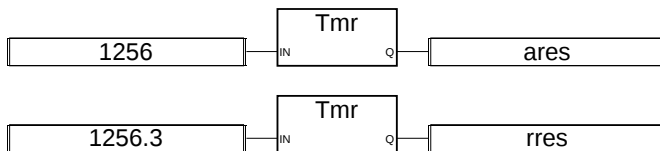
IN 是以千分之一秒為單位的數值 (如果 IN 是實數的話則取 IN 的整數部份)

Q 計時器 IN 的值以時間數值來表現

描述：

將任何變數轉換成計時器形態。

(* FBD 範例：使用轉換成時間方塊 *)



(* ST 相等式 : *)

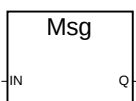
ares := TMR (1256); (* ares := t#1s256ms *)

rres := TMR (1256.3); (* rres := t#1s256ms *)

(* IL 相等式 : *)

```
LD      1256
TMR
ST      ares
LD      1256.3
TMR
ST      rres
```

MSG



引數：

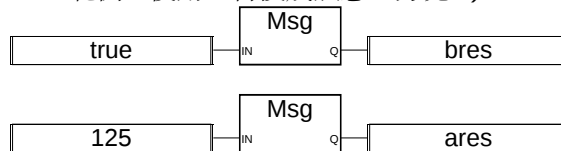
IN 布林，整數，實數 任何非字串的值

Q 訊息 如果 IN 為布林值則得 'false' 或 'true'
如果 IN 為類比值則以十進位字串來表示

描述：

將任何變數轉換成訊息型態。

(* FBD 範例：使用“轉換成訊息”方塊 *)



(* ST 相等式：*)

bres := MSG (true); (* bres 為 'TRUE' *)

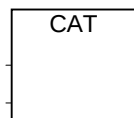
ares := MSG (125); (* ares 為 '125' *)

(* IL 相等式：*)

```

LD      true
MSG
ST      bres
LD      125
MSG
ST      ares
  
```

CAT



註：此運算元，其輸入的數量可以擴展到多於兩個。

引數：

(輸入) 訊息 (將所有字串加起來，其長度必須不超過輸出訊息的容許量)

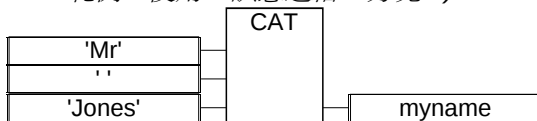
語言參考

輸出 訊息 將輸入訊息連結起來

描述：

將數個訊息字串連結成一個訊息字串。

(* FBD 範例：使用 “訊息連結” 方塊 *)



(* ST 相等式：使用運算元 + *)

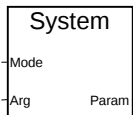
myname := ('Mr'+' ') + 'Jones';

(* 意義：myname := 'Mr Jones' *)

(* IL 相等式：*)

```
LD      'Mr'
ADD     ' '
ADD     'Jones'
ST      myname
```

SYSTEM



引數：

Mode 整數 辨識系統參數以及執行模式

Arg 整數，計時器 寫入一個新值

Param 整數 執行參數之後的返回值

描述：

使用系統參數。

下面的明細表是 **SYSTEM** 功能中所允許的指令 (已定義過的關鍵字)：

命令	意義
<code>SYS_TALLOWED</code>	讀取允許的週期時間
<code>SYS_TCURRENT</code>	讀取現在的週期時間
<code>SYS_TMAXIMUM</code>	讀取最大的週期時間
<code>SYS_TOVERFLOW</code>	讀取週期時間溢位次數
<code>SYS_TRESET</code>	重置週期計數器
<code>SYS_TWRITE</code>	改變週期時間
<code>SYS_ERR_TEST</code>	檢查執行期錯誤
<code>SYS_ERR_READ</code>	讀取最早的執行期錯誤

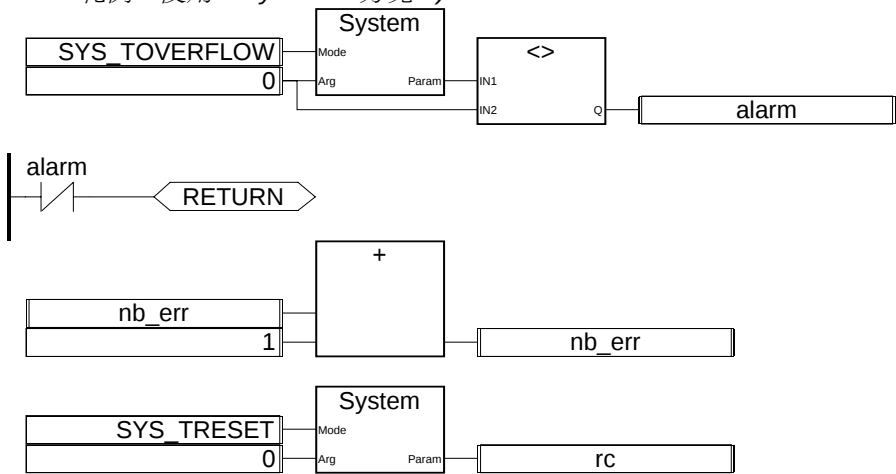
下面是對 **SYSTEM** 功能中已定義過的函式，其所需要的參數：

命令	引數	返回值
<code>SYS_TALLOWED</code>	0	允許的週期時間
<code>SYS_TCURRENT</code>	0	現在的週期時間
<code>SYS_TMAXIMUM</code>	0	偵測到的最大週期時間
<code>SYS_TOVERFLOW</code>	0	溢位次數
<code>SYS_TRESET</code>	0	0
<code>SYS_TWRITE</code>	新的允許週期時間	寫入的時間值
<code>SYS_ERR_TEST</code>	0	如果未偵測到錯誤則

語言參考

ST		為0
SYS_ERR_RE AD	0	最早的錯誤碼

(* FBD 範例：使用 “System” 方塊 *)



(* ST 相等式：*)

```
alarm := (SYSTEM (SYS_TOVERFLOW, 0) <> 0);
```

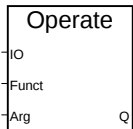
```
If (alarm) Then
```

```
  nb_err := nb_err + 1;
```

```
  rc := SYSTEM ( SYS_TRESET, 0);
```

```
End_if;
```

OPERATE



引數：

IO 任何形態 輸入或輸出變數

Funct 整數 執行的動作

Arg 整數 I/O 操作的參數

Q 整數 返回之核對值

描述：

直接存取 I/O 接點。

OPERATE 參數的意義取決於 I/O 介面的使用。參考您的硬體手冊或對應 I/O 板的技術註記，以學習更多有關 OPERATE 的功能。

B.9.2 標準的功能方塊

下面是由 ISaGRAF 系統所提供的標準功能方塊。這些功能方塊是已經被定義過，且不需要在程式庫中宣告。

布林 SR 設定雙重觸發器

RS 重置雙重觸發器

R_Trig 上升邊緣偵測

F_Trig 下降邊緣偵測

SEMA 信號發送器

計數 CTU 向上計數器

CTD 向下計數器

CTUD 向上及向下計數器

計時器 TON 啟動延時

TOF 關閉延時

TP 脈沖定時

整數類比值 CMP 完整比較的功能方塊

StackInt 整數類比值的堆疊

實數類比值 AVERAGE 對數個樣本求其平均

HYSTER 對不同的實數產生布林遲滯現象

LIM_ALARM 高 / 低限制警告

INTEGRAL 對時間的積分

DERIVATE 根據時間的微分

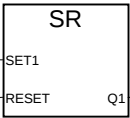
訊號產生 BLINK 布林訊號的閃爍處理

SIG_GEN 訊號產生器

語言參考

註：當新的“C” 功能方塊被創造出來後，就可以從 FBD 語言中叫出來使用。

SR



引數：

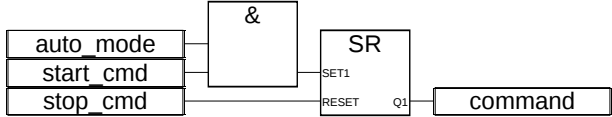
- SET1** 布林 如果為真，則將 Q1 設為真 (支配端)
- RESET** 布林 如果為真，則將 Q1 重置為假
- Q1** 布林 布林記憶狀態

描述：

設定雙重設定的值域：(請看下方的真值表)

S	R	Q	re
et	es	1	su
1	et		lt
			Q
			1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(* FBD 範例：使用 “SR” 方塊 *)



(* ST 相等式：我們假設 SR1 是 SR 方塊的樣例 *)

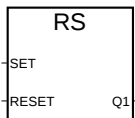
SR1((auto_mode & start_cmd), stop_cmd);

command := SR1.Q1;

(* IL 相等式：*)

```
LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
LD      SR1.Q1
ST      command
```

RS



引數：

SET 布林 如果為真，則將Q1 設為真

RESET 布林 如果為真，則將Q1 重置為假 (支配端)

Q1 布林 布林記憶狀態

描述：

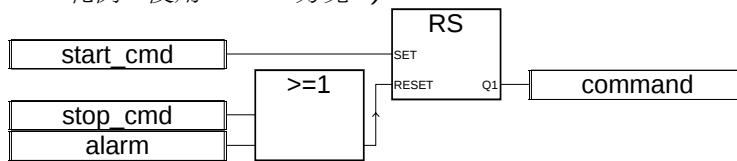
重置雙穩定的值域：(請看下方的真值表)

S	R	Q	re
et	es	1	su
	et		lt
			Q
			1
0	0	0	0
0	0	1	1

語言參考

0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

(* FBD 範例：使用 “RS” 方塊 *)



(* ST 相等式：我們假設 RS1 是 RS 方塊的樣例 *)

RS1(start_cmd, (stop_cmd OR alarm));

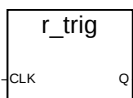
command := SR1.Q1;

(* IL 相等式：*)

```

LD      start_cmd
ST      RS1.set
LD      stop_cmd
OR      alarm
ST      RS1.reset
CAL     RS1
LD      RS1.Q1
ST      command
  
```

R_TRIG



引數：

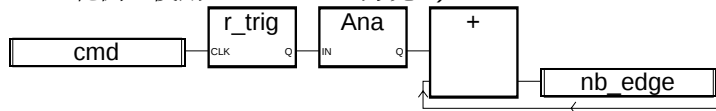
CLK 布林 任何布林變數

Q 布林 當 CLK 的狀態由假變為真則得真
如果是其他情況則得假

描述：

偵測布林變數的上升邊緣。

(* FBD 範例：使用 “R_TRIG” 方塊 *)



(* ST 相等式：我們假設 R_TRIG1 是 R_TRIG 方塊的樣例 *)

R_TRIG1(cmd);

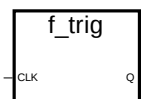
nb_edge := ANA(R_TRIG1.Q) + nb_edge;

(* IL 相等式：*)

```

LD      cmd
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ANA
ADD     nb_edge
ST      nb_edge
  
```

F_TRIG



引數：

CLK 布林 任何布林變數

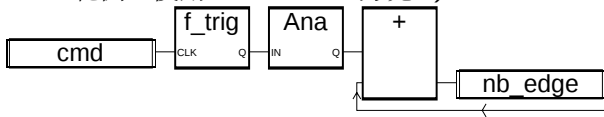
Q 布林 當 CLK 的狀態由真變為假則得真
如果是其他情況則得假

語言參考

描述：

偵測布林變數的下降邊緣。

(* FBD 範例：使用 “F_TRIG” 方塊 *)



(* ST 相等式：我們假設 F_TRIG1 是 F_TRIG 方塊的樣例 *)

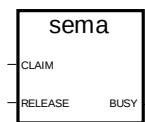
F_TRIG1(cmd);

nb_edge := ANA(F_TRIG1.Q) + nb_edge;

(* IL 相等式：*)

```
LD      cmd
ST      F_TRIG1.clk
CAL     F_TRIG1
LD      F_TRIG1.Q
ANA
ADD     nb_edge
ST      nb_edge
```

SEMA



引數：

CALIM 布林 “測試與設定” 指令

RELEASE 布林 重置信號

BUSY 布林 信號的狀態

描述：

(* “X” 是一個初始值為假的布林變數 *)

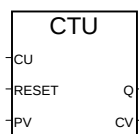
busy := x;

```

If claim Then
    x := True;
Else
If release Then
    busy := False;
    x := False;
End_if;
End_if;

```

CTU



引數：

CU 布林 計數命令輸入 (當 CU 為真才計數)

RESET 布林 重置指令 (高優先權)

PV 整數 設定的最大計數值

Q 布林 溢位：當 CV = PV 時得真

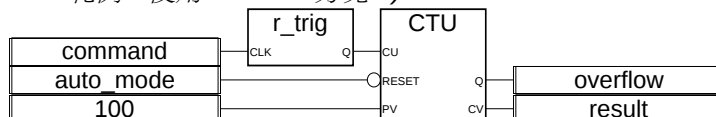
CV 整數 計數器的結果

注意：CTU 方塊並不區分計數命令輸入 (CU) 的上升或下降邊緣，它必須結合一個 “R_TRIG” 或 “F_TRIG” 方塊來做出一個脈波計數器。

描述：

從 0 開始，每次計數加一到給予的值 (整數)。

(* FBD 範例：使用 “CTU” 方塊 *)



語言參考

(* ST 相等式：我們假設 R_TRIG1 是 R_TRIG 方塊的樣例，CTU1 是 CTU 方塊的樣例 *)

```
CTU1(R_TRIG1(cmmand),NOT(auto_mode),100);
```

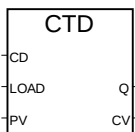
```
overflow := CTU1.Q;
```

```
result := CTU1.CV;
```

(* IL 相等式：*)

```
LD          command
ST  R_TRIG1.clk
CAL  R_TRIG1
LD  R_TRIG1.Q
ST  CTU1.cu
LDN auto_mode
ST  CTU1.reset
LD  100
ST  CTU1.pv
CAL CTU1
LD  CTU1.Q
ST  overflow
LD  CTU1.cv
ST  result
```

CTD



引數：

CD 布林 計數命令輸入 (當 CD 為真才向下計數)

LOAD 布林 讀取指令 (高優先權)
(當 LOAD 為真時 CV = PV)

PV 整數 程式設定的初始值

Q 布林 下溢位：當 CV = 0 時得真

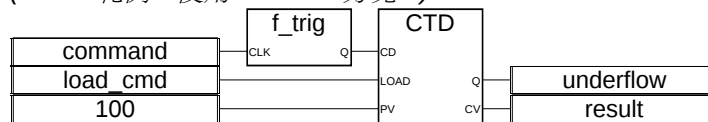
CV 整數 計數器的結果

注意：CTD 方塊並不區分計數命令輸入 (CD) 的上升或下降邊緣，它必須結合一個 “R_TRIG” 或 “F_TRIG” 方塊來做出一個脈波計數器。

描述：

從給予的值 (整數) 開始，每次計數減一，直到 0。

(* FBD 範例：使用 “CTD” 方塊 *)



(* ST 相等式：我們假設 F_TRIG1 是 F_TRIG 方塊的樣例，CTD1 是 CTD 方塊的樣例 *)

CTD1(F_TRIG1(cmmand),load_cmd,100);

underflow := CTD1.Q;

result := CTD1.CV;

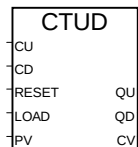
(* IL 相等式：*)

```

LD      command
ST      F_TRIG1.clk
CAL     F_TRIG1
LD      F_TRIG1.Q
ST      CTD1.cd
LD      load_cmd
ST      CTD1.load
LD      100
ST      CTD1.pv
CAL     CTD1
LD      CTD1.Q
ST      underflow
LD      CTD1.cv
  
```

ST

result

CTUD

引數：

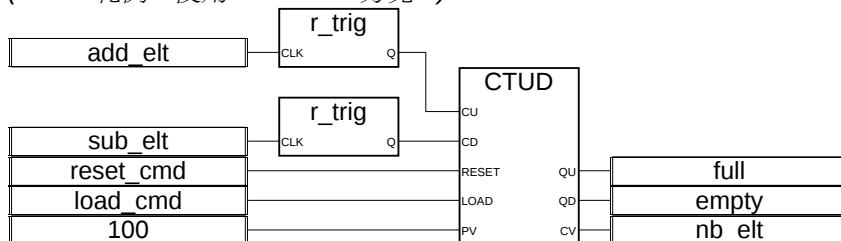
CU 布林 向上計數 (當 CU 為真)**CD** 布林 向下計數 (當 CD 為真)**RESET** 布林 重置指令 (高優先權)
(當 RESET 為真時 CV = 0)**LOAD** 布林 讀取指令 (當 LOAD 為真時 CV = PV)**PV** 整數 設定的最大計數值**QU** 布林 溢位：當 CV = PV 時得真**QD** 布林 下溢位：當 CV = 0 時得真**CV** 整數 計數器的結果

注意：CTUD 方塊並不區分計數命令輸入 (CU 與 CD) 的上升或下降邊緣，它必須結合 “R_TRIG” 或 “F_TRIG” 方塊來做出一個脈波計數器。

描述：

從 0 開始，每次計數加一到給予的值 (整數)，或是從給予的值 (整數) 開始，每次減一計數到 0 為止。

(* FBD 範例：使用 “CTUD” 方塊 *)



(* ST 相等式：我們假設 R_TRIG1 和 R_TRIG2 是 R_TRIG 方塊的樣例，
CTUD1 是 CTUD 方塊的樣例 *)

CTUD1(R_TRIG1(add_elt), R_TRIG2(sub_elt), reset_cmd, load_cmd,
100);

full := CTUD1.QU;

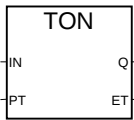
empty := CTUD1.QD;

nb_elt := CTUD1.CV;

(* IL 相等式：*)

```
LD      add_elt
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTUD1.cu
LD      sub_elt
ST      R_TRIG2.clk
CAL     R_TRIG2
LD      R_TRIG2.Q
ST      CTUD1.cd
LD      reset_cmd
ST      CTUD1.reset
LD      load_cmd
ST      CTUD1.load
LD      100
ST      CTUD1.pv
CAL     CTUD1
LD      CTUD1.QU
ST      full
LD      CTUD1.QD
ST      empty
LD      CTUD1.CV
ST      nb_elt
```

TON



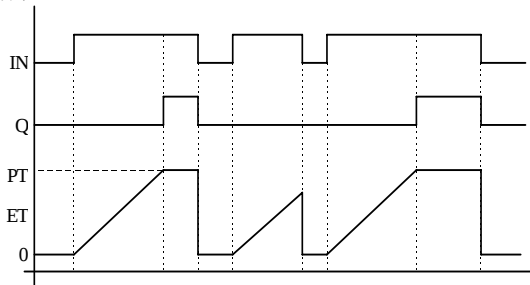
引數：

- IN** 布林 如果訊號為上升邊緣，則開始內部計時器
如果訊號為下降邊緣，則停止並重置內部計時器
- PT** 計時器 程式規劃的最大時間值
- Q** 布林 如果得真，則表示已經計時到程式規劃的時間
- ET** 計時器 目前經過的延遲時間

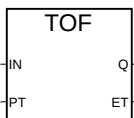
描述：

啟動內部計時器，直到所設定的延遲時間。

時序圖：



TOF



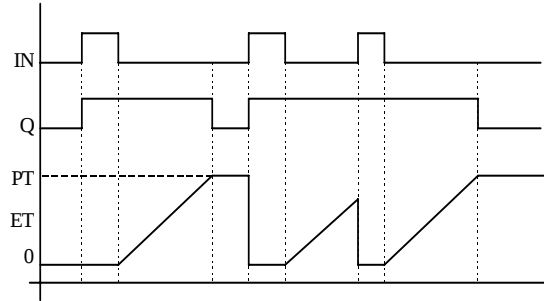
引數：

- IN** 布林 如果訊號為下降邊緣，則開始內部計時器
如果訊號為上升邊緣，則停止並重置內部計時器
- PT** 計時器 程式規劃的最大時間值
- Q** 布林 如果得真：表示時間尚未計完
- ET** 計時器 目前經過的延遲時間

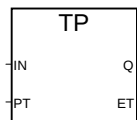
描述：

啟動內部計時器，直到所設定的延遲時間。

時序圖：



TP



引數：

IN 布林 如果訊號為上升邊緣，則開始內部計時器（如果計時器並無動作）
如果訊號為假並且計時器已計時完畢，則停止並重置內部計時器
當計時器正在計時的時候，任何對 IN 所作的變動對計時器來說
並無影響

PT 計時器 程式規劃的最大時間值

Q 布林 如果得真：表示計時器正在計時

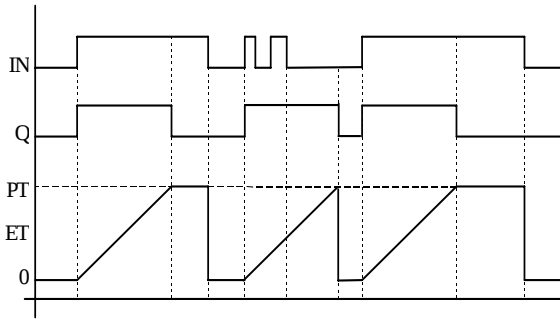
ET 計時器 目前經過的延遲時間

描述：

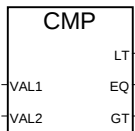
啟動內部計時器，直到所設定的延遲時間。

時序圖：

語言參考



CMP



引數：

VAL1 整數 任何含正負號的整數類比值

VAL2 整數 任何含正負號的整數類比值

LT 布林 如果 val1 小於 val2 則得真

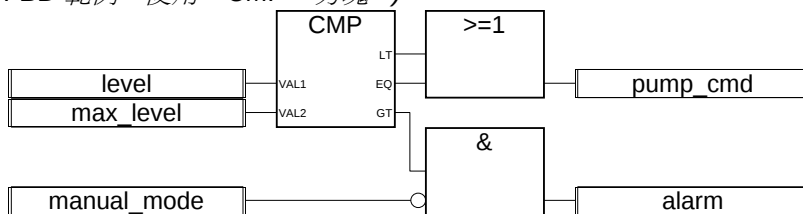
EQ 布林 如果 val1 等於 val2 則得真

GT 布林 如果 val1 大於 val2 則得真

描述：

比較兩個值：兩值是否相等，或是第一個值小於或大於第二個值。

(* FBD 範例：使用 “CMP” 方塊 *)



(* ST 相等式：我們假設 CMP1 是 CMP 方塊的樣例 *)

CMP1(level, max_level);

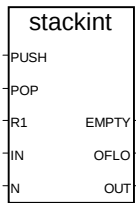
pump_cmd := CMP1.LT OR CMP1.EQ;

alarm := AMP1.GT AND NOT(manual_mode);

(* IL 相等式 : *)

```
LD      level
ST  CMP1.val1
LD  max_level
ST  CMP1.val2
CAL CMP1
LD  CMP1.LT
OR  CMP1.EQ
ST  pump_cmd
LD  CMP1.GT
ANDNmanual_mode
ST  alarm
```

STACKINT



引數 :

- PUSH** 布林 推入指令 (只有於上升邊緣時)
將IN 之值加進堆疊的最上層
- POP** 布林 取出指令 (只有於上升邊緣時)
刪除堆疊中最後一個被推進的值 (堆疊的最上層)
- R1** 布林 重置堆疊使其成為空堆疊
- IN** 整數 推入的值
- N** 整數 應用程式定義的堆疊大小
- EMPTY** 布林 如果堆疊是空的話則得真
- OFLO** 布林 溢位: 如果堆疊已滿則得真
- OUT** 整數 在堆疊最上層的值

描述:

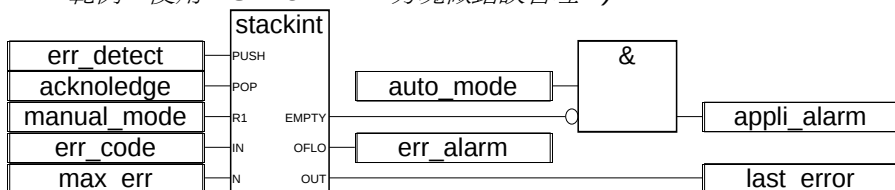
語言參考

管理整數值的堆疊。

“STACKINT” 功能方塊需要作一個上升訊號檢知用以作為 **PUSH** 和 **POP** 的輸入命令。最大的堆疊容量為 **128**。應用程式定義的堆疊容量 **N** 不能小於 **1** 或大於 **128**。

請注意 **OFLO** 的值只能在經過重置後才有作用 (**R1** 至少為真過一次，且已變回假的狀態)。

(* FBD 範例：使用 “STACKINT” 方塊做錯誤管理 *)



(* ST 相等式：我們假設 **STACKINT1** 是 **STACKINT** 方塊的樣例 *)

```
STACKINT1(err_detect, acknowledge, manual_mode, err_code, max_err);
```

```
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);
```

```
err_alarm := STACKINT1.OFLO;
```

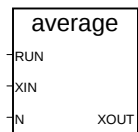
```
last_error := STACKINT1.OUT;
```

(* IL 相等式：*)

```
LD      err_detect
ST      STACKINT1.push
LD      acknowledge
ST      STACKINT1.pop
LD      manual_mode
ST      STACKINT1.r1
LD      err_code
ST      STACKINT1.IN
LD      max_err
ST      STACKINT1.N
```

CAL	STACKINT1
LD	auto_mode
ANDN	STACKINT1.empty
ST	appli_alarm
LD	STACKINT1.OFLO
ST	err_alarm
LD	STACKINT1.OUT
ST	last_error

AVERAGE



引數：

RUN 布林 真則執行 / 假則復歸

XIN 實數 任何實數類比值變數

N 整數 應用程式定義的取樣數量

XOUT 實數 對XIN 的值求其平均數

描述：

每個週期儲存一個值，並將所有已經儲存的值加以運算求其平均值。只儲存 N 個值。

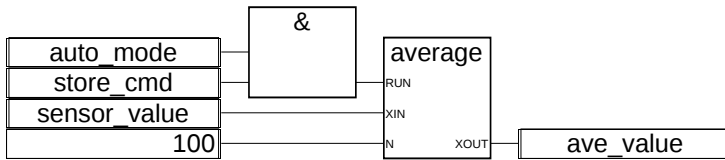
取樣數量 **N** 的數目不能超過 **128**。

如果“**RUN**”端的指令是**假**（重置模式），則輸入的值等於輸出的值。

當儲存的數目已到達最大之值 **N**，則第一個被儲存的值會被最後一個儲存值取代。

(* FBD 範例：使用“AVERAGE”方塊：*)

語言參考



(* ST 相等式：我們假設 AVERAGE1 是 AVERAGE 方塊的樣例 *)

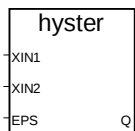
```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
```

```
ave_value := AVERAGE1.XOUT;
```

(* IL 相等式：*)

```
LD      auto_mode
AND      store_cmd
ST      AVERAGE1.run
LD      sensor_value
ST      AVERAGE1.xin
LD      100
ST      AVERAGE1.N
CAL     AVERAGE1
LD      AVERAGE1.XOUT
ST      ave_value
```

HYSTER



引數：

XIN1 實數 任何實數類比值

XIN2 實數 用以測試 XIN1 是否已超過 XIN2+EPS

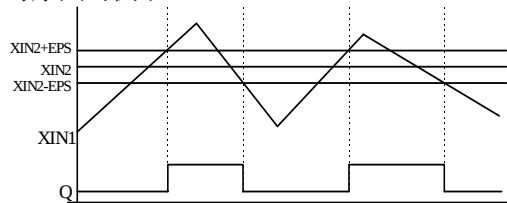
EPS 實數 遲滯值 (必須大於零)

Q 布林 如果 XIN1 大於 XIN2+EPS 並且還沒低於 XIN2-EPS 則得真

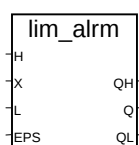
描述：

對高限幅的實數值作遲滯。

時序圖的例子：



LIM_ALRM



引數：

H 實數 高限制值

X 實數 輸入：任何實數類比值

L 實數 低限制值

EPS 實數 遲滯值 (必須大於零)

QH 布林 “高” 警告：如果 X 在高限制 H 之上則得真

Q 布林 警告輸出：如果 X 超出限制則得真

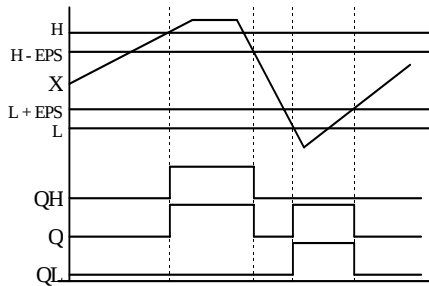
QL 布林 “低” 警告：如果 X 低於低限制 H 則得真

描述：

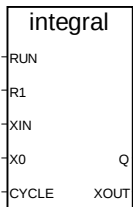
對高和低限制的實數值作遲滯。

可以對高和低限制作遲滯。高和低限制的遲滯 δ 是 EPS 參數的一半。下面是時序圖的例子：

語言參考



INTEGRAL



引數：

RUN 布林 模式：真則積分 / 假則保持

R1 布林 溢位重置

XIN 實數 輸入：任何實數值

X0 實數 初始值

CYCLE 計時器 取樣週期

Q 布林 R1 的反相

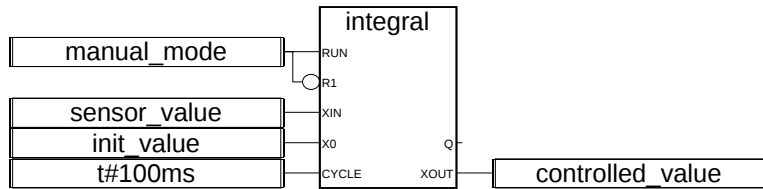
XOUT 實數 積分後的輸出

描述：

實數值的積分。

如果“**CYCLE**”的參數值小於 ISaGRAF 應用程式的週期時間，則取樣週期的時間將被應用程式的週期時間取代。

(* FBD 範例：使用“INTEGRAL”方塊 *)



(* ST 相等式：我們假設 INTEGRAL1 是 INTEGRAL 方塊的樣例 *)

```

INTEGRAL1(manual_mode,    NOT(manual_mode),    sensor_value,
init_value, t#100ms);
controlled_value := INTEGRAL1.XOUT;

```

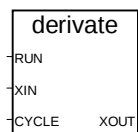
(* IL 相等式：*)

```

LD      manual_mode
ST      INTEGRAL1.run
STN     INTEGRAL1.R1
LD      sensor_value
ST      INTEGRAL1.XIN
LD      init_value
ST      INTEGRAL1.X0
LD      t#100ms
ST      INTEGRAL1.CYCLE
CAL     INTEGRAL1
LD      INTEGRAL1.XOUT
ST      controlled_value

```

DERIVATE



引數：

RUN 布林 模式：真則正常 / 假則重置

XIN 實數 輸入：任何實數值

CYCLE 計時器 取樣週期

語言參考

XOUT 實數 微分輸出

描述：

實數值的微分。

如果“**CYCLE**”的參數值小於 *ISaGRAF* 應用程式的週期時間，則取樣週期的時間將被應用程式的週期時間取代。

(* FBD 範例：使用 “**DERIVATE**” 方塊 *)



(* ST 相等式：我們假設 **DERIVATE1** 是 **DERIVATE** 方塊的樣例 *)

DERIVATE1(manual_mode, sensor_value, t#100ms);

derivated_value := **DERIVATE1**.XOUT;

(* IL 相等式：*)

```
LD      manual_mode
ST      DERIVATE1.run
LD      sensor_value
ST      DERIVATE1.XIN
LD      t#100ms
ST      DERIVATE1.CYCLE
CAL     DERIVATE1
LD      DERIVATE1.XOUT
ST      derivated_value
```

BLINK



引數：

RUN 布林 模式：真則閃爍 / 假則將輸出重置為假

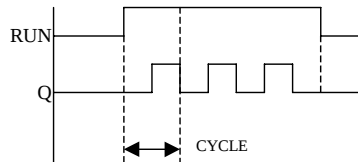
CYCLE 計時器 閃爍週期

Q 布林 輸出的閃爍訊號

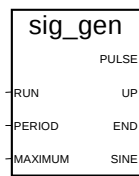
描述：

產生閃爍訊號。

時序圖：



SIG_GEN



引數：

RUN 布林 模式：真則執行 / 假則復歸

PERIOD 計時器 取樣的持續時間

MAXIMUM 整數 最大的計數值

PULSE 布林 每次取樣之後將被反相

UP 整數 向上計數器，每次取樣都會增加

END 布林 當向上計數完成時得真

SINE 實數 正弦訊號 (週期=計數週期)

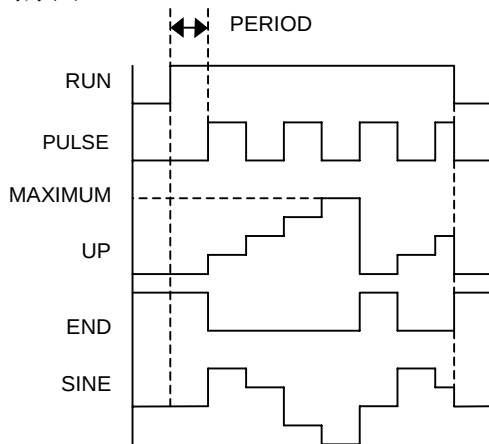
描述：

產生多種的訊號：布林的閃爍訊號、整數的向上計數訊號以及正弦波。

當計數到達最大值時，計數器將重新從零開始計數。因此 **END** 的值只能保持一個 **PERIOD** 的時間。

語言參考

時序圖：



B.9.3 標準的函數

下面是由 ISaGRAF 系統所提供的標準的函式。這些函式是已經被定義過，且不需要在程式庫中宣告。

數學 ABS 絕對值

EXPT, POW 指數，乘冪運算

LOG 對數

SQRT 平方根

TRUNC 刪除小數部份

三角運算 ACOS, ASIN, 反餘弦，反正弦，

ATAN 反正切

COS, SIN, 餘弦，正弦，

TAN 正切

暫存器控制 ROL, ROR 左旋，右旋

SHL, SHR 左移，右移

資料處理 MIN, MAX 最小值，最大值，

LIMIT 限制

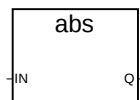
MOD 餘數

MUX4, MUX8 多路轉換器 (4 個或 8 個輸入)

SEL 二進位選擇器

ODD 奇數校驗
 RAND 隨機值
 資料轉換 ASCII 字元→ASCII 碼
 CHAR ASCII 碼→字元
 字串管理 MLEN 取得字串長度
 DELETE 刪除子字串
 INSERT 插入字串
 FIND 尋找子字串
 REPLACE 取代子字串
 LEFT, MID 擷取字串的左邊字串，中間字串
 RIGHT 擷取字串的右邊字串
 DAY_TIME 日期與時間
 陣列管理 ARCREATE 開啓整數值的陣列
 ARREAD 讀取陣列元素
 ARWRITE 寫入陣列元素
 二進位檔案管理 F_ROPEN 在讀取模式下開啓二進位檔案
 F_WOPEN 在寫入模式下開啓二進位檔案
 F_CLOSE 關閉二進位檔案
 F_EOF 測試二進位檔案結尾
 FA_READ 在二進位檔案中讀取類比值
 FA_WRITE 寫入類比值到二進位檔案中
 FM_READ 在二進位檔案中讀取字串
 FM_WRITE 寫入字串到二進位檔案中

ABS



引數：

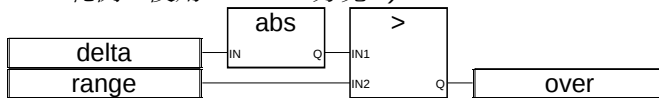
IN 實數 任何含正負號的實數類比值
Q 實數 絕對值 (永遠為正值)

描述：

語言參考

產生一實數值的絕對值 (正值)。

(* FBD 範例：使用 “ABS” 方塊 *)



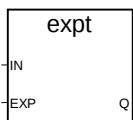
(* ST 相等式：*)

over := (ABS (delta) > range);

(* IL 相等式：*)

LD delta
ABS
GT range
ST over

EXPT



引數：

IN 實數 任何含正負號的實數類比值

EXP 整數 整數類比值的指數

Q 實數 (IN^{EXP})

描述：

產生此運算後的實數結果：(基底^{指數}) ‘基底’ 當成第一個參數，而 ‘指數’ 則為第二個。

(* FBD 範例：使用 “EXPT” 方塊 *)



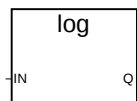
(* ST 相等式 : *)

tb_size := ANA (EXPT (2.0, range));

(* IL 相等式 : *)

LD 2.0
EXPT range
ANA
ST tb_size

LOG



引數：

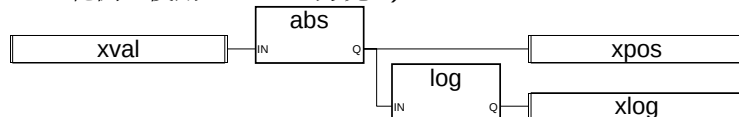
IN 實數 必須大於零

Q 實數 輸入值的對數 (基底 10)

描述：

實數值的對數 (基底 10) 運算。

(* FBD 範例：使用 “LOG” 方塊 *)



(* ST 相等式 : *)

xpos := ABS (xval);
xlog := LOG (xpos);

(* IL 相等式 : *)

LD xval
ABS
ST xpos
LOG

語言參考

ST xlog

POW



引數：

IN 實數 將被加入的實數類比數字

EXP 實數 乘幂 (exponent)

Q 實數 (IN^{EXP})

如果 **IN** 不為 0.0 且 **EXP** 為 0.0 則得 1.0

如果 **IN** 為 0.0 且 **EXP** 為負數則得 0.0

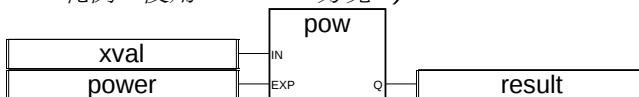
如果 **IN** 為 0.0 且 **EXP** 為 0.0 則得 0.0

如果 **IN** 為負數且 **EXP** 不為整數則得 0.0

描述：

產生此運算後的實數結果： $(\text{基底}^{\text{指數}})$ ‘基底’ 當成第一個參數，而 ‘指數’ 則為第二個。‘指數’ 為實數值。

(* FBD 範例：使用 “POW” 方塊 *)



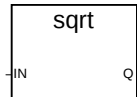
(* ST 相等式：*)

`result := POW (xval, power);`

(* IL 相等式：*)

LD xval
POW power
ST result

SQRT



引數：

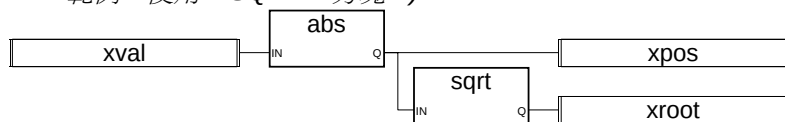
IN 實數 必須大於零或是等於零

Q 實數 輸入值的平方根

描述：

計算實數值的平方根。

(* FBD 範例：使用 “SQRT” 方塊 *)



(* ST 相等式：*)

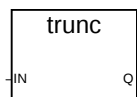
`xpos := ABS (xval);`

`xroot := SQRT (xpos);`

(* IL 相等式：*)

```
LD      xval
ABS
ST      xpos
SQRT
ST      xroot
```

TRUNC



引數：

IN 實數 任何實數類比值

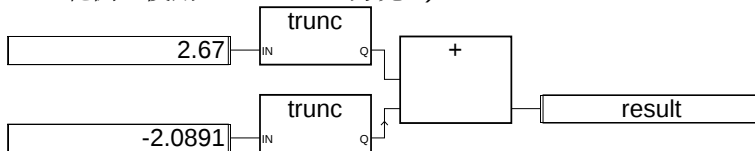
Q 實數 如果 $IN > 0$ ，則為小於或等於輸入值的最大整數
如果 $IN < 0$ ，則為大於或等於輸入值的最小整數

語言參考

描述：

將實數值的小數部份忽略，只保留其整數部份。

(* FBD 範例：使用 “TRUNC” 方塊 *)



(* ST 相等式：*)

`result := TRUNC (+2.67) + TRUNC (-2.0891);`

(* 意思是：`result := 2.0 + (-2.0) := 0.0;` *)

(* IL 相等式：*)

LD 2.67

TRUNC

ST temporary (* 第一次 TRUNC 的暫時結果 *)

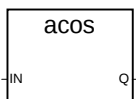
LD -2.0891

TRUNC

ADD temporary

ST result

ACOS



引/數：

IN 實數 必須確定在 $[-1.0 \dots +1.0]$

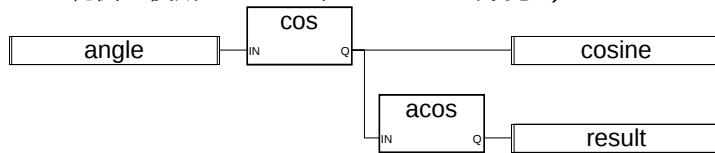
Q 實數 對輸入值作反餘弦 (確定於 $[0.0 \dots \pi]$)

對無效的輸入值則等於 0.0

描述：

對實數值作反餘弦運算。

(* FBD 範例：使用 “COS” 和 “ACOS” 方塊 *)



(* ST 相等式 : *)

`cosine := COS (angle);`

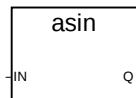
`result := ACOS (cosine); (* result := angle; *)`

(* IL 相等式 : *)

```

LD      angle
COS
ST      cosine
ACOS
ST      result
  
```

ASIN



引數：

IN 實數 必須確定在 $[-1.0 \dots +1.0]$

Q 實數 對輸入值作反正弦 (確定於 $[-\pi/2 \dots +\pi/2]$)

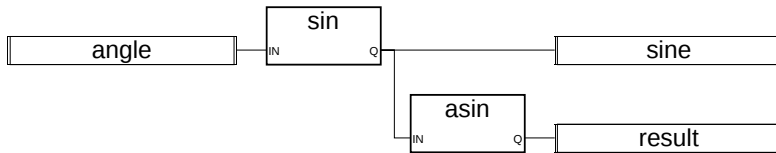
對無效的輸入值則等於 0.0

描述：

對實數值作反正弦運算。

(* FBD 範例：使用 “SIN” 和 “ASIN” 方塊 *)

語言參考



(* ST 相等式 : *)

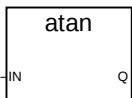
sine := SIN (*angle*);

result := ASIN (*sine*); (* *result* := *angle*; *)

(* IL 相等式 : *)

LD *angle*
SIN
ST *sine*
ASIN
ST *result*

ATAN



引數：

IN 實數 任何實數類比值

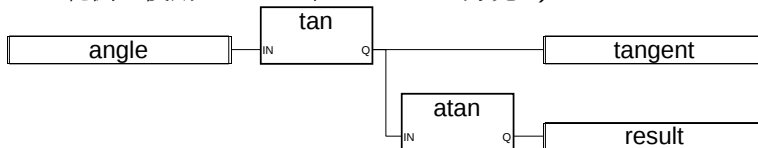
Q 實數 對輸入值作反正切 (確定於 $[-\pi/2 \dots +\pi/2]$)

對無效的輸入值則等於 0.0

描述：

對實數值作反正切運算。

(* FBD 範例：使用 “TAN” 和 “ATAN” 方塊 *)



(* ST 相等式 : *)

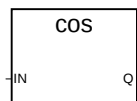
tangent := TAN (*angle*);

$result := ATAN(tangent); (* result := angle; *)$

(* IL 相等式 : *)

LD angle
TAN
ST tangent
ATAN
ST result

COS



引數：

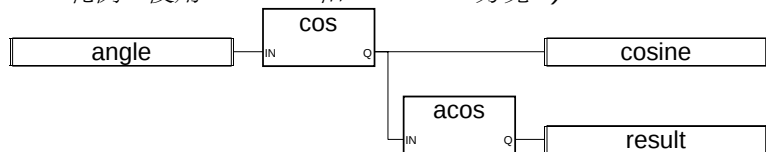
IN 實數 任何實數類比值

Q 實數 對輸入值作餘弦 (確定於 [-1.0...+1.0])

描述：

對實數值作餘弦運算。

(* FBD 範例：使用 “COS” 和 “ACOS” 方塊 *)



(* ST 相等式 : *)

$cosine := COS(angle);$

$result := ACOS(cosine); (* result := angle; *)$

(* IL 相等式 : *)

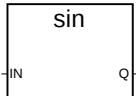
LD angle
COS
ST cosine

語言參考

ACOS

ST *result*

SIN



引數：

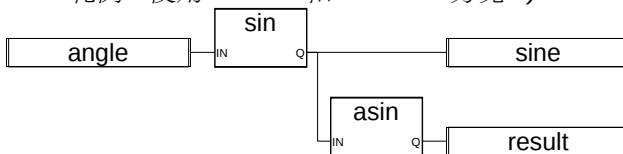
IN 實數 任何實數類比值

Q 實數 對輸入值作正弦 (須訂於 [-1.0...+1.0])

描述：

對實數值作正弦運算。

(* FBD 範例：使用 “SIN” 和 “ASIN” 方塊 *)



(* ST 相等式：*)

sine := SIN (*angle*);

result := ASIN (*sine*); (* *result* := *angle*; *)

(* IL 相等式：*)

LD *angle*

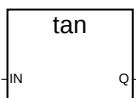
SIN

ST *sine*

ASIN

ST *result*

TAN



引數：

IN 實數 不能等於 $\pi/2$ 至 π

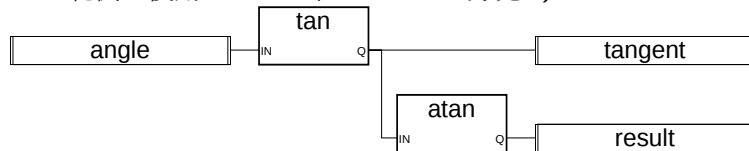
Q 實數 對輸入值作正切

對無效的輸入值則等於 $1E+38$

描述：

對實數值作正切運算。

(* FBD 範例：使用 “TAN” 和 “ATAN” 方塊 *)



(* ST 相等式：*)

`tangent := TAN (angle);`

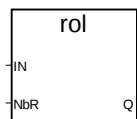
`result := ATAN (tangent); (* result := angle; *)`

(* IL 相等式：*)

```

LD      angle
TAN
ST      tangent
ATAN
ST      result
  
```

ROL



引數：

IN 整數 任何整數類比值

NbR 整數 位元旋轉的位數 (須介於 [1....31])

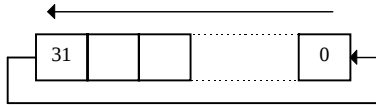
Q 整數 向左旋轉後的值

如果 $\text{NbR} \leq 0$ 的話是沒有作用的

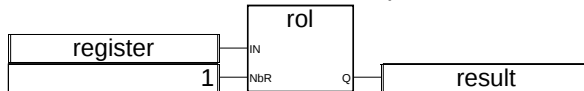
語言參考

描述：

使整數中的位元向左旋轉，整個整數 32 位元皆一起旋轉：



(* FBD 範例：使用 “ROL” 方塊 *)



(* ST 相等式：*)

result := ROL (register, 1);

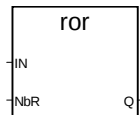
(* register = 2#0100_1101_0011_0101 *)

(* result = 2#1001_1010_0110_1010 *)

(* IL 相等式：*)

```
LD      register
ROL 1
ST      result
```

ROR



引數：

IN 整數 任何整數類比值

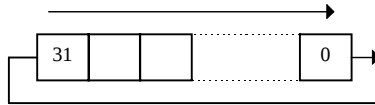
NbR 整數 位元旋轉的位數 (須位於 [1...31])

Q 整數 向右旋轉後的值

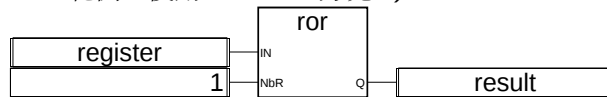
如果 NbR ≤ 0 的話是沒有作用的

描述：

使整數中的位元向右旋轉，整個整數 32 位元皆一起旋轉：



(* FBD 範例：使用 “ROR” 方塊 *)



(* ST 相等式：*)

result := ROR (register, 1);

(* register = 2#0100_1101_0011_0101 *)

(* result = 2#1010_0110_1001_1010 *)

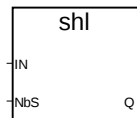
(* IL 相等式：*)

LD register

ROR 1

ST result

SHL



引數：

IN 整數 任何整數類比值

NbS 整數 位元移位的位數 (須介於 [1....31])

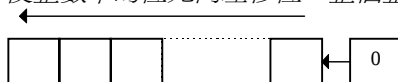
Q 整數 向左移位後的值

如果 NbS ≤ 0 的話是沒有作用的

0 用來置換最低的位元

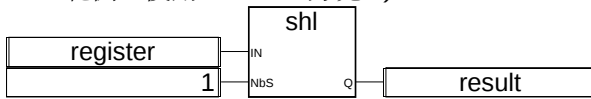
描述：

使整數中的位元向左移位，整個整數 32 位元皆一起旋轉：



語言參考

(* FBD 範例：使用 “SHL” 方塊 *)



(* ST 相等式：*)

result := SHL (register, 1);

(* register = 2#0100_1101_0011_0101 *)

(* result = 2#1001_1010_0110_1010 *)

(* IL 相等式：*)

```
LD      register
SHL 1
ST      result
```

SHR



引數：

IN 整數 任何整數類比值

NbS 整數 位元移位的位數 (須介於 [1...31])

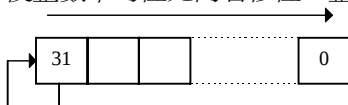
Q 整數 向右移位後的值

如果 NbS ≤ 0 的話是沒有作用的

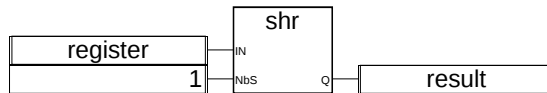
最高的位元在每次移位時都會被複製

描述：

使整數中的位元向右移位，整個整數 32 位元皆一起旋轉：



(* FBD 範例：使用 “SHR” 方塊 *)



(* ST 相等式 : *)

result := SHR (*register*, 1);

(* *register* = 2#1100_1101_0011_0101 *)

(* *result* = 2#1110_0110_1001_1010 *)

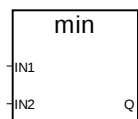
(* IL 相等式 : *)

LD *register*

SHR 1

ST *result*

MIN



引數：

IN1 整數 任何帶正負號的整數類比值

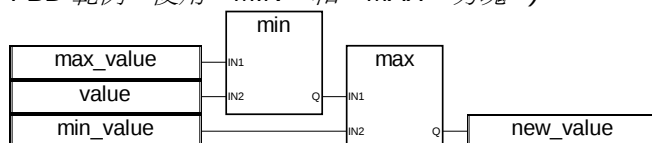
IN2 整數 (不能為實數)

Q 整數 兩個輸入值中的最小值

描述：

在兩個整數值中產生最小的值。

(* FBD 範例：使用 “MIN” 和 “MAX” 方塊 *)



(* ST 相等式 : *)

new_value := MAX (MIN (*max_value*, *value*), *min_value*);

語言參考

(* 將值限制在 [min_value...max_value] 間 *)

(* IL 相等式 : *)

LD max_value

MIN value

MAX min_value

ST new_value

MAX



引數：

IN1 整數 任何帶正負號的整數類比值

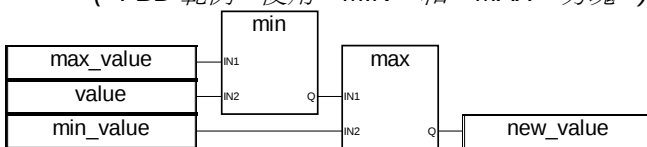
IN2 整數 (不能為實數)

Q 整數 兩個輸入值中的最大值

描述：

在兩個整數值中產生最大的值。

(* FBD 範例：使用 “MIN” 和 “MAX” 方塊 *)



(* ST 相等式 : *)

new_value := MAX (MIN (max_value, value), min_value);

(* 將值限制在 [min_value...max_value] 間 *)

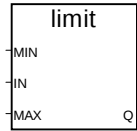
(* IL 相等式 : *)

LD max_value

MIN value

MAX min_value
ST new_value

LIMIT



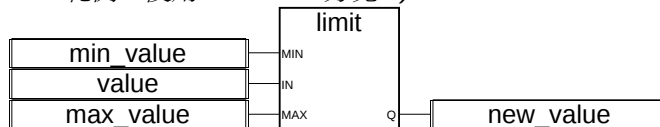
引數：

MIN 整數 最小的允許值
IN 整數 任何帶正負號的整數類比值
MAX 整數 最大的允許值
Q 整數 被限制在允許範圍內的輸入值

描述：

將整數值限定在一個給予的範圍內。如果值介於最小與最大之間則保留其值，或是如果超過最大值則將值變成最大值，或是如果低於最小值則將值變成最小值。

(* FBD 範例：使用 “LIMIT” 方塊 *)



(* ST 相等式：*)

new_value := LIMIT (min_value, value, max_value);

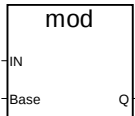
(* 將值限制在 [min_value...max_value] 間 *)

(* IL 相等式：*)

LD min_value
 LIMIT value,max_value
 ST new_value

MOD

語言參考



引數：

IN 整數 任何帶正負號的整數類比值

Base 整數 必須大於零

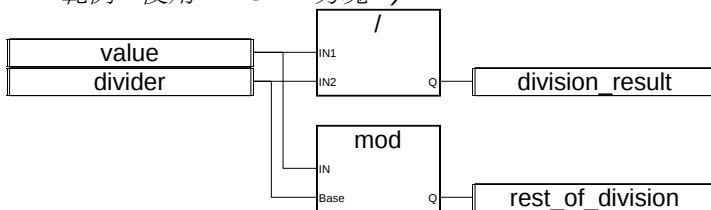
Q 整數 餘數計算 (輸入 MOD 基礎)

如果 $\text{Base} \leq 0$ 則返回-1

描述：

對整數值作餘數運算。

(* FBD 範例：使用 “MOD” 方塊 *)



(* ST 相等式：*)

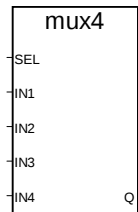
$\text{division_result} := (\text{value} / \text{divider});$ (* 整數除法 *)

$\text{rest_of_division} := \text{MOD}(\text{value}, \text{divider});$ (* 相除後的餘數 *)

(* IL 相等式：*)

```
LD      value
DIV     divider
ST      division_result
LD      value
MOD     divider
ST      rest_of_divison
```

MUX4



引數：

SEL 整數 整數值選取器 (必須設定在 [0...3] 間)

IN1...IN4 整數 任何整數類比值

Q 整數 if SEL = 0 則 = IN1

if SEL = 1 則 = IN2

if SEL = 2 則 = IN3

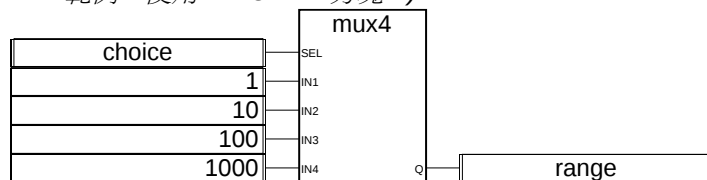
if SEL = 3 則 = IN4

如果選取器收到0....3 以外的值則得零

描述：

含 4 個輸入的多路轉換器：從四個整數值中選取一值。

(* FBD 範例：使用 “MUX4” 方塊 *)



(* ST 相等式：*)

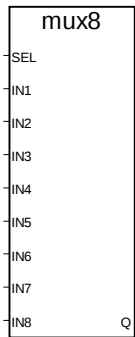
`range := MUX4 (choice, 1, 10, 100, 1000);`

(* 從四個預先定義的範圍內選取一個，例如，如果 choice 是 1，則 range 為 10 *)

(* IL 相等式：*)

```
LD          choice
            MUX4 1,10,100,1000
            ST   range
```

MUX8



引數：

SEL 整數 整數值選取器 (必須設定在 [0...7] 間)

IN1...IN8 整數 任何整數類比值

Q 整數 if SEL = 0 則 = IN1

if SEL = 1 則 = IN2

...

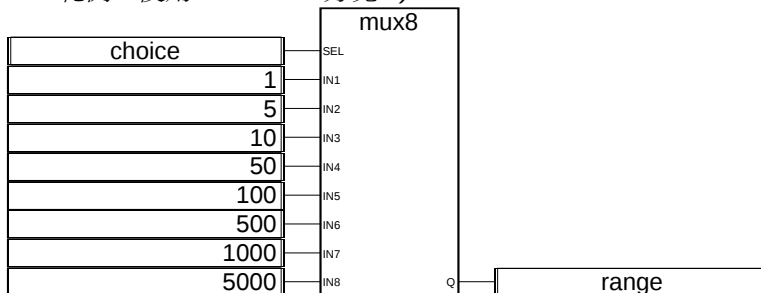
if SEL = 7 則 = IN8

如果選取器收到 0...7 以外的值則得零

描述：

含 8 個輸入的多路轉換器：從八個整數值中選取一值。

(* FBD 範例：使用 “MUX8” 方塊 *)



(* ST 相等式：*)

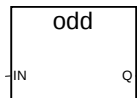
range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);

(* 從八個預先定義的範圍內選取一個，例如，如果 choice 是 3，則 range 為 50 *)

(* IL 相等式 : *)

LD choice
MUX8 1,5,10,50,100,500,1000,5000
ST range

ODD



引數：

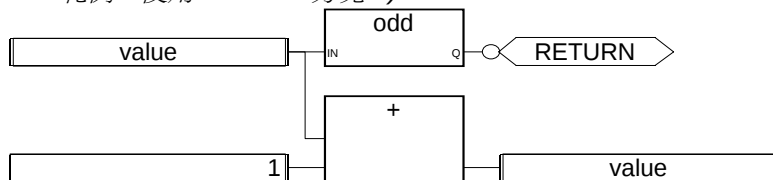
IN 整數 任何含正負號的整數類比值

Q 布林 如果輸入值為奇數則得真
 如果輸入值為偶數則得假

描述：

測試整數的奇偶：結果是奇數或偶數。

(* FBD 範例：使用 “ODD” 方塊 *)



(* ST 相等式 : *)

If NOT (ODD (value)) THEN Return; End_if;

value := value + 1;

(* 使值永遠為偶數 *)

(* IL 相等式 : *)

LD value

語言參考

ODD

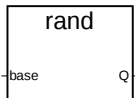
RETNC

LD value

ADD 1

ST value

RAND



引數：

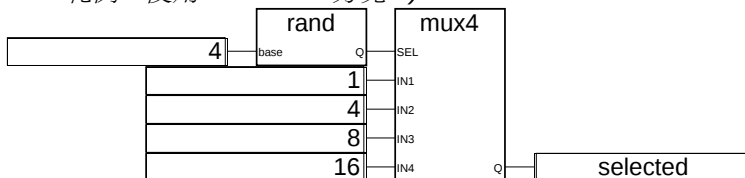
base 整數 定義允許數字的範圍

Q 整數 設定範圍內的亂數值

描述：

在允許的範圍內產生亂數值。

(* FBD 範例：使用 “RAND” 方塊 *)



(* ST 相等式：*)

```
selected := MUX4 ( RAND (4), 1, 4, 8, 16 );
```

(*

隨意從已定義的四個值中選取一值

從RAND 的輸出值設定為 [0...3] 之間，

因此從MUX4 得到的值，將會到是下列四個值中的其中一個任意值

如果從RAND 得到的值為 0，則selected:=1;

如果從RAND 得到的值為 1，則selected:=4;

如果從RAND 得到的值為 2，則selected:=8;

如果從RAND 得到的值為 3，則selected:=16;

*)

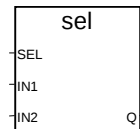
(* IL 相等式 : *)

LD 4

RAND

MUX41,4,8,16

ST selected

SEL

引數：

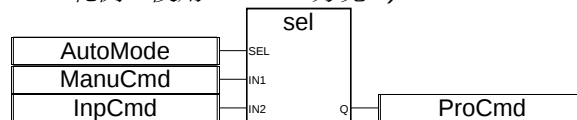
SEL 布林 指定選取值**IN1,IN2** 整數 任何整數類比值

Q 整數 如果 SEL 為假則得 IN1
 如果 SEL 為真則得 IN2

描述：

二元選取器：從兩個整數值中選取一值。

(* FBD 範例：使用 “SEL” 方塊 *)



(* ST 相等式 : *)

ProCmd := SEL (AutoMode, ManuCmd, InpCmd);

(* 處理指令的選取 *)

(* IL 相等式 : *)

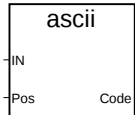
LD AutoMode

語言參考

SEL ManuCmd, InpCmd

ST ProCmd

ASCII



引數：

IN 訊息 任何不是空的字串

Pos 整數 所選字元的位置

其範圍是 [0...字串長度]

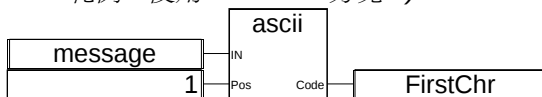
Code 整數 選取位元的編碼 (其值為 [0...255] 間)

如果返回值為 0 就表示 Pos 超出字串的範圍

描述：

在字串中對字元產生其 ASCII 碼。

(* FBD 範例：使用 “ASCII” 方塊 *)



(* ST 相等式：*)

FirstChr := ASCII (message, 1);

(* FirstChr 是此字串中第一個字元的 ASCII 碼 *)

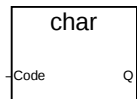
(* IL 相等式：*)

LD message

ASCII 1

ST FirstChr

CHAR



引數：

Code 整數 介於 [0...255] 間的碼

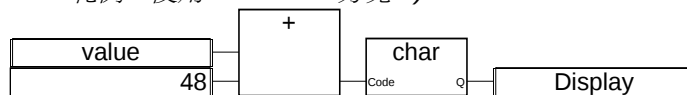
Q 訊息 字元字串

輸出字元為輸入 ASCII 碼的轉換
(ASCII 碼為 256 的餘數)

描述：

從給予的 ASCII 碼中產生一相對應的字元。

(* FBD 範例：使用 “CHAR” 方塊 *)



(* ST 相等式：*)

`Display := CHAR ( value + 48 );`

(* value 的值設為 [0...9] 之間 *)

(* 48 的 ASCII 碼為 '0' *)

(* 結果為從 '0' 到 '9' 的字元字串 *)

(* IL 相等式：*)

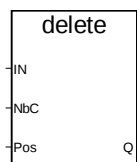
`LD value`

`ADD 48`

`CHAR`

`ST Display`

DELETE



語言參考

引數：

IN 訊息 任何非空字串

NbC 整數 欲刪除的字元數量

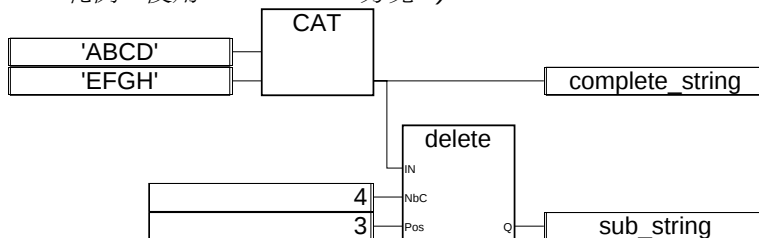
Pos 整數 欲刪除第一個字元的位置
(字串中的第一個字元的位置為 1)

Q 訊息 經修改後的字串
如果 $Pos < 1$ 則得空字串
如果 $Pos > IN$ 字串長度，則得原始字串
如果 $NbC \leq 0$ 則得原始字串

描述：

刪除字串的一部份。

(* FBD 範例：使用 “DELETE” 方塊 *)



(* ST 相等式：*)

$complete_string := 'ABCD' + 'EFGH';$ (* complete_string 為 'ABCDEFGH'

*)

$sub_string := DELETE (complete_string, 4, 3);$ (* sub_string 為 'ABGH' *)

(* IL 相等式：*)

LD 'ABCD'

ADD 'EFGH'

ST complete_string

DELETE 4,3

ST sub_string

FIND

引數：

In 訊息 任何字串

Pat 訊息 任何非空字串 (樣品)

Pos 整數 如果找不到 **Pat** 的字串，則得 0

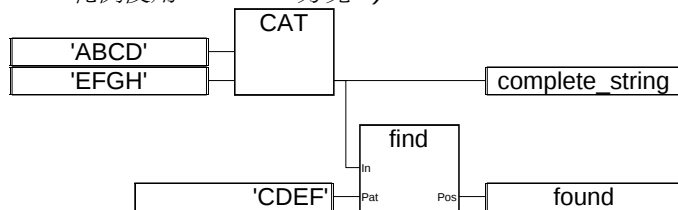
=**Pat** 字串中第一個出現的第一個字元的位置
(第一個位置為 1)

此功能不分大小寫

描述：

尋找字串中的子字串。得到此子字串在字串中的位置。

(* FBD 範例使用 “FIND” 方塊 *)



(* ST 相等式 : *)

`complete_string := 'ABCD' + 'EFGH';` (* complete_string 為 'ABCDEFGH' *)

`found := FIND (complete_string, 'CDEF');` (* found 為 3 *)

(* IL 相等式 : *)

`LD 'ABCD'`

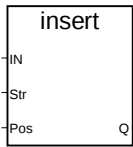
`ADD 'EFGH'`

`ST complete_string`

`FIND 'CDEF'`

`ST found`

INSERT



引數：

IN 訊息 原始字串

Str 訊息 欲插入字串

Pos 整數 欲插入的位置

字串將被插入在設定的位置之前

(第一個允許的位置為 1)

Q 訊息 經修改後的字串

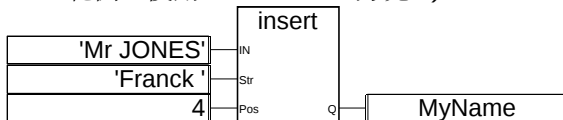
如果 $Pos \leq 0$ 則得空字串

如果 Pos 的值大於 IN 的字串長度則字串將會互相連結

描述：

在一字串中插入另一個字串到給予的位置中。

(* FBD 範例：使用 “INSERT” 方塊 *)



(* ST 相等式：*)

$MyName := INSERT('Mr JONES', 'Frank', 4);$

(* $MyName$ 為 'Mr Frank JONES' *)

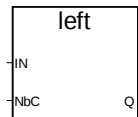
(* IL 相等式：*)

$LD \quad 'Mr JONES'$

$INSERT \quad 'Frank', 4$

$ST \quad MyName$

LEFT



引數：

IN 訊息 任何非空字串

NbC 整數 欲被取出的字元數量

不能超過輸入字串的長度

Q 訊息 輸入字串的左邊部份(其長度=NbC)

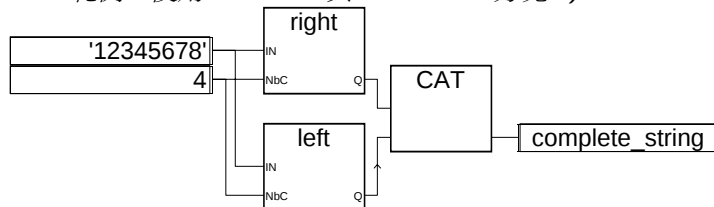
如果 NbC ≤ 0，則得空字串

如果 NbC ≥ IN 的字串長度，則得到完整的輸入字串

描述：

取出字串的左邊部份，其欲取出字元的數目是設計者給的。

(* FBD 範例：使用 “LEFT” 與 “RIGHT” 方塊 *)



(* ST 相等式：*)

`complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);`

(* complete_string 為 '56781234')

從 RIGHT 呼叫的值結果為 '5678'

從 LEFT 呼叫的值結果為 '1234'

*)

(* IL 相等式：先作呼叫 LEFT 的部份 *)

LD '12345678'

LEFT 4

ST sub_string (* 中繼結果 *)

LD '12345678'

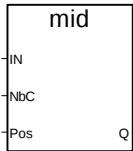
RIGHT 4

語言參考

ADD sub_string

ST complete_string

MID



引數：

IN 訊息 任何非空字串

NbC 整數 欲被取出的字元數量
不能超過輸入字串的長度

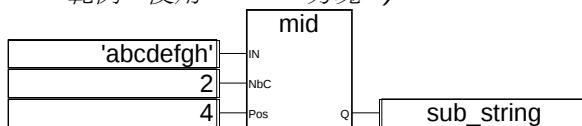
Pos 整數 子字串的位置
子字串的第一個字元將是 Pos 所指的字元
(第一個允許位置為 1)

Q 訊息 字串的中間部份 (其長度=NbC)
如果下的參數不被允許，則會得到一空字串

描述：

取出一字串的左邊部份，其欲取出字元的數目和第一個字元的位置是設計者給的。

(* FBD 範例：使用 “MID” 方塊 *)



(* ST 相等式：*)

sub_string := MID ('abcdefgh', 2, 4);

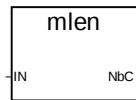
(* sub_string 為 'de' *)

(* IL 相等式：*)

LD 'abcdefgh'

MID 2,4

ST sub_string

MLEN

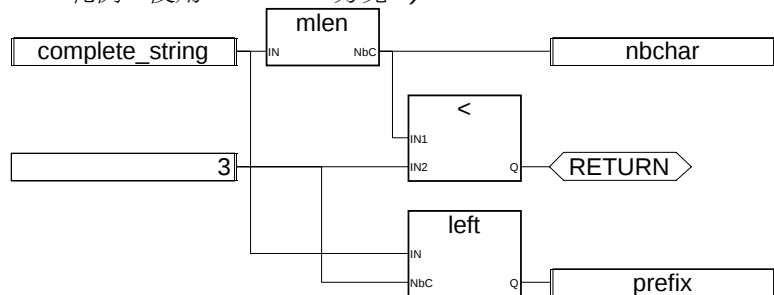
引數：

IN 訊息 任何字串**NbC** 整數 輸入字串的字元數量

描述：

計算一字串的長度。

(* FBD 範例：使用 “MLEN” 方塊 *)



(* ST 相等式：*)

nbchar := MLEN (complete_string);

if (nbchar < 3) Then Return; End_if;

prefix := LEFT (complete_string, 3);

(* 此段程式將字串左邊的三個字元取出，並將結果指定給 prefix 這個訊息變數。如果字串長度小於三個字元，則將不會有任何作用。*)

(* IL 相等式：*)

LD complete_string

MLEN

ST nbchar

LT 3

語言參考

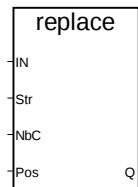
RETC

LD complete_string

LEFT 3

ST prefix

REPLACE



引數：

IN 訊息 任何字串

Str 訊息 欲插入的字串 (將置換 NbC 個字元)

NbC 整數 欲刪除的字元數目

Pos 整數 第一個被更改的字元位置

(第一個允許的位置是 1)

Q 訊息 修改後的字串

-於位置 Pos 處刪除 NbC 個字元

-然後子字串 Str 被插入到位置 Pos

如果 Pos <= 0 則返回一空字串

如果 Pos 的值大於 IN 字串的長度則傳回兩個字串的結合

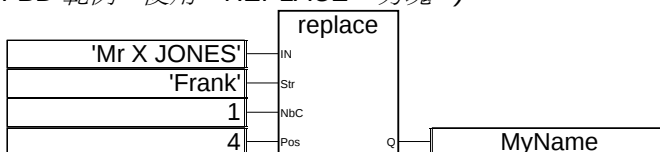
(IN+Str)

如果 NbC <= 0 則傳回原始字串

描述：

將字串中的一部份置換為新設定的字元。

(* FBD 範例：使用 “REPLACE” 方塊 *)



(* ST 相等式 : *)

MyName := REPLACE ('Mr X JONES', 'Frank', 1, 4);

(* MyName 為 'Mr Frank JONES' *)

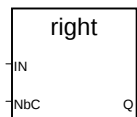
(* IL 相等式 : *)

LD 'Mr X JONES'

REPLACE 'Frank',1,4

ST MyName

RIGHT



引數：

IN 訊息 任何非空字串

NbC 整數 不能超過輸入字串的長度

Q 訊息 輸入字串的右邊部份(其長度=NbC)

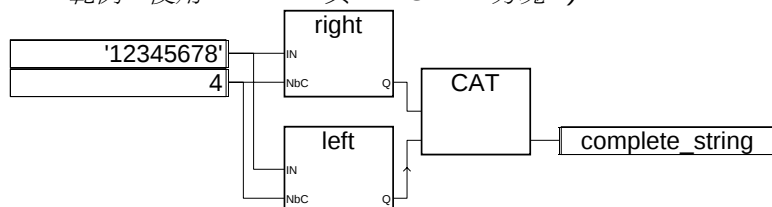
如果 NbC ≤ 0，則得一空字串

如果 NbC ≥ IN 的字串長度，則得到完整的輸入字串

描述：

取出字串的右邊部份，其欲取出字元的數目是設計者給的。

(* FBD 範例：使用 “LEFT” 與 “RIGHT” 方塊 *)



(* ST 相等式 : *)

complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);

(* complete_string 為 '56781234' *)

語言參考

從RIGHT 呼叫的值結果為 '5678'

從LEFT 呼叫的值結果為 '1234'

*)

(* IL 相等式：先作呼叫LEFT 的部份 *)

LD '12345678'

LEFT 4

ST sub_string(* 中繼結果 *)

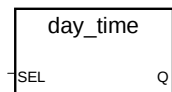
LD '12345678'

RIGHT 4

ADD sub_string

ST complete_string

DAY_TIME



引數：

SEL 整數 輸出選擇

0=取得正確日期

1=取得正確時間

2=取得星期幾

Q 訊息 時間/日期以字串表示

如果 SEL=0 則得 'YYYY/MM/DD'

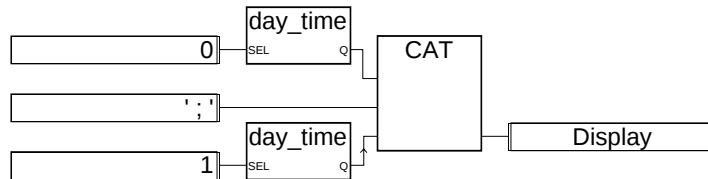
如果 SEL=1 則得 'HH:MM:SS'

如果 SEL=2 則得日期名稱(例：'Monday')

描述：

將日期或時間以字串來表示。

(* FBD 範例：使用 "DAY_TIME" 方塊 *)



(* ST 相等式 : *)

`Display := DAY_TIME (0) + ';' + DAY_TIME (1);`

(* Display 的輸出格式為 : 'YYYY/MM/DD ; HH:MM:SS' *)

(* IL 相等式 : 先作呼叫 day_time(1)的部份 *)

LD 1

DAY_TIME

ST hour_str (* 中繼結果 *)

LD 0

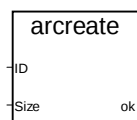
DAY_TIME

ADD ';'

ADD hour_str

ST Display

ARCREATE



引數 :

ID 整數 矩陣的編號 (必須設定在 [0...15] 間)

Size 整數 矩陣內的元素數量

ok 整數 執行狀態 :

1 = 如果已成功創造一矩陣

2 = 不允許的矩陣數量或矩陣已創造

3 = 不允許的元素數量

4 = 記憶體不足

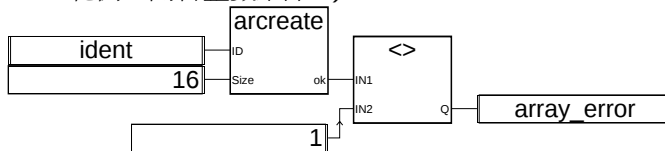
語言參考

描述：

開啓整數矩陣。

注意：應用程式最多只能容下 **16** 個矩陣。矩陣包含**整數類比值**。就如同動態配置記憶體一樣，如果矩陣的大小太接近可利用的記憶體大小的話，此函式可能會造成系統錯誤。

(* FBD 範例：開啓整數矩陣 *)



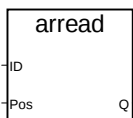
(* ST 相等式：*)

```
array_error := (ARCREATE (ident, 16) <> 1);
```

(* IL 相等式：*)

```
LD    ident
ARCREATE    16
NE    1
ST    array_error
```

ARREAD



引數：

ID 整數 矩陣的編號 (必須設定在 [0...15] 間)

Pos 整數 矩陣內的元素位置

必須設定在 [0...size-1] 間

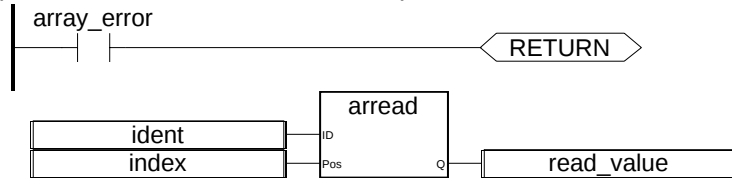
value 整數 由元素所讀取的值

如果參數不被允許，則得 0

描述：

在整數矩陣中，讀取一個元素。

(* FBD 範例：使用矩陣管理方塊 *)



(* ST 相等式：*)

If (array_error) Then Return; End_if;

read_value := ARREAD (ident,index);

(* array_error 是由 ARCREATE 呼叫而得 *)

(* IL 相等式：*)

LD array_error

RETC

LD ident

ARREAD index

ST read_value

ARWRITE



引數：

ID 整數 矩陣的編號 (必須設定在 [0...15] 間)

Pos 整數 矩陣內的元素位置 (必須設定在 [0...size-1] 間)

IN 整數 元素的新值

ok 整數 執行狀態：

1 = 如果已成功寫入

2 = 不允許的矩陣編號

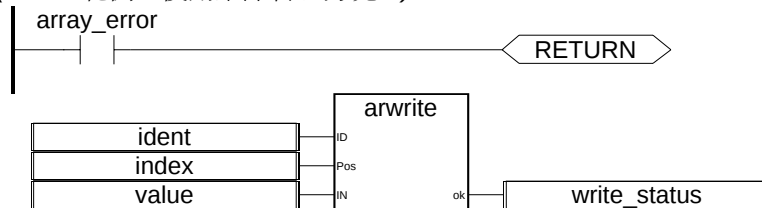
語言參考

3 = 不允許的元素位置

描述：

儲存 (寫入) 數值到整數矩陣中。

(* FBD 範例：使用矩陣管理方塊 *)



(* ST 相等式：*)

If (array_error) Then Return; End_if;

write_status := ARWRITE (ident, index, value);

(* array_error 是由 ARCREATE 呼叫而得 *)

(* IL 相等式：*)

LD array_error

RETC

LD ident

ARWRITE index,value

ST write_status

F_ROPEN



引數：

Path 訊息 檔案名稱

可以使用 \ 或 / 符號來表明的完整路徑。為使應用程式具有可攜性，/ 或 \ 是有同等功用的。

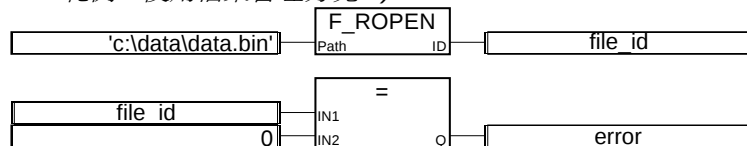
ID 整數 檔案編號

如果錯誤產生則得 0：檔案並不存在。

描述：

以讀取模式開啓二進位檔案。必須和 `FX_READ` 和 `F_CLOSE` 一起使用。此函式並不包含在 `ISaGRAF` 的模擬器中。

(* FBD 範例：使用檔案管理方塊 *)



(* ST 相等式：*)

```
file_id := F_ROPEN('c:\data\data.bin');
```

```
error := (file_id=0);
```

(* IL 相等式：*)

```
LD    'c:\data\data.bin'
```

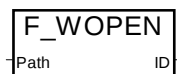
```
F_ROPEN
```

```
ST    file_id
```

```
EQ    0
```

```
ST    error
```

F_WOPEN



引數：

Path 訊息 檔案名稱

可以使用 \ 或 / 符號來表明的完整路徑。爲使應用程式具有可攜性，/ 或 \ 是有同等功用的。

ID 整數 檔案編號

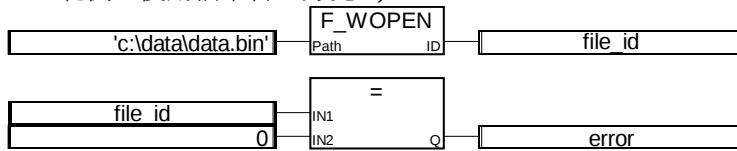
如果錯誤產生則得 0。假如檔案已經存在，將會被覆蓋掉。

描述：

語言參考

以寫入模式開啓二進位檔案。必須和 `FX_WRITE` 和 `F_CLOSE` 一起使用。
此函式並不包含在 `ISaGRAF` 的模擬器中。

(* FBD 範例：使用檔案管理方塊 *)



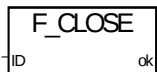
(* ST 相等式 : *)

```
file_id := F_WOPEN('c:\data\data.bin');
error := (file_id=0);
```

(* IL 相等式 : *)

```
LD    'c:\data\data.bin'
F_WOPEN
ST    file_id
EQ    0
ST    error
```

F_CLOSE



引數：

ID 整數 檔案編號：其值經由 `F_ROPEN` 或 `F_WOPEN` 返回所得。

ok 布林 返回之狀態

如果檔案關閉成功，則返回真

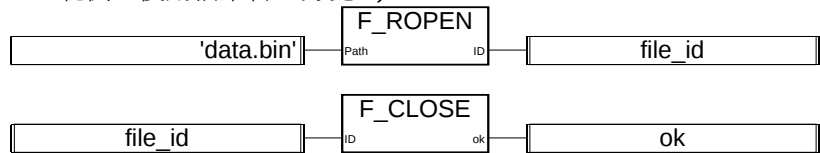
如果錯誤產生，則返回假

描述：

關閉經由 `F_ROPEN` 或 `F_WOPEN` 開啓的二進位檔案。

此函式並不包含在 `ISaGRAF` 的模擬器中。

(* FBD 範例：使用檔案管理方塊 *)



(* ST 相等式 : *)

```
file_id := F_ROPEN('data.bin');
```

```
ok := F_CLOSE(file_id);
```

(* IL 相等式 : *)

```
LD    'data.bin'
```

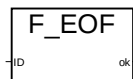
```
F_ROPEN
```

```
ST    file_id
```

```
F_CLOSE      (* file_id 已在目前的 IL 結果中 *)
```

```
ST    ok
```

F_EOF



引數：

ID 整數 檔案編號：其值經由 F_ROPEN 或 F_WOPEN 返回所得。

ok 布林 檔案結束指示器

如果最後的讀或寫程序呼叫遇到檔案結尾時，則返回真。

和 FM_READ 一起使用時，如果最後一個字元並不是字串的結尾，最後讀取的訊息不一定是正確的。

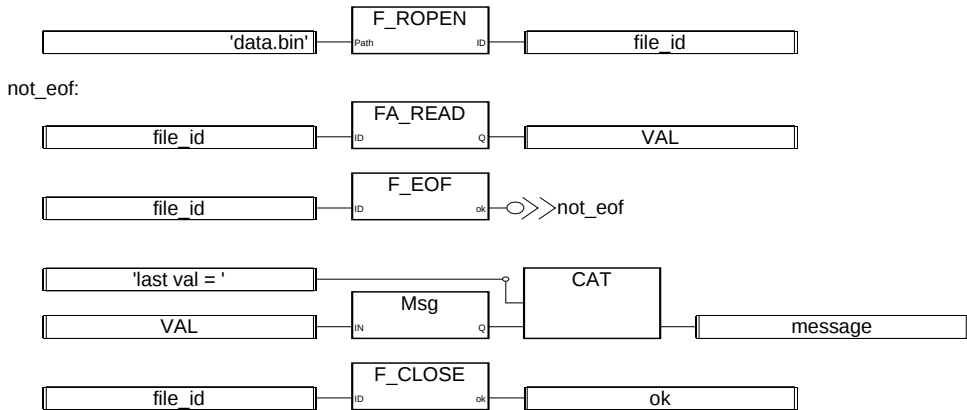
描述：

測試檔案是否已到達結尾。

此函式並不包含在 ISaGRAF 的模擬器中。

(* FBD 範例：使用檔案管理方塊 *)

語言參考



(* ST 相等式 : *)

```

file_id := F_ROPEN('data.bin');
WHILE not(F_EOF(file_id))
  VAL := FA_READ(file_id);
END_WHILE;
MESSAGE := 'last val=' + msg(VAL);
ok := F_CLOSE(file_id);
  
```

(* IL 相等式 : *)

```

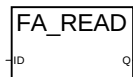
LD  'data.bin'
F_ROPEN
ST  file_id
LD  file_id
F_EOF
JMPC  END_OF_FILE
NOT_EOF: LD  file_id
FA_READ
ST  VAL
LD  file_id
F_EOF
JMPNC  NOT_EOF(* 如果尚未 eof 則繼續讀取 *)
END_OF_FILE: LD  VAL
MSG
  
```

```

ST  val_msg (* 將 VAL 轉換為字串 *)
LD  'last val='
ADD val_msg
ST  MESSAGE
LD  file_id
F_CLOSE
ST  ok

```

FA_READ



引數：

- ID** 整數 檔案編號：其值經由 **F_ROPEN** 返回所得。
- Q** 整數 從檔案讀取回來的整數類比值

描述：

從二進位檔案讀取回來的類比變數。須和 **F_ROPEN** 和 **F_CLOSE** 一起使用。

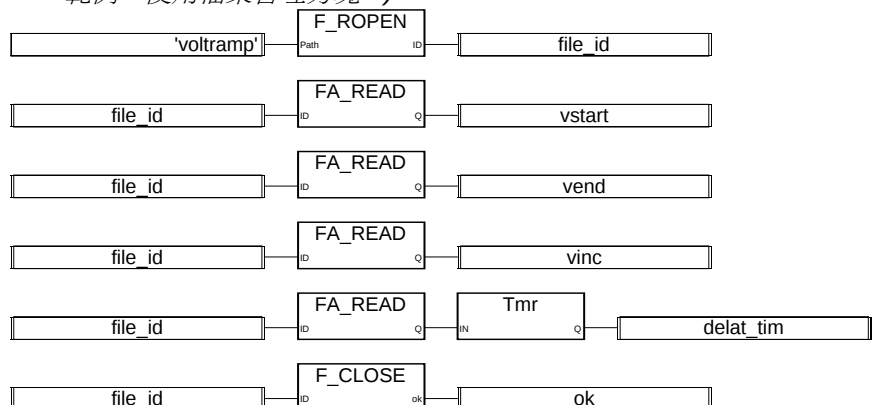
此程序從先前位置開始，對檔案進行循序式的讀取。

在 **F_ROPEN** 動作後的第一次呼叫，將會讀取檔案的前面的四個位元組，每次呼叫則會將讀取指標加四。

想要檢查是否已到達檔案的結尾，則使用 **F_EOF**。

此函式並不包含在 **ISaGRAF** 的模擬器中。

(* FBD 範例：使用檔案管理方塊 *)



語言參考

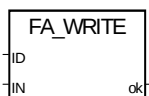
(* ST 相等式 : *)

```
file_id := F_ROPEN('voltramp.bin');  
vstart := FA_READ(file_id);  
vend := FA_READ(file_id);  
vinc := FA_READ(file_id);  
delta_tim := tmr(FA_READ(file_id));  
ok := F_CLOSE(file_id);
```

(* IL 相等式 : *)

```
LD 'voltramp.bin'  
F_ROPEN  
ST file_id  
FA_READ (* 讀取 vstart *)  
ST vstart  
LD file_id  
FA_READ (* 讀取 vend *)  
ST vend  
LD file_id  
FA_READ (* 讀取 vinc *)  
ST vinc  
LD file_id  
FA_READ (* 讀取 delta_tim *)  
TMR (* 轉換成時間 *)  
ST delta_tim  
LD file_id  
F_CLOSE  
ST ok
```

FA_WRITE



引數：

ID 整數 檔案編號：其值經由 **F_WOPEN** 返回所得。

IN 整數 欲寫入檔案中的整數類比值

OK 布林 執行狀態：如果成功則得真

描述：

寫入類比變數到二進位檔案中。

此程序從先前位置開始，對檔案進行循序式的讀取。

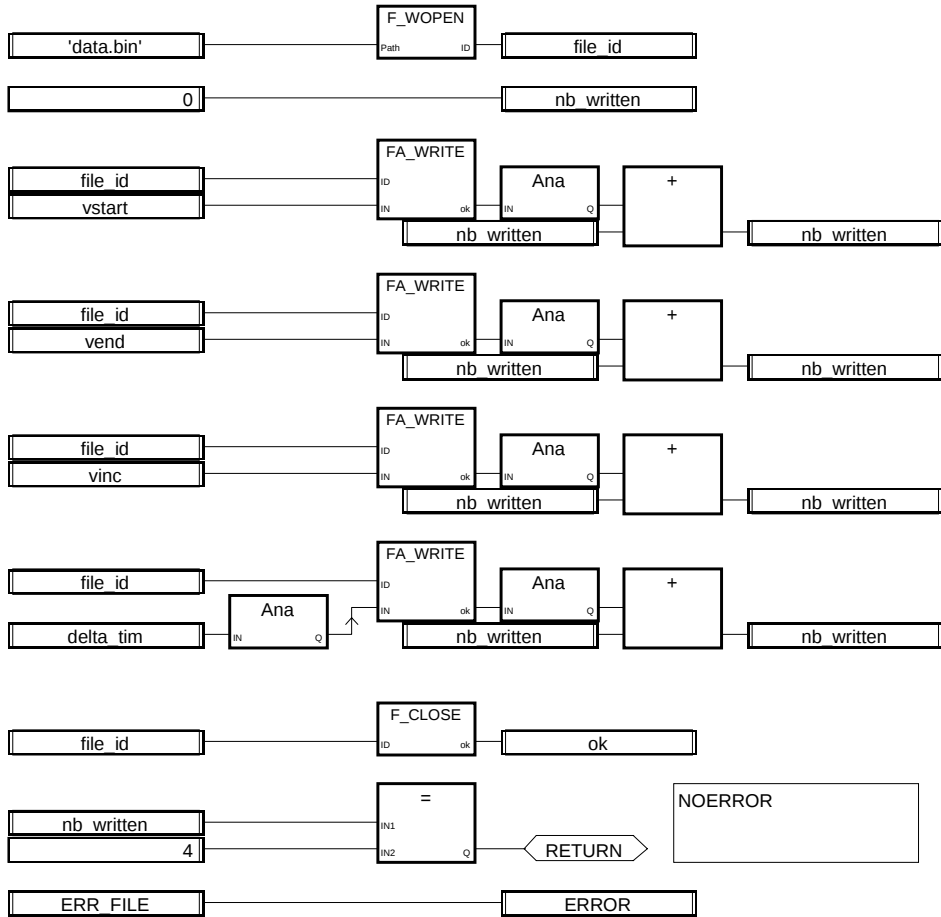
在 **F_WOPEN** 動作後的第一次呼叫，將會寫入檔案的前面的四個位元組，

每次呼叫則會將讀取指標加四。

此函式並不包含在 **ISaGRAF** 的模擬器中。

(* FBD 範例 *)

語言參考



(* ST 相等式 : *)

```
file_id := F_WOPEN('voltramp.bin');
```

```
nb_written := 0;
```

```
nb_written := nb_written + ana(FA_WRITE(file_id,vstart));
```

```
nb_written := nb_written + ana(FA_WRITE(file_id,vend));
```

```
nb_written := nb_written + ana(FA_WRITE(file_id,vinc));
```

```
nb_written := nb_written + ana(FA_WRITE(file_id,ana(delta_tim)));
```

```
ok := F_CLOSE(file_id);
```

```
IF (nb_written <> 4) THEN
```

```
    ERROR := ERR_FILE;
```

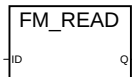
```
END_IF;
```

```
(* IL 相等式 : *)  
LD 'voltramp.bin'  
F_OPEN  
ST file_id  
LD 0  
ST nb_written  
LD file_id (* 寫入 vstart *)  
FA_WRITE vstart  
ANA  
ADD nb_written  
ST nb_written  
LD file_id (* 寫入 vend *)  
FA_WRITE vend  
ANA  
ADD nb_written  
ST nb_written  
LD file_id (* 寫入 vinc *)  
FA_WRITE vinc  
ANA  
ADD nb_written  
LD (* 寫入 delta_tim *)  
ANA (* 轉換成整數 *)  
ST ana_delta_tim  
LD file_id  
FA_WRITE ana_delta_tim  
ANA  
ADD nb_written  
ST nb_written  
F_CLOSE  
ST ok  
LD nb_written  
EQ 4
```

語言參考

```
RETC          (* 若等於4 就跳回 *)  
LD  ERR_FILE  (* 否則錯誤 *)  
ST  ERROR
```

FM_READ



引數：

ID 整數 檔案編號：其值經由 **F_ROPEN** 返回所得。
Q 訊息 從檔案讀取回來的字串

描述：

從二進位檔案讀取回來的訊息變數。

須和 **F_ROPEN** 和 **F_CLOSE** 一起使用。

此程序從先前位置開始，對檔案進行循序式的讀取。

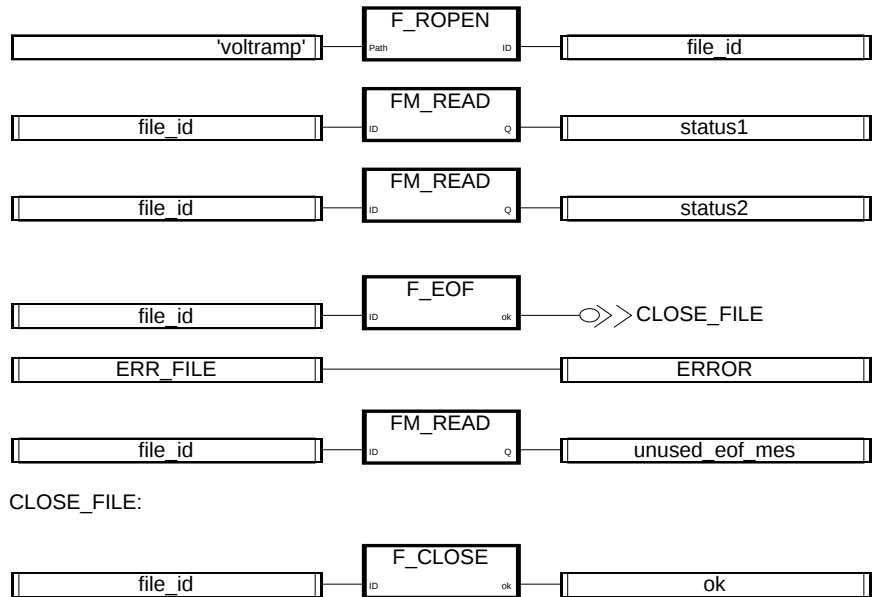
在 **F_ROPEN** 動作後的第一次呼叫，將會讀取檔案的前面的字串，每次呼叫則會將讀取指標加上字串長度。

字串的結束可以是 **null(0)**、列結束 (**'\n'**)，或歸回 (**'\r'**)。

要檢查是否已到達檔案的結尾，則使用 **F_EOF**。

此函數並不包含在 **ISaGRAF** 的模擬器中。

(* FBD 範例：使用檔案管理方塊 *)



(* ST 相等式 : *)

```

file_id := F_ROPEN('voltramp.bin');
status1 := FM_READ(file_id);
status2 := FM_READ(file_id);
IF (F_EOF(file_id)) THEN
  ERROR := ERR_FILE;
  unused_eof_mes := FM_READ(file_id);
END_IF;
ok := F_CLOSE(file_id);
  
```

(* IL 相等式 : *)

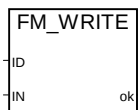
```

LD  'voltramp.bin'
    F_ROPEN
ST  file_id
FM_READ      (* 讀取 status1 *)
ST  status1
LD  file_id
FM_READ      (* 讀取 status2 *)
ST  status2
  
```

語言參考

```
LD file_id
F_EOF
JMPNC CLOSE_FILE (* 若檔案結束則跳過不動作 *)
LD ERR_FILE
ST ERROR
LD file_id
FM_READ (* 讀取unused_eof_mes *)
ST unused_eof_mes
CLOSE_FILE LD file_id
F_CLOSE
ST ok
```

FM_WRITE



引數：

ID 整數 檔案編號：其值經由 F_WOPEN 返回所得。

IN 訊息 欲寫入檔案的字串

ok 布林 執行狀態
成功則得真

描述：

寫入訊息變數到二進位檔案中。

須和 F_WOPEN 和 F_CLOSE 一起使用。

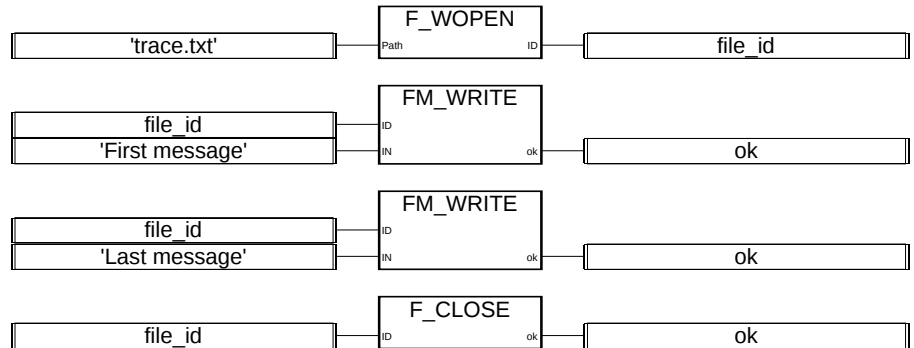
寫入到檔案中的字串以空字元作為結束。

此程序從先前位置開始，對檔案進行循序式的讀取。

在 F_WOPEN 動作後的第一次呼叫，將會寫入字串到檔案中的最前面，每次呼叫則會將寫入指標加上寫入字串的長度。

此函數並不包含在 ISaGRAF 的模擬器中。

(* FBD 範例：使用檔案管理方塊 *)



(* ST 相等式 : *)

```

file_id := F_WOPEN('trace.txt');
ok := FM_WRITE(file_id, 'First message');
ok := FM_WRITE(file_id, 'Last message');
ok := F_CLOSE(file_id);

```

(* IL 相等式 : *)

```

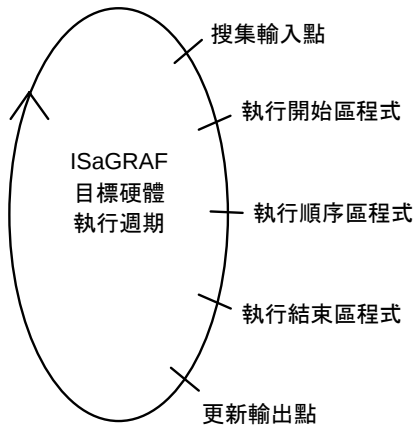
LD  'trace.txt'
F_WOPEN
ST  file_id
FM_WRITE  'First message'  (* 寫入第一個字串 *)
ST  ok
LD  file_id
FM_WRITE  'Last message' (* 寫入第二個字串 *)
ST  ok
LD  file_id
F_CLOSE
ST  ok

```


C. 目標硬體使用者指南

C.1 簡介

ISaGRAF 目標程式是一個在你的工業電腦系統或 IC 板上執行 ISaGRAF 應用程式的即時軟體。它根據以下預定的程序而執行：



ISaGRAF 目標硬體的執行週期包含搜集實際的輸入點，處理 ISaGRAF 工作平台上應用程式所定義的資料，然後更新實際的輸出點。

本章的第一部份解釋如何從個別的作業系統環境入門，依序為 DOS、OS-9、VxWorks 與 NT 等目標硬體。對於每一個系統，首先你會學到如何去執行 ISaGRAF 目標程式，接下來你會找到特殊的資訊關於目標硬體的啟動程序，錯誤處理，一般的表現...等。

第二部分描述使用者定義的 C 函數、功能方塊和轉換函數的施行方法，以便加強 ISaGRAF 目標硬體的功能。

第三部分提供 ISaGRAF 的 Modbus 通訊的資料，包含通訊格式和不同的函數碼。

第四部分描述一些斷電和目標硬體重新啟動所需的軟體工具。

此章節假設使用者已經很熟悉 ISaGRAF 工作平台。

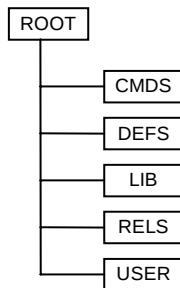
C.2 安裝

安裝大約需要 1 Mbyte 的磁碟空間。在安裝磁片內的 `install.bat` 檔可用來安裝所有在指定作業環境下的必要檔案。

例：`a:\install a: c:\path`

將從 `a:` 碟安裝檔案至 `c:\path`

目錄結構如下：



`ROOT` 目錄內含有一些工具軟體和讀我 (`readme`) 檔案

`CMDS` 目錄內含有執行檔案

`DEFS` 目錄內含有 C 的標頭檔案

`LIB` 目錄內含有函數庫檔案

`RELS` 目錄內含有 C 的物件檔案

`USER` 目錄內含有使用者定義的 C 函數、功能方塊和轉換函數等檔案 (原始碼及標頭檔)

現在你就可以在你想安裝的平台上開始安裝了。

C.3 ISaGRAF DOS 目標硬體入門

C.3.1 執行 ISaGRAF : ISA.EXE

在 DOS 環境下，目標硬體將執行一個單工的程式 ISA.EXE。如果不知如何下達參數，可在命令列下鍵入 `isa -?`，即可得到操作說明。

在此作業環境下，請不要重複使用通訊埠，以免造成不好的執行結果。因為 ISaGRAF 會佔用一個通訊埠，但卻無法防止原先已架設在記憶體內的中斷程式。

☐ 通訊埠連結選項：-t

ISaGRAF 目標程式會佔用一個通訊埠來跟工作平台或監控軟體連接。此通訊埠可以是 COM1、COM2、COM3，完全視 BIOS 的版本而定。使用者可以用 -t 選項來選擇要連接哪個通訊埠。

沒有預設值： 若未使用 -t 下達通訊埠編號，ISaGRAF 目標硬體將無法對外通訊，此時錯誤號碼 7 會被顯示出來。

在 DOS 環境下並不支援乙太網路通訊，詳細資訊可以詢問你的供應商。

通訊埠的參數（諸如傳輸速率、字元長度...等）必須在執行 ISA.EXE 之前先設定好。並請確定工作平台上的通訊參數與目標硬體的通訊參數需一致，方可正常連結。（請參閱使用者指南之“程式的管理”章節）

範例：

MODE COM1:9600,N,8,1

設定通訊埠參數為

傳輸速率 9600

沒有同位檢查

字元長度 8 bits

停止位元長度 1 bit

請注意在某些 BIOS 版本並不支援 19200 以上的傳輸速率。

ICS Triplex ISaGRAF 公司提供 ISAMOD.EXE 讓使用者去設定通訊埠參數：

如 ISAMOD COM1

與 MODE COM1:19200,N,8,1 相同

▣ slave 編號：-s 選項

此選項設定目標硬體的 slave 編號。除了編號 13 (\$0D) 以外，它可以是 1 到 255 中的任何值。此編號用於通訊連結協定中。當連結多個目標硬體時，它的主要用意為用來區別不同目標硬體。請在使用 ISaGRAF 工作平台時，確定 slave 編號與目標硬體設定一致（參考使用者指南：程式管理章節）。

預設值：slave 號碼預設值為 1（與工作平台的預設值相同）

▣ 範例：

isamod COM1 設定通訊埠 COM1 的參數為 19200 鮑率、沒有同位元、8 位元資料長度、1 個停止位元。

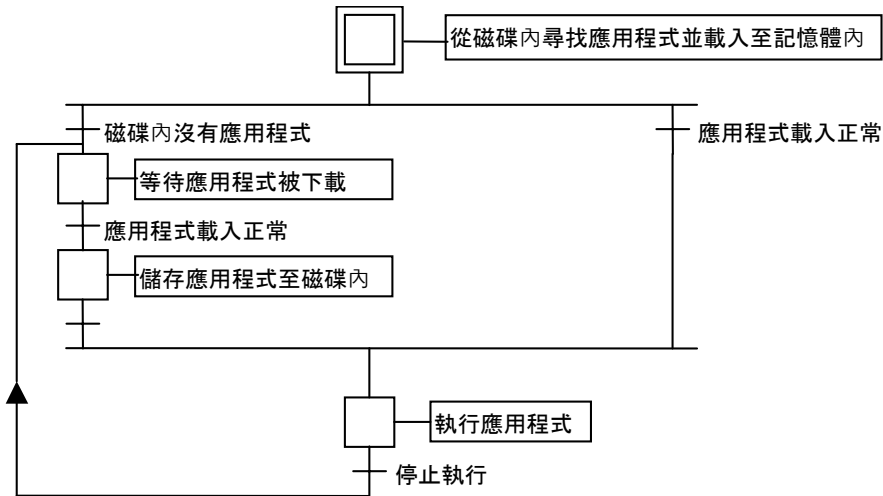
isa -t=COM1 啟動 ISaGRAF 目標硬體為 slave 編號 1，連結埠為 COM1。

isa -s=3 -t=COM1 啟動 ISaGRAF 目標硬體為 slave 編號 3，連結埠為 COM1。

特殊事項

▣ ISaGRAF 的啟動

當目標程式啟動後，它會執行以下的流程



- 定義

應用程式碼是一個由工作平台編譯後產生並下載至目標硬體的二進位資料，它將被目標硬體執行，並可連結到符號表。

應用符號表是一個由工作平台編譯後產生並下載至目標硬體的 ASCII 資料，它可連結符號物件與目標程式的內部物件。除了特殊的符號處理外，它並不是必須的。詳細資料請參見使用者指南：進階程式技巧章節。

- 應用程式的備份

當新的應用程式從工作平台上下載至目標硬體，它會以下列命名方式儲存在目標硬體磁碟目前目錄內：

ISAx1 ISaGRAF 應用程式碼備份檔案 (x 為 slave 編號)

此外如果先前應用符號表曾被下載的話，它會以下述的命名方式儲存在目前目錄內：

ISAx6 ISaGRAF 應用符號表 (x 為 slave 編號)

當 ISaGRAF 目標硬體被啟動後，應用程式碼和符號表會在目前目錄內被搜尋並載入至記憶體內。

如果找不到符號表，目標程式將只執行應用程式。

如果找不到應用程式碼，目標程式將一直等待，直到應用程式碼被下載。

如果不經由除錯器來下載應用程式，想要於 ISaGRAF 目標硬體開機時執行指定的應用程式，可將工作平台磁碟內的下列檔案複製到目標硬體的目前目錄內。假如你的目標硬體沒有磁碟可用，你也可以使用虛擬磁碟機。

比如 ISaGRAF 工作平台的路徑為 \ISAWIN 目錄，則案件 MYPROJ 的應用程式碼將為

```
\ISAWIN\APL\MYPROJ\appli.x8m
```

而應用符號表為

```
\ISAWIN\APL\MYPROJ\appli.tst
```

範例：

先將路徑移至目標硬體的目前目錄

```
copy \ISAWIN\APL\MYPROJ\appli.x8m isa11
```

然後 isa.exe 將可執行 'myproj' 的應用程式。

以上的命令都可建在一個批次檔內，然後可以從工作平台下的工具選單下啟動 (請參閱使用者指南：程式管理章節)。

❏ 錯誤管理和訊息輸出

ISaGRAF 目標硬體軟體整合了錯誤偵測管理。你可以在附表中找到警告錯誤表及其說明。

錯誤偵測程序如下：

錯誤由錯誤及引數號碼所組成，傳送至 ISaGRAF 的錯誤處理函數處理。

假如工作平台製作選項內的錯誤偵測旗標被設定，則錯誤將被處理，否則錯誤訊息將遺失，且錯誤管理程序結束。

當處理時：

錯誤號碼 (十進位制) 與引數 (十六進位制) 顯示在內定的標準輸出。

目標硬體使用者指南

錯誤號碼與引數放進 FIFO (先進先出) 錯誤緩衝區的環狀結構中，是為了方便於稍後回復出來。錯誤緩衝區的大小可由工作平台產生選項來設定，當緩衝區滿了，每次新的錯誤進來，最舊的錯誤將被排擠掉。

錯誤能由除錯器或執行中的應用 (用 **SYSTEM** 函數呼叫，參考使用者指南) 呼叫出來。

當除錯器偵測到錯誤時，錯誤說明訊息將顯示到錯誤視窗中。除錯器視應用的狀態 (執行與否) 顯示錯誤來源的物件名稱 (變數或程式)，或放入方括弧 [x] 中的錯誤引數 (十進位值)，不同的引數號碼有不同的意義。

當目標硬體被啟動或偵測到有錯誤時，一個歡迎或錯誤的編號將被顯示在目標硬體的標準輸出上。如果不希望錯誤被顯示出來可使用下列命令：

如 `isa -t=COM1 -s=1 >NUL`

□ 系統時鐘

因為 ISaGRAF 是被設計成可以在任何作業環境上執行的，所以同步週期及變數的更新時間皆是標準的刻度 (tick)，大約為 55ms。

因此是不可能得到一個比 55ms 還小的時間精度。所以當設定的週期時間小於或等於 55ms 且不等於 0 時，將產生一個錯誤訊息，編號為 62，而且不會去觸發執行週期。

不要改變系統的週期時間將產生一個益處，就是 ISaGRAF 的執行不會干擾應用程式。

如果你要更正確的執行細節，請詢問你的硬體供應商。

□ 離開鍵

如果想在測試時結束 ISaGRAF 的 DOS 目標程式，可以同時按下

shift+ctrl+alt 鍵

當然工業上的設計並不允許按下某些鍵便使程式停止，所以必須提供某些方法使這些鍵失效。

當使用這些結束鍵來停止目標程式，有產生一個危險就是 IO 板並未被關閉，所以正確的結束方式應該為

從工作平台上停止應用程式 (如此將關閉 IO 板)。

使用結束鍵停止 ISaGRAF 目標程式。

⇒ 應用程式碼的大小

由於 ISaGRAF DOS 目標硬體是架構於 Intel 真實模式上，所以應用程式碼最大只能到 64K 位元組，因此由工作平台上下載的大小不可大於此值。在某些非常少見的情況下，ISaGRAF 配置內部使用的結構後也會超過此限制，如此將於應用程式下載後，毀損整個系統。此外 DOS 目標硬體的另一個限制為記憶體容量最大只到 640K byte。

假如你需要額外的記憶體，請詢問你的硬體供應商。

C.4 ISaGRAF OS9 目標硬體入門

首先你必須將某些檔案傳送至你的 OS-9 目標硬體 (至少須傳送 CMD5 目錄上的可執行檔)。

然後你可以在 OS-9 系統執行如下的使用說明：

```
isa -?  
isaker -?  
isatst -?  
isanet -?
```

C.4.1 執行單工的 ISaGRAF 目標程式 : isa

ISaGRAF 的目標硬體可以使用單工的方式來執行。但這是一個比較極端的狀況，請確定不要有其他程式重複使用和 isa 相同的通訊埠，以維持正常的運作。在 OS-9 的多工環境下，在相同的 CPU 內可以跑很多個單工的 ISaGRAF 目標程式，只要他們的 slave 號碼和通訊連結埠皆不同即可。

單工的目標程式主要是被設計成在作業環境功能並非很好時，諸如低成本的 IC 板或 MS-DOS，或是移植至新的作業平台上測試時。因此較被人所接受的方式為多工的作業環境。

此外 ISaGRAF 的單工目標程式無法避免背景程式或中斷所造成的混亂。

☐ 通訊連結及組態：-t 選項

ISaGRAF 單工目標程式使用一個序列通訊埠來與工作平台的除錯器通訊，此通訊埠便是由 -t 選項指定的。

沒有預設值： 若未使用 -t 指定通訊埠，ISaGRAF 目標硬體將無法對外通訊，此時錯誤號碼 7 會被顯示出來。

單工目標程式並不支援乙太網路通訊。

通訊埠是被開啓並設定成二進位資料傳送模式（沒有控制字元，沒有 XON/XOFF）。其它的通訊參數必須在目標程式被執行前先設定好。當使用工作平台的除錯器做連結時，請確定工作平台上的通訊埠參數與目標硬體的通訊參數一致（請參閱使用者指南：程式管理章節）。

範例：

```
xmode /t0 baud=19200
```

將設定通訊埠 /t0 的傳輸速率為 19200

▣ slave 號碼：-s 選項

此選項用來設定目標硬體的 slave 號碼。除了編號 13 (\$0D) 外，它可以是 1 到 255 中的任何值。此編號用於通訊連結協定中。當連結多個目標硬體時，它的主要用意為用來區別不同目標硬體。請在使用 ISaGRAF 工作平台時，確定 slave 編號與目標硬體設定一致（參考使用者指南：程式管理章節）。

預設值： slave 號碼的預設值為 1（與工作平台的 slave 號碼相同）

▣ 例如：

isa -t=/t0 啟動一個單工的 ISaGRAF 目標程式，使用預設的 slave 號碼 (1)，並連結至通訊埠 /t0。

isa -s=3 -t=/t1 啟動一個單工的 ISaGRAF 目標程式，使用 slave 號碼 3，並連結至通訊埠 /t1。

isa -t=/t0 &

isa -s=3 -t=/t1 啟動 2 個 ISaGRAF 單工目標程式，一個使用預設的 slave 號碼 1，並連結至通訊埠 /t0，另一個使用 slave 號碼 3，並連結至通訊埠 /t1。

C.1.1 執行 ISaGRAF 的多工目標程式：isaker， isatst，isanet

爲了提高執行效率，目標硬體可執行多工的目標程式，一部份負責通訊工作 (通訊連結程式 isatst 或 isanet)，另一部分負責應用程式的執行 (核心程式 isaker)。

這是一個比較彈性的架構，它允許使用者將多個通訊程式連結至同一個核心程式，或最多 4 個核心程式連結至同一個通訊程式。這種方式使得某些應用變得很方便，諸如一邊要使用工作平台來查看應用程式的執行狀況，一邊又可使用人機介面同時連結至同一個目標硬體。

核心程式和通訊程式是獨立的，且可以分開啟動的，但是需先執行核心程式，以便先初始化系統環境，然後通訊程式才能和它連結。

ISaGRAF 的多工目標程式無法防止背景程式或中斷所造成的混亂。

C.4.1.1 執行核心程式：isaker

▣ slave 號碼：-s 選項

此選項用來設定目標程式的 slave 號碼。除了編號 13 (\$0D) 外，它可以是 1 到 255 中的任何值。此編號用於通訊連結協定中。當連結多個目標硬體時，它的主要用意爲用來區別不同目標硬體。請在使用 ISaGRAF 工作平台時，確定 slave 編號與目標硬體設定一致 (參考使用者指南：程式管理章節)。

預設值： slave 號碼的預設值爲 1 (與工作平台的 slave 號碼相同)

C.4.1.2 執行序列通訊程式：isatst

▣ 通訊連結及組態：-t 選項

序列通訊程式使用一個序列通訊埠與工作平台相連結，此通訊埠是由 -t 選項來指定的。

沒有預設值： 若未使用 -t 下達通訊埠名稱，ISaGRAF 目標硬體將無法對外通訊，此時錯誤號碼 7 會被顯示出來。

Isatst 無法用來做乙太網路通訊。

通訊埠是被開啓並設定成二進位資料傳送模式（沒有控制字元，沒有 XON/XOFF）。其它的通訊參數必須在目標程式被執行前先設定好。當使用工作平台的除錯器做連結時，請確定工作平台上的通訊埠參數與目標硬體的通訊參數一致（請參閱使用者指南：程式管理章節）。

範例：

```
xmode /t0 baud=19200
```

將設定通訊埠 /t0 的傳輸速率為 19200

▣ slave 號碼選項：-s

此選項設定目標程式的 slave 號碼。除了編號 13 (\$0D) 外，它可以是 1 到 255 中的任何值。此編號是在通訊連結協定中使用。-s 可以重複使用 4 次來連結最多 4 個不同 slave 號碼的核心程式。當連結多個目標硬體時，它的主要用意為用來區別不同目標硬體。請在使用 ISaGRAF 工作平台時，確定 slave 編號與目標硬體設定一致（參考使用者指南：程式管理章節）。

預設值： slave 號碼預設值為 1（與工作平台預設值相同）

▣ 通訊程式邏輯號碼：-c 選項

目標硬體使用者指南

此選項用來設定通訊程式邏輯號碼，它可以用來一次管理多個的通訊程式。它可以是 1 到 255 之間的任何值，但不可重複。

預設值：與最後設定的 `-s` 值一樣，如此可保證與前一版的 ISaGRAF 相容 (3.0 版)。

C.4.1.3 執行乙太網路通訊程式 : isanet

▣ 通訊連結及組態 : `-t` 選項

isanet 使用標準的乙太網路與除錯器連結，其通訊埠號碼 (port number) 由 `-t` 來指定。

沒有預設值： 若未使用 `-t` 下達通訊埠號碼，目標硬體無法對外通訊。此時錯誤編號 7 會被顯示出來。

當使用工作平台的除錯器時，請確定工作平台的通訊參數 (參見使用指南：程式管理章節) 與目標硬體一致。

以 ISaGRAF 而言，OS-9 的目標硬體為伺服器 (server)，而工作平台為客戶端 (client)，並且以指定的埠號碼互相連結。

在使用乙太網路連結前，請確定通訊的驅動模組已被規畫好，你可以使用 ping 到 OS-9 系統的方法來作測試。

▣ slave 號碼 : `-s` 選項

此選項設定目標程式的 slave 號碼。除了編號 13 (\$0D) 外，它可以是 1 到 255 中的任何值。此編號是在通訊連結協定中使用。`-s` 可以重複使用 4 次來連結最多 4 個不同 slave 號碼的核心程式。當連結多個目標硬體時，它的主要用意為用來區別不同目標硬體。請在使用 ISaGRAF 工作平台時，確定 slave 編號與目標硬體設定一致 (參考使用者指南：程式管理章節)。

預設值：slave 號碼預設值為 1 (與工作平台預設值相同)

☞ 通訊程式邏輯號碼：-c 選項

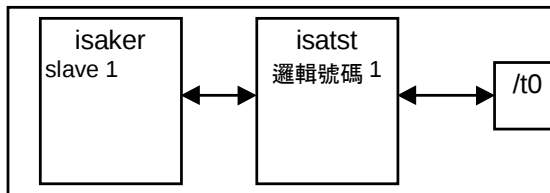
此選項用來設定通訊程式邏輯號碼，它可以用來一次管理多個的通訊程式。它可以是 1 到 255 之間的任何值，但不可重複。

預設值：與最後設定的 -s 值一樣，如此可保證與前一版的 ISaGRAF 相容 (3.0 版)。

C.4.1.4 範例：

isaker &

`isatst -t=/t0`



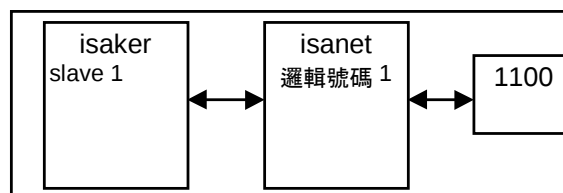
啟動：

一個 ISaGRAF 核心程式，預設的 slave 號碼為 1

一個 ISaGRAF 序列通訊程式 (於 /t0 埠)，並連結到 slave 號碼為 1 的核心程式上，並使用預設的通訊邏輯號碼 (最後指定的 slave 號碼=內定值=1)。

isaker &

`isanet -t=1100`



目標硬體使用者指南

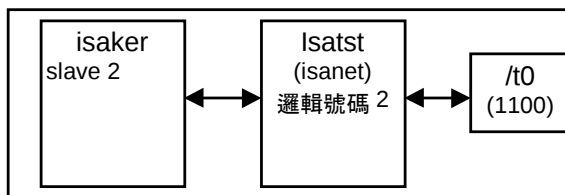
啟動：

一個 ISaGRAF 核心程式，預設的 slave 號碼為 1

一個 ISaGRAF 乙太網路通訊程式 (1100 的埠號碼)，連結到 slave 號碼為 1 的核心程式上，並使用預設的邏輯號碼 (最後指定的 slave 號碼=內定值=1)。

isaker -s=2 &

isatst -t=/t0 -s=2 (或 isanet -t=1100 -s=2)



啟動：

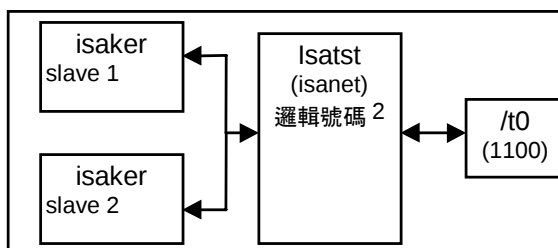
一個 ISaGRAF 核心程式，slave 號碼為 2

一個 ISaGRAF 序列 (乙太網路) 通訊 (在 /t0 (1100) 之通訊埠上)，連結到 slave 號碼為 2 的核心程式上，並使用預設的邏輯號碼 (最後指定的 slave 號碼=2)。

isaker -s=1 &

isaker -s=2 &

isatst -t=/t0 -s=1 -s=2 (或 isanet -t=1100 -s=1 -s=2)



啟動：

一個 ISaGRAF 核心程式，slave 號碼為 1

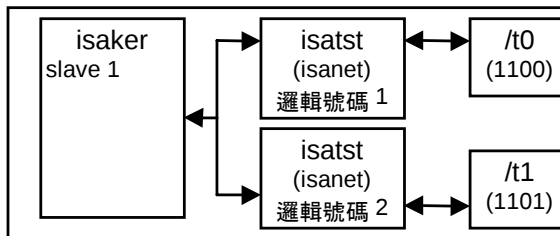
一個 ISaGRAF 核心程式，slave 號碼為 2

一個 ISaGRAF 序列 (乙太網路) 通訊 (在 /t0 (1100) 之通訊埠上)，連結到 slave 號碼為 1 及 2 的核心程式上，並使用預設的邏輯號碼 (最後指定的 slave 號碼=2)。

isaker -s=1 &

isatst -t=/t0 -s=1 -c=1 & (或 isanet -t=1100 -s=1 -c=1 &)

isatst -t=/t1 -s=1 -c=2 (或 isanet -t=1101 -s=1 -c=2)



啓動：

一個 ISaGRAF 核心程式，slave 號碼為 1

一個 ISaGRAF 序列 (乙太網路) 通訊 (在 /t0 (1100) 之通訊埠上)，連結到 slave 號碼為 1 的核心程式上，並使用邏輯號碼 1。

一個 ISaGRAF 序列 (乙太網路) 通訊 (在 /t0 (1100) 之通訊埠上)，連結到 slave 號碼為 1 的核心程式上，並使用邏輯號碼 2。

附註：

序列和乙太網路通訊可混合使用。

C.4.2 特殊事項

☞ 通訊連結

由於 OS-9 的序列字元管理模組 (SCF) 非常的彈性，幾乎任何 Microwave 支援的雙向 (bi-directional) 通訊埠皆可使用：

例如：

序列通訊埠可以是一個架在另一片 CPU 板上的網路通訊埠。

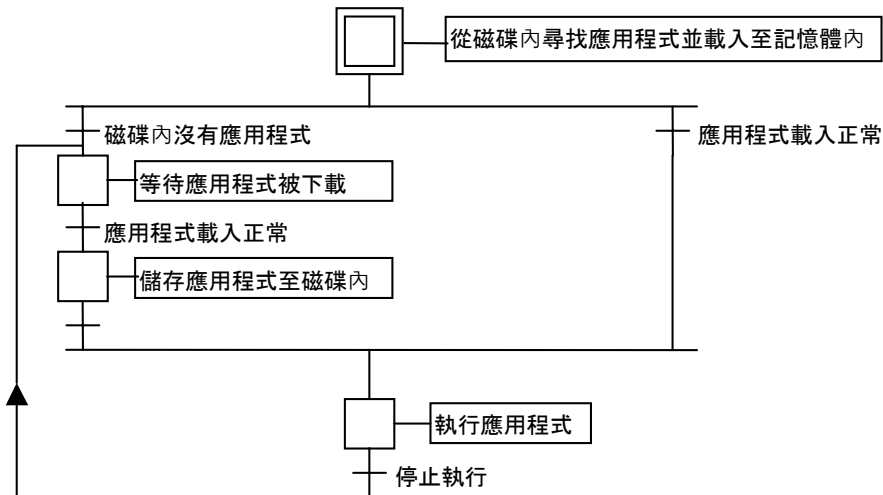
目標硬體使用者指南

之後可用 `-t` 選項來設定：`-t=/nr/MASTER/t0`

MASTER 是一個記憶體網路上的 CPU 板名稱，而 `/t0` 是架在它之上的通訊埠。

ISaGRAF 啟動

當目標硬體啟動後，下列流程將被執行。



• 定義

應用程式是一個由工作平台編譯後產生並下載至目標硬體的二進位碼，它將被目標硬體執行，並可連結到符號表。

應用符號表是一個由工作平台編譯後產生並下載至目標硬體的 ASCII 資料，它可連結符號物件與目標程式的內部物件。除了特殊的符號處理外，它並不是必須的。詳細資料請參考使用者指南：進階程式技巧。

• ISaGRAF OS-9 物件與多工應用

每一個 ISaGRAF 物件都以 'ISAxn' 來命名，其中 **x** 為核心程式的 slave 號碼，**n** 為一個有意義的編號。除了 ISAy3 中之 **y** 為通訊邏輯號碼外，其它都有特殊之意義。

不同的應用程式（核心程式與通訊連結程式）可同時在同一個 CPU 上執行，因為它們可以用 slave 號碼和通訊邏輯號碼來分別。不過當某些應用程

式共同控制相同的 IO 板時，使用者要特別小心，除非是某些 I/O 伺服器，建議不同的應用程式最好別共用同一塊 IO 板。

OS-9 的物件命名：

磁碟檔案：

ISAx1 ISaGRAF 應用程式碼備份檔

ISAx6 ISaGRAF 應用符號備份檔

記憶模組：

ISAx0 ISaGRAF 核心系統資料

ISAx1 ISaGRAF 應用程式碼

ISAx2 ISaGRAF 核心即時資料庫

ISAy3 ISaGRAF 通訊資料交換緩衝區

ISAx4 ISaGRAF 應用程式碼線上修訂 1

ISAx5 ISaGRAF 應用程式碼線上修訂 2

ISAx6 ISaGRAF 應用符號

所以使用者必須小心不要使用相同名字的物件。

- 應用程式備份

當應用程式從工作平台上下載至目標硬體時，它會以下列命名方式儲存在目標硬體磁碟的目前目錄內。

ISAx1 ISaGRAF 應用程式碼檔案 (x 為 slave 號碼)

此外如果先前應用符號表曾被下載的話，它會以下述的命名方式儲存在目前目錄內：

ISAx6 ISaGRAF 應用符號表 (x 為 slave 號碼)

當 ISaGRAF 目標硬體開始時，會搜尋現在目錄下的應用程式碼及應用符號檔，再將它們載入記憶體中。

如果找不到符號表，目標程式將只執行應用程式。

假如應用程式碼不存在於記憶體中，目標硬體會等待應用程式碼被下載。

目標硬體使用者指南

爲了於開機後執行指定的應用，而不使用除錯器連結的方法，可以：

第一個方法爲從工作平台所安裝的 PC 端直接拷貝這些檔案到目標硬體目前磁碟去。你可以使用任何的檔案傳送工具來拷貝，也可以使用工作平台的“工具”功能表 (參考使用者指南的程式管理章節)，使這些操作更容易。

第二個方法爲使用自己的工具，從工作平台安裝的 PC 端將應用程式碼檔案 (如果需要的話，再加上應用符號表檔案) 儲存在非消失的記憶體中 (如 PROM 或 EPROM)。

然後在系統開機後，假如需要的話 (如更快速的讀寫或中斷點管理的原因)，你可以用自己的工具將應用程式碼 (需要的話再加上應用符號表) 從 PROM 載入到 RAM 內。譬如 ISAx1 (如果必要的話連 ISAx6 一起) 由 PROM 載入至 RAM 內。

警告：

假如應用程式碼是不可寫入的，ISaGRAF 除錯器的中斷點管理無法正確執行，但是你的應用程式之前已完全測試過的話，這並不是個問題。

假如 ISaGRAF 工作平台安裝在標準的 \ISAWIN 目錄下：

案件 MYPROJ 的應用程式碼檔案是：

\ISAWIN\APL\MYPROJ\appli.x6m (相當於目標硬體端的 isax1)

案件 MYPROJ 的應用符號檔案是：

\ISAWIN\APL\MYPROJ\appli.tst (相當於目標硬體端的 isax6)

❏ 錯誤管理和訊息輸出

ISaGRAF 目標硬體軟體整合了錯誤偵測管理。你可以在附表中找到警告錯誤表及其說明。

錯誤偵測程序如下：

錯誤由錯誤及引數號碼所組成，傳送至 ISaGRAF 的錯誤處理函數處理。

假如工作平台製作選項內的錯誤偵測旗標被設定，則錯誤將被處理，否則錯誤訊息將遺失，且錯誤管理程序結束。

當處理時：

錯誤號碼 (十進位制) 與引數 (十六進位制) 顯示在內定的標準輸出。

錯誤號碼與引數放進 FIFO (先進先出) 錯誤緩衝區的環狀結構中，是爲了方便於稍後回復出來。錯誤緩衝區的大小可由工作平台產生選項來設定，當緩衝區滿了，每次新的錯誤進來，最舊的錯誤將被排擠掉。

錯誤能由除錯器或執行中的應用 (用 SYSTEM 函數呼叫，參考使用者指南) 呼叫出來。

當除錯器偵測到錯誤時，錯誤說明訊息將顯示到錯誤視窗中。除錯器視應用的狀態 (執行與否) 顯示錯誤來源的物件名稱 (變數或程式)，或放入方括弧中[x]的錯誤引數 (十進位值)，不同的引數號碼有不同的意義。

當目標硬體被啟動或偵測到有錯誤時，一個歡迎或錯誤的編號將被顯示在目標硬體的標準輸出上。如果不希望錯誤被顯示出來可使用下列命令：

```
prog_name [options] >>>/nil
```

□ 週期間隔、程式行爲與程式優先權

ISaGRAF 在週期的結尾 (開始新週期之前) 實行下面的規則：

假如週期時間被指定 (從工作平台指定，參考使用者指南的程式管理章節)，CPU 會將剩餘的時間區段 (指定的週期時間減應用程式所用的時間) 作廢。假如剩餘的時間爲負值，將會產生時間溢位，且 CPU 會釋放 1 個 tick 時間，來強制這樣的排程。

假如沒有指定週期時間，或剩餘時間小於一個 tick 或等於零，CPU 會釋放 1 個 tick 時間，來強制這樣的排程。

目標硬體的時間正確率相當於一個 OS-9 的系統 tick。

指定週期時間通常用在觸發固定的週期時間或讓出 CPU 給其它執行於 OS-9 系統的程序。

目標硬體使用者指南

當沒有資料透過通訊連結進來時，通訊程式是位於睡眠狀態中。當需要時，這個程式透過與核心程式問答格式的方式來得到執行應用的資訊。通訊程式問核心問題，於週期的結束時，核心會給予通訊程式回答。

ISaGRAF task 不會更改給定的優先權，使用者可以根據上述的 ISaGRAF task 行為與全部的應用程式需求自由調整這些優先權。譬如，想要確定你的 OS-9 系統是規劃成一個不被較低優先權程式 preempted 的系統，你可以更改 OS-9 的工作管理參數 (如 **MIN_AGE** 與 **MAX_AGE**)。

▣ 終端模式

OS-9 通訊連結程式若接受連續的 3 個換行字元 (\$0D)，將會啟動一個 OS-9 的命令列程式 (shell) (假如這個程式可使用的話)。

所以在 ISaGRAF 的目標硬體上是可以使用 OS-9 命令列方式來操作的。

例如：

從 PC 端：

關閉工作平台除錯器。

啟動 Windows 終端機 (位於附屬應用程式群組內)，並注意是否與目標硬體的通訊參數設定一致。

連續按 3 個 return 鍵。

於是你將進入 OS-9 的命令列內

若要離開終端模式，鍵入 **logout** 離開它。

警告：

終端模式必須以正確的方式來離開 (使用 **logout** 命令，而不可再鍵入其它字元)。否則將會導致工作平台無法和目標硬體相連接。

C.5 ISaGRAF VxWorks 目標硬體入門

在啟動 ISaGRAF VxWorks 目標硬體之前，必須先執行某些命令以便將整個作業環境規劃好。這些命令可以被放在一個草稿檔 (script) 內，我們將於下一個章節再討論它。

C.5.1 系統資源管理 : isassr.o

在規劃任何 ISaGRAF 目標硬體時，這個模組是必須的，且必須是第一個被載入的 ISaGRAF 目標硬體模組，它將使多工目標程式系統資源管理致能。

C.5.2 isa.o、isakerse.o 與 isakeret.o 的一般事項

要執行 ISaGRAF，下列模組的其中之一必須載入。

isa.o : 使 ISaGRAF 單工目標程式致能。(只能使用序列通訊)

isakerse.o : 使 ISaGRAF 多工目標程式致能。(只能使用序列通訊)

isakeret.o : 使 ISaGRAF 多工目標程式致能。(序列 或且 乙太網路通訊皆可)

這些模組將於下一章中詳細說明。

⇒ 序列通訊連結組態

基本上，ISaGRAF 目標硬體使用一個序列通訊與除錯器連結，但 ISaGRAF 目標程式在開啓它時，並不會重新設定它的通訊參數，因此使用者可完全調整通訊埠的參數。雖然如此仍然需要二位元 (binary) 的資料傳輸模式 (RAW 模式)，所以便提供了 ISAMOD () 函數。

```
uchar ISAMOD
(
char *desc, /* 序列通訊埠名稱 */
uint32 baudrate /* 傳輸速率 */
)
```

)

說明：

將指定的序列通訊埠規劃成二位元傳輸模式，並設定成給定的傳輸速率。

傳回值：

成功：0，錯誤：BAD_RET

當使用工作平台除錯器時，請確定兩邊的通訊參數皆相同（請參考使用者指南：程式管理）。

□ 系統時率

全域變數 CLKRATE (uint32) 必須被給定初始值，以便初始化 VxWorks 的系統時率。你可以使用

```
CLKRATE = sysClkRateGet ()
```

CLKRATE 的預設值為 60Hz。

C.5.3 執行 ISaGRAF 單程式：isa.o

ISaGRAF 目標程式可以是單工的，但這種規劃通常有限制，所以為了維持良好的執行效率，請不要重複使用通訊埠。在 VxWorks 的多工系統下，同一個 CPU 上可同時執行多個單工的 ISaGRAF 目標程式，只要它們 slave 號碼與通訊連結埠不同即可。

單工目標程式主要是設計給系統資源較差的作業系統，如 MS-DOS 作業系統的 PC 或用來製作測試版，以用來整合至一個新的系統時。所以多工系統是比較受到歡迎的。

此外 ISaGRAF 單工目標程式無法防避背景程式或中斷所產生的干擾。

□ slave 設定

ISaGRAF 目標程式是以 slave 號碼來區分的。它的值除了 13 (\$0D) 外可以是 1 到 255 間的任何值，此值主要用於通訊協定內。當多個目標程式同時執

行時，必須加以區別。因此在啟動 ISaGRAF 目標程式前，必須先設定好，所以便提供了 `isa_register_slave()` 函數。

```
uchar isa_register_slave
(
    uchar slave/* slave 號碼 */
)
```

說明：

新增一個 slave 號碼到多工目標硬體管理系統。

傳回值：

成功：0，失敗：BAD_RET

⇒ 應用程式備份檔儲存單元

全域變數 `TSK_FUNIT (char *)` 可被初始成應用程式備份檔的路徑名稱。ISaGRAF 的目標程式簡單地使用標準的檔案管理函數 `fopen`、`fread`、`fwrite`、`fclose` 來進行應用程式備份。預設值是空字串 (“”), 表示沒有儲存單元。

例如：

`TSK_FUNIT = "host name:/C:/ISaGRAF/目標硬體/apl/"`

設定應用程式碼檔案路徑為 `host_name` PC 的 C 碟 `ISaGRAF\目標硬體\apl\`。小心不要忘了最後的斜線，不然路徑會變成 `ISaGRAF\目標硬體`，而把 `apl` 當成檔名。

假如有需要的話，這個變數可以於目標程式啟動前，設定成不同的路徑給不同的目標程式使用。稍後的章節將再詳細說明。

⇒ 週期控制

`TSK_NBTCKSCHED (uint 32)` 變數可用來設定單位為 tick 的週期延遲，此值在 ISaGRAF 目標程式每個週期結束時被使用。

預設值為 0。

目標硬體使用者指南

假如有需要，此值可設定成很多個給不同的目標程式使用。

特殊事項可於週期時間、程式行為與程式優先權章節內找到更多的說明。

▣ ISaGRAF 目標程式啟動

只要環境規劃做好後，最後便可啟動 ISaGRAF 目標程式：`isa_main`。

```
uchar isa_main  
(  
    uchar slave,    /* slave 號碼 */  
    char *com /* 序列通訊埠名稱 */  
)
```

說明：

啟動一個 ISaGRAF 目標程式。

傳回值：

成功：0，失敗：一個非 0 的值

`slave` 號碼與 `slave` 設定章節的說明一樣。

如果 `slave` 號碼和通訊埠名稱皆不同的話，便可同時執行數個目標程式。

當使用工作平台除錯器時，請確定兩邊使用相同的 `slave` 號碼。(參考使用者指南：程式管理)

▣ 範例

下述的例子說明如何去啟動一個 `slave` 號碼為 1 的 ISaGRAF 單工目標程式，並架設在 `/tyCo/1` 的序列通訊埠上。

目前的目錄為目標程式被安裝的所在路徑。

載入 `isassr.o` 模組

```
ld < RELS/isassr.o
```

載入 `isa.o` 模組

`ld < CMDS/isa.o`

規劃序列通訊

`ISAMOD ("tyCo/1",19200)`

系統時率

`CLKRATE = sysClkRateGet ()`

設定 slave 號碼

`isa_register_slave (1)`

檔案儲存單元 (為預設值，可跳過)

`TSK_FUNIT = ""`

週期控制 (為預設值，可跳過)

`TSK_NBTCKSCHED = 0`

啟動 ISaGRAF 目標程式

`sp (isa_main, 1, "tyCo/1")`

C.5.4 執行 ISaGRAF 多工目標程式：isakerse.o 與

isakeret.o

爲了提高 ISaGRAF 目標程式的執行效率，所以將通訊的部分從核心程式分離出來。如此架構較具有彈性，它允許使用者可只執行一個核心程式，但卻可以連結最多 4 個通訊程式，或者執行 4 個核心程式，而同時連結到同一個通訊程式，這使某些應用變得更方便。例如可同時使用人機介面通訊又可使用工作平台除錯器來除錯。

核心程式和通訊程式可分別被啟動，唯一的要求是必須先啟動核心程式，它會進行系統環境初始工作，然後通訊程式方可連結到它。

目標硬體使用者指南

ISaGRAF 多工目標程式無法避免背景程式或中斷所造成的干擾。

有兩個目標程式可依據通訊硬體架構來選用：

核心和序列通訊程式：`isakerse.o`

此模組啟動核心程式和序列通訊程式。

核心和序列 或與 乙太網路通訊程式：`isakeret.o`

此模組啟動核心程式和序列 或與 乙太網路通訊程式。

啟動 `isakerse.o` 和 `isakeret.o` 兩個模組的方式是相同的。當啟動 ISaGRAF 通訊程式：`tst_main_ex` 時 (下面的章節會說明)，除了 `isakeret.o` 外，你可以設通訊埠名稱或乙太網路埠號碼。

以 ISaGRAF 而言，VxWorks 目標硬體為通訊伺服器，而除錯器則為客戶端，並以設定的通訊埠號碼連結到目標硬體。

▣ 核心程式的設定

此選項設定目標硬體的 `slave` 編號。除了編號 13 (`$0D`) 以外，它可以是 1 到 255 中的任何值。此編號用於通訊連結協定中。當連結多個目標硬體時，它的主要用意為用來區別不同目標硬體。因此在核心程式啟動之前，你必須先設定它。可以使用 `isa_register_slave()` 函數。

```
uchar isa_register_slave  
(  
    uchar slave /* slave 號碼 */  
)
```

說明：

增加一個新的 `slave` 號碼到多工管理系統中。

傳回值：

成功：0，失敗：`BAD_RET`

☞ 通訊程式設定

ISaGRAF 通訊程式是以它的通訊邏輯編號來區別的，它是用來管理當多個通訊程式同時啟動時。它可以是 1 到 255 間的任何值，但不可重複。因此在啟動通訊程式時，你必須先設定它們，可以使用 `isa_register_com()` 函數。

```
uchar isa_register_com
(
    uchar com_id      /* 通訊程式辨別碼 */
)
```

說明：

增加一個通訊程式的設定至多工程式管理系統

傳回值：

成功：0，失敗：BAD_RET

☞ 應用程式備份檔儲存單元

全域變數 `TSK_FUNIT (char *)` 可被初始成應用程式備份檔的路徑名稱。ISaGRAF 的目標程式使用標準的檔案管理函數 `fopen`、`fread`、`fwrite`、`fclose` 來進行應用程式備份。預設值為空字串 (“”), 表示沒有儲存單元。

例如：

`TSK_FUNIT = "host name:/C:/ISaGRAF/目標硬體/apl/"`

設定應用程式碼檔案路徑為 `host_name` PC 的 C 碟 `ISaGRAF\目標硬體\apl\`。小心不要忘了最後的斜線，不然路徑會變成 `ISaGRAF\目標硬體`，而把 `apl` 當成檔名。

假如有需要的話，這個變數可以於目標程式啟動前，設定成不同的路徑給不同的目標程式使用。稍後的章節將再詳細說明。

☞ 週期控制

目標硬體使用者指南

TSK_NBTCKSCHED (uint 32) 變數可用來設定單位為 tick 的週期延遲，此值在 ISaGRAF 目標程式每個週期結束時被使用。預設值為 0。假如有需要，此值可設定成很多個給不同的目標程式使用。

▣ ISaGRAF 目標程式啟動

只要環境規劃做好後，最後便可啟動 ISaGRAF 目標程式：isa_main。

```
uchar isa_main
(
    uchar slave,    /* Slave 號碼 */
    char *com /* 序列通訊埠名稱 */
)
```

說明：

啟動一個 ISaGRAF 目標程式。

傳回值：

成功：0，失敗：一個非 0 的值

slave 號碼與 slave 設定章節的說明一樣。

如果 slave 號碼和通訊埠名稱皆不同的話，便可同時執行數個目標程式。

▣ ISaGRAF 通訊程式啟動

一但系統環境規劃好了，最後一個步驟便是啟動 ISaGRAF 通訊程式：tst_main_ex。

```
uchar tst_main_ex
(
    char *com, /* 通訊埠名稱 */
    uchar *slave, /* 4 個位元組的記憶空間,儲存與它連結的核心程式的 slave 號碼 */
    uchar com_id /* 通訊連結辨別碼 */
)
```

)

說明：

啓動一個 ISaGRAF 通訊程式

傳回值：

假如有錯誤發生的話，會傳回一個非零之值。

上述 4 個位元組的記憶空間設定通訊程式所連結的核心程式 slave 號碼。假如少於 4 個核心程式被連結，其它的位元組必須填零。一但程式啓動後，這些記憶空間便沒有必要了。

通訊埠名稱爲序列通訊欲使用的通訊埠。

如果通訊辨別碼皆不同的話，可同時啓動多個通訊程式。

當使用工作平台除錯器時，請確定兩邊使用相同的通訊參數（參考使用者指南：程式管理）。

□ 範例：

此例說明如何啓動：

slave 號碼爲 1 的 ISaGRAF 核心程式。

辨別碼爲 1 的 ISaGRAF 序列通訊程式，並使用 /tyCo/1 通訊埠連結到 slave 號碼爲 1 的核心程式。

辨別碼爲 2 的 ISaGRAF 乙太網路通訊程式，並使用編號 1100 的通訊埠連結到 slave 號碼爲 1 的核心程式。

目前路徑爲 ISaGRAF 目標程式所安裝的路徑。

載入 isassr.o 模組

ld < RELS/isassr.o

載入 isakeret.o 模組（若不使用乙太網路，你可以使用 isakerse.o 模組）

ld < CMDS/isakeret.o

序列通訊的組態

目標硬體使用者指南

ISAMOD (“/tyCo/1”,19200)

系統時率

CLKRATE = sysClkRateGet ()

slave 號碼的設定

isa_register_slave (1)

通訊的設定

isa_register_com (1)

isa_register_com (2)

檔案儲存單元 (如果欲使用預設值，可跳過)

TSK_FUNIT = “”

週期控制 (如果欲使用預設值，可跳過)

TSK_NBTCKSCHED = 0

ISaGRAF 核心程式的啟動

sp (isa_main, 1, “”)

通訊程式連結之 slave 號碼

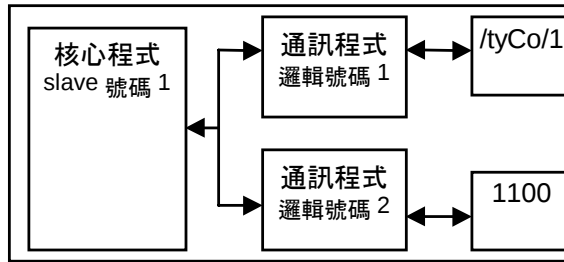
SlavesLink = 0x01000000

ISaGRAF 通訊程式的啟動

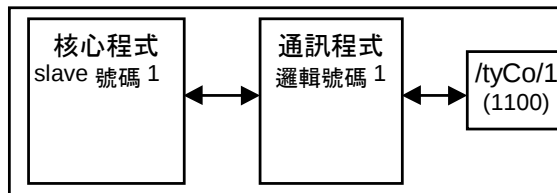
sp (tst_main_ex, “/tyCo/1”, &SlavesLink, 1)

sp (tst_main_ex, “1100”, &SlavesLink, 2)

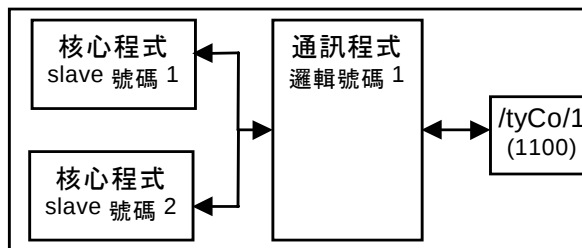
以上的程序將以下圖表示



你也可以規劃成下列的架構



以上為最基本的規劃，啟動一個核心與序列通訊程式（或乙太網路通訊程式）

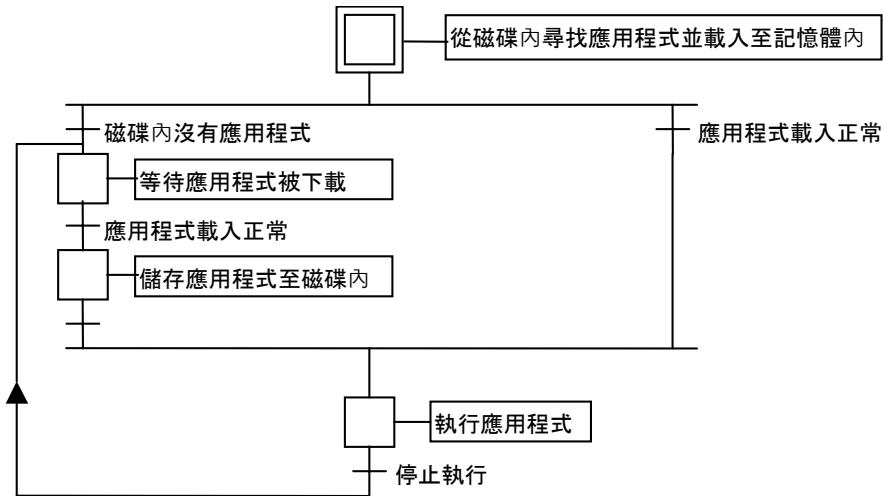


以上為另一個規劃，包含 2 個核心程式並連結到同一個序列通訊程式（或乙太網路通訊程式）。此例 SlavesLink = 0x01020000。

C.5.5 特殊事項

⇒ ISaGRAF 啟動

當目標程式啟動後，下列流程將被執行。



- 定義

應用程式碼是一種二元資料，由工作平台產生及下載，然後由目標硬體執行。應用程式碼加上符號表會更完整。

應用符號表是一種 ASCII 資料，由工作平台產生及下載。平常目標硬體並不需要它，除了使用者要用到符號管理時才用的到。關於符號表的詳細說明可參考使用者手冊的“進階程式技術”章節。

磁碟檔單元的路徑由 ISaGRAF 目標硬體開機檔的全域變數 TSK_FUNIT 所指定 (內定值 = “”，表示不指定磁碟檔單元)。

- ISaGRAF 多工應用

不同的應用程式 (核心程式及通訊程式) 可以於同一個 CPU 上同時執行，只要它們有不同的 slave 編號與不同的通訊邏輯號碼，但是當不同的應用共享相同的物件 (如 I/O 板) 時，使用者就得特別注意了，除非有提供如 I/O 伺服器或 semaphore 機構的情況下，才可以共用相同的物件，否則不同的應用程式還是得使用不同的板子。

- 應用程式備份

當工作平台除錯器下載新的應用程式到目標硬體後，會將應用程式碼儲存 (目標硬體使用標準檔案管理函數 fopen, ...) 成如下的檔名：

路徑 ISAx1 ISaGRAF 應用程式碼備份檔 (x 是 slave 號碼)

假如應用符號表先前已下載過，亦會在目標硬體目前目錄下儲存成如下的名稱：

路徑 **ISAx6** ISaGRAF 應用符號備份檔 (x 是 slave 號碼)

路徑是由 ISaGRAF 目標硬體開機檔內的全域變數 **TSK_FUNIT** 所指定，它的內定值為空字串 “”，表示無磁碟檔單元。

當 ISaGRAF 目標硬體開始時，會搜尋現在目錄下的應用程式碼及應用符號檔，再將它們載入記憶體中。

如果找不到符號表，目標程式將只執行應用程式。

假如應用程式碼不存在於記憶體中，目標硬體會等待應用程式碼被下載。

爲了於開機後執行指定的應用，而不使用除錯器連結的方法，可以：

第一個方法爲從工作平台所安裝的 PC 端直接拷貝這些檔案到應用程式備份儲存單元去，你可以使用任何的檔案傳送工具來拷貝，也可以使用工作平台的“工具”功能表（參考使用者指南的程式管理章節），使這些操作更容易。

第二個方法爲使用自己的工具，從工作平台安裝的 PC 端將應用程式碼檔案（如果需要的話，再加上應用符號表檔案）儲存在非消失的記憶體中（如 PROM 或 EPROM）。

然後在系統開機後，假如需要的話（如更快速的讀寫或中斷點管理的原因），你可以用自己的工具將應用程式碼（需要的話再加上應用符號表）從 PROM 載入到 RAM 內。

然後在 ISaGRAF 開始時，你必須指定應用程式碼（需要的話再加上應用符號表）於記憶中的位址，可用全域變數 **SSR** 來指定，如下：

SSR[x][1].space = 應用程式碼所在的位址

如果需要的話，再加上：

SSR[x][6].space = 應用符號表所在的位址

目標硬體使用者指南

你可以寫一個簡短的程序來做到這樣事。全域變數 `SSR` 宣告成 `str_ssr` 結構型態 (定義於 `tasy0ssr.h` 檔中)。

警告：

假如應用程式碼是不可寫入的，ISaGRAF 除錯器的中斷點管理無法正確執行，但是你的應用程式之前已完全測試過的話，這並不是個問題。

假如 ISaGRAF 工作平台安裝在標準的 `\ISAWIN` 目錄下：

案件 `MYPROJ` 的應用程式碼檔案是：

`\ISAWIN\APL\MYPROJ\appli.x6m` (相當於目標硬體端的 `isax1`)

案件 `MYPROJ` 的應用符號檔案是：

`\ISAWIN\APL\MYPROJ\appli.tst` (相當於目標硬體端的 `isax6`)

❏ 錯誤管理與訊息輸出

ISaGRAF 目標硬體軟體整合了錯誤偵測管理。你可以在附表中找到警告錯誤表及其說明。

錯誤偵測程序如下：

錯誤由錯誤及引數號碼所組成，傳送至 ISaGRAF 的錯誤處理函數處理。

假如工作平台製作選項內的錯誤偵測旗標被設定，則錯誤將被處理，否則錯誤訊息將遺失，且錯誤管理程序結束。

當處理時：

錯誤號碼 (十進位制) 與引數 (十六進位制) 顯示在內定的標準輸出。

錯誤號碼與引數放進 FIFO (先進先出) 錯誤緩衝區的環狀結構中，是為了方便於稍後回復出來。錯誤緩衝區的大小可由工作平台產生選項來設定，當緩衝區滿了，每次新的錯誤進來，最舊的錯誤將被排擠掉。

錯誤能由除錯器或執行中的應用 (用 `SYSTEM` 函數呼叫，參考使用者指南) 呼叫出來。

當除錯器偵測到錯誤時，錯誤說明訊息將顯示到錯誤視窗中。除錯器視應用的狀態（執行與否）顯示錯誤來源的物件名稱（變數或程式），或放入方括弧中的錯誤引數（十進位值），不同的引數號碼有不同的意義。

於目標硬體端，當錯誤偵測到時，錯誤值顯示在內定的標準輸出上，也就是錯誤使用 VxWorks 的函數來直接顯示，如：

```
ioGlobalStdSet()
```

或 `ioTaskStdSet()`

⇒ 週期間隔、程式行為與程式優先權

ISaGRAF 在週期的結尾（開始新週期之前）實行下面的規則：

假如週期時間被指定（從工作平台指定，參考使用者指南的程式管理章節），CPU 會將剩餘的時間區段（指定的週期時間減應用程式所用的時間）作廢。假如剩餘的時間為負值，將會產生時間溢位，且 CPU 會釋放 TSK_NBTCKSCHED（由 ISaGRAF 開機檔所設定的變數）所指定的 tick 時間，來強制這樣的排程。

假如沒有指定週期時間，或剩餘時間小於一個 tick 或等於零，CPU 會釋放 TSK_NBTCKSCHED 所指定的 tick 時間，來強制這樣的排程。

目標硬體的時間正確率相當於一個 VxWorks 的系統 tick。

指定週期時間通常用在觸發固定的週期時間或讓出 CPU 給其它執行於 VxWorks 系統的程序。

當沒有資料透過通訊連結進來時，通訊程式是位於睡眠狀態中。當需要時，這個程式透過與核心程式問答格式的方式來得到執行應用的資訊。通訊程式問核心問題，於週期的結束時，核心會給予通訊程式回答。

ISaGRAF task 不會更改給定的優先權，使用者可以根據上述的 ISaGRAF task 行為與全部的應用程式需求自由調整這些優先權。

C.6 ISaGRAF NT 目標硬體入門

C.6.1 執行ISaGRAF

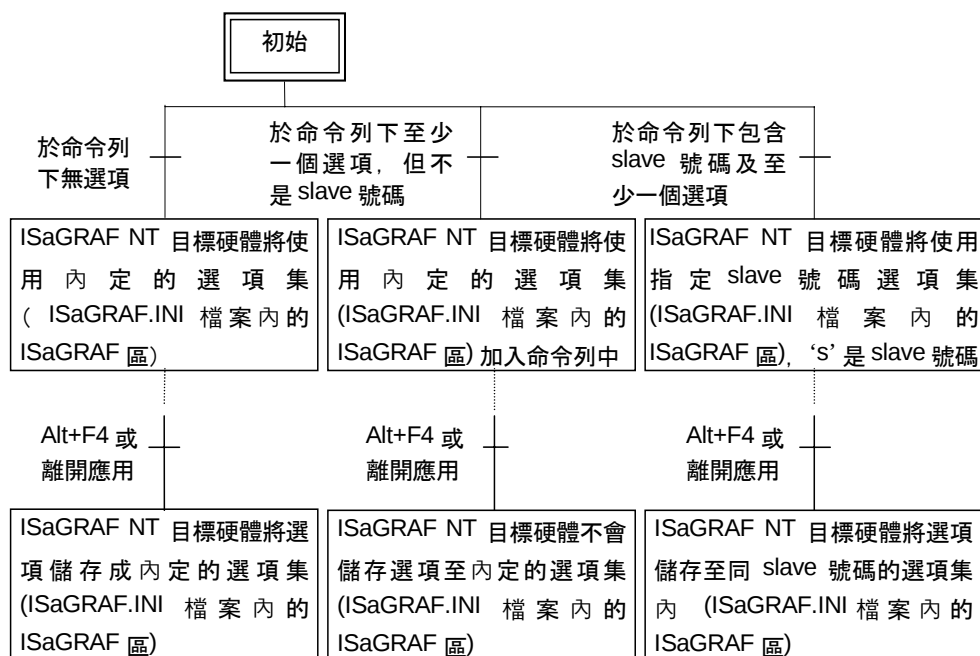
在 NT 的執行架構中，目標硬體執行單一的程式：WISAKER.EXE，它可以被呼叫起來執行很多次。也就是允許可以有很多的 NT 目標硬體，每一個有不同的 slave 號碼。

目標硬體程式不會防止以中斷啟動的函數執行。

WISAKER 軟體設計於 Windows NT 3.51 或更新版本上執行。

C.6.2 一般的選項訊息

選項根據下面的流程圖來儲存及回復：



注意 ISaGRAF.INI 檔案儲存於現在的工作目錄。

▣ Slave 號碼：-s 選項

這個選項指定目標硬體的 slave 號碼。它可以是除了 13 (\$0D) 之外的 1 至 255 的數字。這個號碼用於通訊連結格式中。當多個目標硬體連到同一個工作平台時，或多個目標硬體執行在同一台 PC 上時，它主要設計來區分不同的目標硬體。當使用工作平台的除錯器時，需確定工作平台的 slave 設定 (參考使用者指南：程式管理) 與目標硬體一致。

內定值：內定的 slave 號碼為 1，或根據 ISaGRAF.INI 檔案所設定的值。

例如：

WISAKER.EXE -s=2

使用者介面： ISaGRAF NT 目標硬體主視窗的“Options/Slave”命令可顯示如下的視窗。



使用滑鼠或方向鍵 (上或下) 來改變這個選項的值。爲了使用這個設定值，ISaGRAF NT 目標硬體必須重新開始。

⇒ 通訊連結與組態：-t 選項

ISaGRAF 目標硬體能使用序列埠或乙太網路來做除錯器的通訊。

通訊出口的名稱用 -t 選項來指定。通訊介面被設計成與任何機器相容，使用 COM1，COM2，COM3 或 COM4 來做為序列埠通訊，用 1100 以上的埠號碼 (port number) 來當做乙太網路通訊。

內定值： 內定的乙太網路通訊的埠號碼 (port number) 爲 1100，內定的序列通訊爲 COM1，或根據 ISaGRAF.INI 檔案來設定。

注意：內定的通訊連結爲乙太網路。

例如：

WISAKER -t=COM2

WISAKER -t=1101

序列組態：

假如指定選項爲 -t=COMx，則僅能使用一些選項。

下面是序列連結的組態選項：

選 項	值	意義

<i>b</i>	6	鮑率
<i>a</i>	0	
<i>u</i>	0	
<i>d</i>	1	
	2	
	0	
	0	
	2	
	4	
	0	
	0	
	4	
	8	
	0	
	0	
	9	
	6	
	0	
	0	
	1	
	9	
	2	
	0	
	0	
<i>p</i>	<i>n</i>	無同位
<i>a</i>	<i>e</i>	元
<i>ri</i>	<i>o</i>	偶數
<i>t</i>		奇數
<i>y</i>		
<i>D</i>	7	資料位
<i>a</i>	頭	元數
<i>t</i>	8	
<i>a</i>		

目標硬體使用者指南

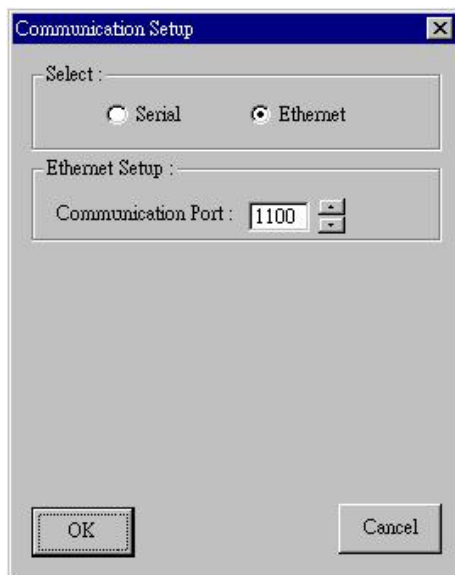
s	1	停止位
t	8	元長度
o	2	
p		
fl	h	硬體控
o	n	制
w		無控制

內定值為 19200，無同位元，8 個資料位元，1 個停止位元，無流量控制。

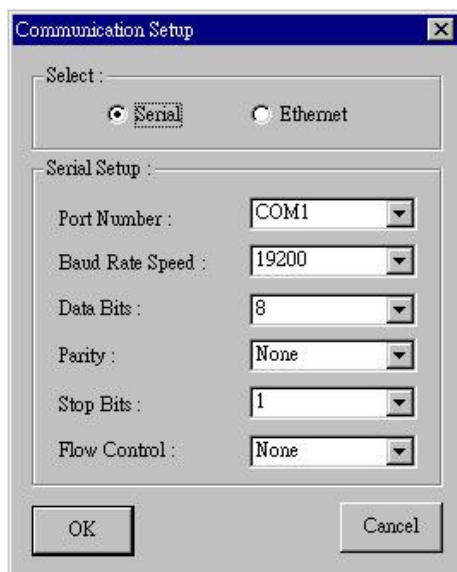
例如：

WISAKER -t=COM1 baud=1200 data=8 parity=n stop=2

使用者介面：ISaGRAF 主視窗的“Options/Communication”命令可顯示如下的視窗。



你可選擇使用序列通訊或乙太網路通訊。乙太網路通訊可以變更埠號碼。埠號碼須與工作平台 PC-PLC 連結所指定的值相同。



若選擇序列通訊，將出現組態設定畫面。設定畫面內的值必須與工作平台 PC-PLC 連結所指定的值相同。

⇒ 虛擬板的圖形模擬：-x 選項

假如這個選項被設定，於 I/O 連結編輯器 (參考 A 部分) 所定義的板子將使用虛擬板來做模擬。

可以設定成 0 或 1，0 表示不做模擬，1 表示開啓模擬。

內定值： 內定值為 0，或根據 ISaGRAF.INI 檔案所設定。

例子：

WISAKER -x=1 將以虛擬板模擬。

使用者介面： 功能表內的項目將被打鉤或未打鉤，用來反應選項的狀態。模擬板出現於圖形面板中。

⇒ ISaGRAF NT 目標硬體的優先權：-p 選項

目標硬體使用者指南

於 NT 上執行的目標硬體指定優先權準位是非常有用的。譬如可以有一個目標硬體的應用以較高的優先權來執行，一個或多個目標硬體應用以較低的優先權放於背景中執行。

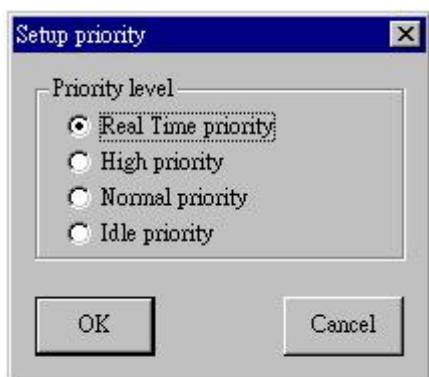
可以設定的值有 0、1、2 及 3。0 是最高的優先權，3 是最低的優先權。

例子：

WISAKER -p=0

WISAKER -p=1

使用者介面： 從 ISaGRAF NT 目標硬體的主視窗下的“Options/Priority”命令來顯示這個視窗。



最高的優先權為即時處理，最低的優先權是閒置的狀態。

0：即時優先權

1：高優先權

2：一般優先權

3：閒置優先權

例子：

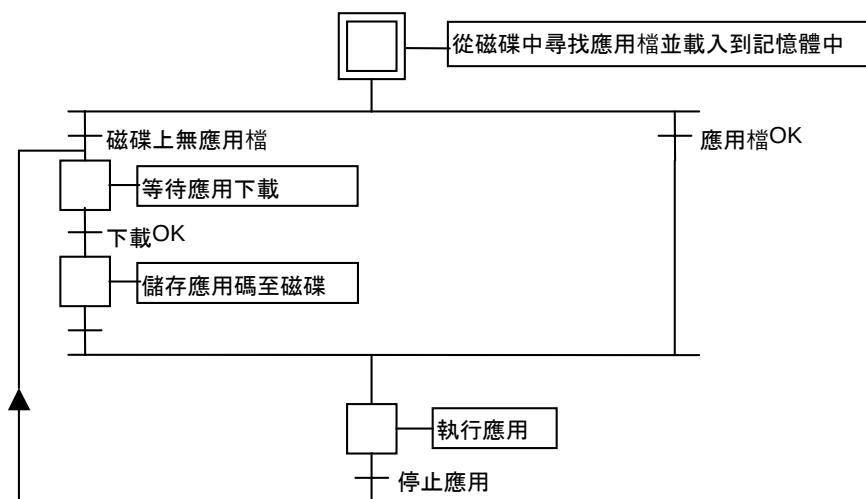
wisaker -t=COM1 開始執行 ISaGRAF 目標硬體，使用內定的 slave 號碼 (1) 與用 COM1 當做它的通訊埠。

wisaker -s=3 -t=COM1 開始執行 ISaGRAF 目標硬體，使用 slave 號碼 3 與 COM1 當做它的通訊埠。

C.6.3 特殊事項

▣ ISaGRAF 啓始

當目標硬體開始時，會執行如下的規則：



• 定義

應用程式碼是一種二元資料，由工作平台產生及下載，然後由目標硬體執行。若加上符號表會更完整。

應用符號表是一種ASCII 資料，由工作平台產生及下載。這個表格產生符號物件與內部目標硬體物件之間的連結關係。平常是不需要這個表格的，除了使用者需要用到符號管理時才會用到。譬如用到 DDE 元件時或 I/O 虛擬板上用到符號名稱時。關於符號表的詳細說明請參考使用者指南的“進階程式技術”章節。

• ISaGRAF 多工應用

不同的應用程式可以在同一個 CPU 上執行，只要設定不同的 slave 號碼與不同的通訊邏輯號碼。但是當不同的應用共享相同的物件時，使用者就得小心了，除非有提供如 I/O 伺服器或 semaphore 機構的情況下可以共用相同的物件，否則不同的應用還是得使用不同板子。

目標硬體使用者指南

- 應用程式備份

當工作平台除錯器下載新的應用程式到目標硬體後，應用程式碼會以如下檔案名稱儲存在目標硬體現在目錄下：

ISAx1 ISaGRAF 應用程式碼備份檔 (x 是 slave 號碼)

若應用符號表先前已下載過，亦會在目標硬體目前目錄下儲存成如下的檔案名稱：

ISAx6 ISaGRAF 應用符號備份檔 (x 是 slave 號碼)

當 ISaGRAF 目標硬體開始時，會搜尋現在目錄下的應用程式檔及應用符號檔，再將它們載入到記憶體中。

假如符號檔不存在，目標硬體會執行應用程式碼，但不載入符號表。

假如應用程式碼不存在於記憶體中，目標硬體會等待應用程式碼被下載。

爲了於開機後執行指定的應用，而不使用除錯器連結的方法，可以直接將這些檔案從硬碟（假如工作平台位於同一台 PC 上）或磁碟片複製到目標硬體的工作目錄下。

假如 ISaGRAF 工作平台安裝於標準的 \ISAWIN 目錄下：

案件 MYPROJ 的應用程式碼檔案是：

\ISAWIN\APL\MYPROJ\appli.x8m

案件 MYPROJ 的應用符號檔案是：

\ISAWIN\APL\MYPROJ\appli.tst

例如：

於 WISAKER.EXE 所在的目錄下，輸入如下的命令：

copy \ISAWIN\APL\MYPROJ\appli.x8m isa11

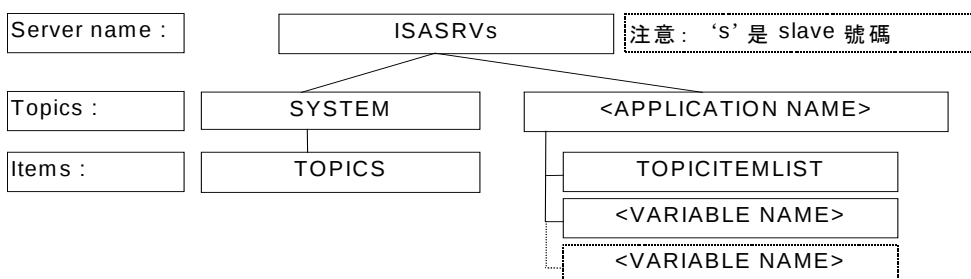
然後 WISAKER.EXE 將會找到並執行 'myproj' 應用。

所有的命令能群聚成一個批次檔，然後可於工作平台工具功能表下執行它（參考使用者手冊：程式管理）。

二 DDE 格式

ISaGRAF NT 目標硬體是一個 DDE (動態資料交換 : Dynamic Data Exchange) 伺服器。任何的客戶端 (client) 軟體都可以與目標硬體連結，來做變數的交換。譬如，Microsoft EXCEL 能透過 DDE 讀取 ISaGRAF 的變數值來做圖形的模擬。使用 DDE 需要於目標硬體上放入應用符號表。

DDE 主題定義如下：



« ISASRVs » 是 DDE 伺服器的名稱，'s' 是 slave 號碼

« SYSTEM » 是標準的 DDE 主題 (topic)

« TOPICS » 給予目前定義的主題集合：執行於 ISaGRAF NT 目標硬體的系統及應用的名稱

« APPLICATION NAME » 是應用程式的名稱

« TOPICITEMLIST » 是於目前主題下的項目集合，這個主題給予能經由 DDE 進入的變數集

« VARIABLE NAME » 是變數名稱

ISaGRAF NT 目標硬體 DDE advise 迴圈速率：-d 選項

DDE 客戶端 (client) 一般在每次需要的時候擷取變數值。假如變數數目很多，這樣可能花費大量的時間。可用另一種叫做 advise 的模式 (advise 迴圈)，這種模式伺服器只有在變數變動時再傳送資料，所以可以達到最小且最有效率的通訊。用這種模式時，伺服器會週期地巡視被標為 advise 變數的變數，以得知那些變數應該被傳送。這個週期稱為 DDE advise 迴圈速率。

這個選項可以指定 DDE advise 迴圈的速率 (以 ms 為單位)。

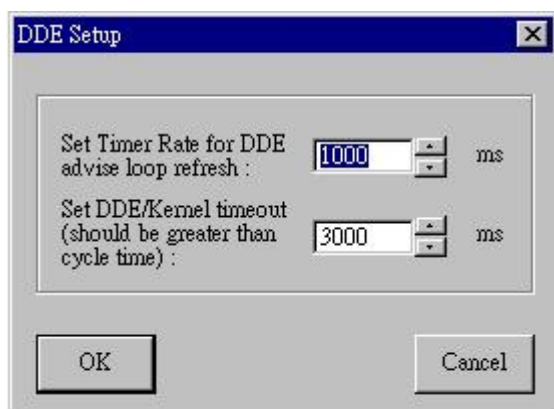
目標硬體使用者指南

內定值： 內定值為 1000ms 或根據 ISaGRAF.INI 檔案所指定。

例如：

WISAKER -d=100

使用者介面： ISaGRAF NT 目標硬體主視窗的“Options/DDE”命令顯示如下的視窗。



❏ 錯誤管理與訊息輸出

ISaGRAF 目標硬體軟體整合了錯誤偵測管理。你可以在附表中找到警告錯誤表及說明。

錯誤偵測程序如下：

錯誤由錯誤及引數號碼所組成，傳送至 ISaGRAF 的錯誤處理函數處理。

假如工作平台產生選項內的錯誤偵測旗標被設定，則錯誤將被處理。否則錯誤訊息將遺失，且錯誤管理程序停止。

當處理時：

錯誤號碼（十進位值）與引數（十六進位值）顯示在輸出上（WISAKER.EXE 視窗上）。

錯誤號碼與引數放進 FIFO 錯誤緩衝區的環狀結構中，是爲了方便於稍後回復出來。錯誤緩衝區的大小可由工作平台製作選項來設定。當緩衝區滿了，每次新的錯誤進來會把最舊的錯誤排擠掉。

錯誤能由除錯器或執行中的應用 (用 SYSTEM call，參考使用者指南) 呼叫出來。

當除錯器偵測到錯誤時，錯誤說明訊息將顯示到錯誤視窗中。除錯器視應用的狀態 (執行與否) 顯示錯誤來源到物件名稱 (變數或程式)，或放入方括弧 [x] 中的錯誤引數 (十進位值)，不同的引數號碼代表不同的意義。

當目標硬體開始時，會在輸出上顯示歡迎的訊息，由 slave 號碼，通訊組態與 DDE 伺服器名稱所組成。

□ 系統時鐘

ISaGRAF NT 目標硬體被設計成可於任何系統上執行，參考時間使用標準的 tick (10ms)，它用來做爲週期同步與計時器變數更新用。

也就是說，計時器變數的正確率不可能比 10ms 小。同樣地，指定週期時間小於等於 10ms 且不是 0，將產生週期區間過載 (編號 62 的錯誤)。參考下一章以得到更多的說明。

假如你的應用需要更正確的說明，請詢問你的供應商。

□ 週期間隔與目標硬體行爲

ISaGRAF 在週期的結尾 (開始新的週期之前) 實行以下的規則：

假如週期時間被指定 (從工作平台指定，參考使用者指南的程式管理章節)，CPU 會將剩餘的時間區段 (指定的週期時間減應用程式所用的時間) 作廢。假如剩餘時間爲負值，將會產生時間過載，然後 CPU 釋放一個 tick 時間來強制這個排程。

假如沒有指定週期時間，或剩餘時間小於一個 tick 或等於零，CPU 會釋放一個 tick 時間來強制這樣的排程。

目標硬體時間正確率相當於一個 Windows NT 的 tick 時間。

指定週期時間通常用在觸發固定的週期時間，或讓出 CPU 給其他執行於 Windows NT 系統的程序。

⇒ 離開鍵

當於非工業用的桌上型 PC 上測試應用程式時，使用者可能希望停止 ISaGRAF：按一個組合鍵來防止非期待的停止。這個按鍵是：

alt + F4

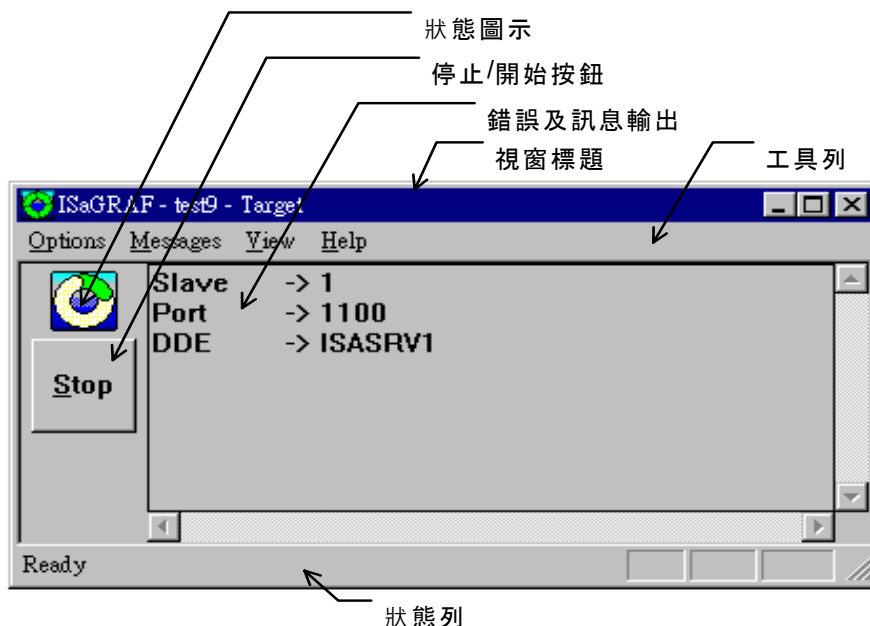
這個快速離開方法有個危險的影響，就是 I/O 板介面未關閉。可以用另一種乾淨的方法來停止你的 ISaGRAF 目標硬體：

從除錯器或從 “Start/Stop” 按鈕停止應用程式（會一併關閉 I/O 板）

從系統功能表停止 ISaGRAF 目標硬體

C.6.4 使用者介面

下面是 ISaGRAF NT 目標硬體的使用者介面：



下面是主要的項目：

- 一個視窗標題
- 一個功能表列
- 一個執行狀態圖示
- 一個“開始/停止”按鈕
- 一個錯誤及訊息輸出
- 一個狀態列

視窗標題包含 « ISaGRAF – 應用程式名稱 – 目標硬體 », 應用程式名稱為執行中的應用程式。當沒應用程式在執行時, 則僅為 « ISaGRAF – – 目標硬體 »。

ISaGRAF NT 目標硬體功能表列：

功能表包含四個功能表：

- Options (選項)
- Messages (訊息)
- View (檢視)
- Help (說明)

目標硬體使用者指南

- “Options” 功能表

(亦可參考 NT 上的第一部份：選項的一般說明)

“Options” 功能表給予執行的選項設定，包含下面的選項：

Slave 可以變更 slave 號碼。變更後的值於下次目標硬體啟動才會生效。假如目標硬體以至少一個選項的命令列來啟動後，這個命令無效。

Communication (通訊) 可設定通訊組態的值。變更後的選項將於下次目標硬體啟動才會生效。假如目標硬體以至少一個非 -s 選項的選項的命令列啟動後，這個命令無效。

DDE 可變更 DDE advise 迴圈速率。變更後的值將於下次目標硬體啟動才會生效。假如目標硬體以至少一個非 -s 選項的選項的命令列啟動後，這個命令無效。

Simulate I/O (模擬 I/O) 可打開或關閉這個選項。變更後的選項僅有在下次應用程式停止/開始之後才生效。

Priority (優先權) 可變更優先權的值。變更後的選項會立即生效。

Default Options (內定選項) 僅回復下述的選項成為內定的執行選項：

通訊

DDE

視窗於螢幕的座標

變更後的選項將於下次目標硬體啟動後才會生效。假如目標硬體以至少一個非 -s 選項的選項的命令列啟動後，這個命令無效。

- “Messages” 功能表

“Messages” 功能表是輸出的管理，包含如下的命令：

Acknowledge (認知) 停止因錯誤或訊息產生的紅色閃燈。

Clear (清除) 清除整個輸出

ISaGRAF NT 目標硬體圖示：

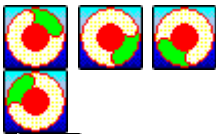
這個圖示反應目標硬體的狀態：

應用程式執行後，圖示啟動

沒有應用程式 (或應用停止)，圖示停止

有錯誤或訊息輸出到輸出視窗上。圖示的中心閃紅燈。要停止閃爍，可選擇 « Messages » 功能表的 « Acknowledge » 項目，或相同功能表的 « Clear » 項目 (它也會一併清除輸出視窗上的訊息)。你可以於錯誤管理與訊息輸出章節找到關於錯誤訊息的更多說明。

不同的狀態列於下列的表格中：

	沒有錯誤	錯誤或有訊息 (中心為紅色的)
應用程式執行中		
沒有應用程式		

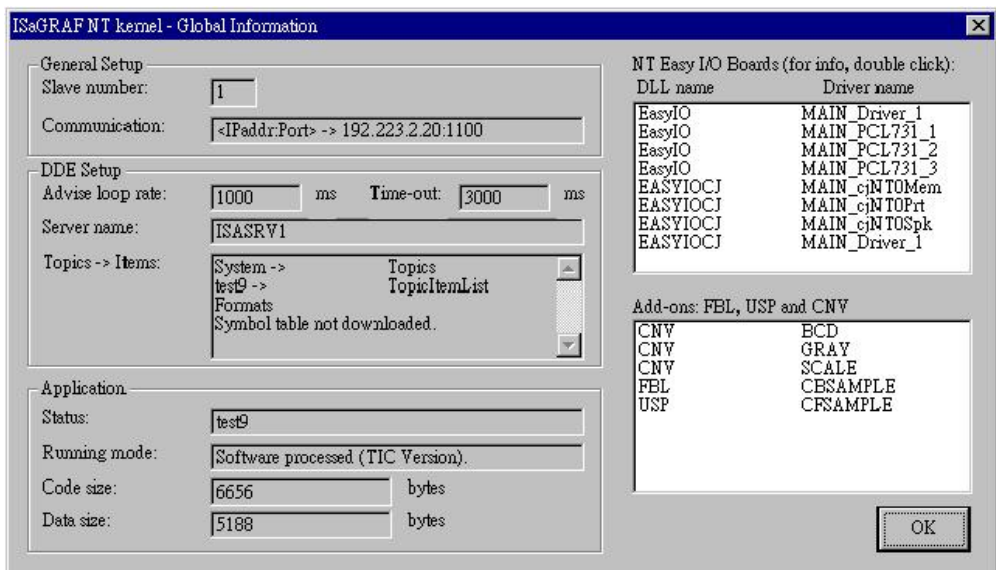
ISaGRAF NT 目標硬體 開始/停止 按鈕：

“Start/Stop” 按鈕與除錯器的“開始/停止”功能完全一樣。按鈕上的文字即為應用程式的執行狀態。假如應用程式處於執行中，將會出現 « Start » 文字，假如應用程式停止 (或沒有應用程式)，則會出現 « Stop » 的文字 (注意，假如沒有應用程式且做了開始的行動後，按鈕將變為停止模式，然後再回到開始模式)。

ISaGRAF NT 目標硬體、一般資訊

目標硬體使用者指南

使用“View / Information”命令，會開啓如下的對話框，顯示目標硬體組態及執行應用的一般資訊：



有三個主題：

一般設定

slave 號碼

通訊組態（假如使用乙太網路當做通訊連結埠，會再加上 NT 系統的埠號碼（port number）與 IP 位址（address））

DDE 設定

advise 迴圈速率

DDE 伺服器名稱

DDE 主題（topic）及項目（item）名稱。它只是一種訊息，不會顯示真實值。實際上介於 < > 欄位內會以真實值取代。

應用程式

當執行應用程式時，應用程式的狀態顯示應用程式名稱，當沒有執行應用程式時，顯示‘No application’字串。

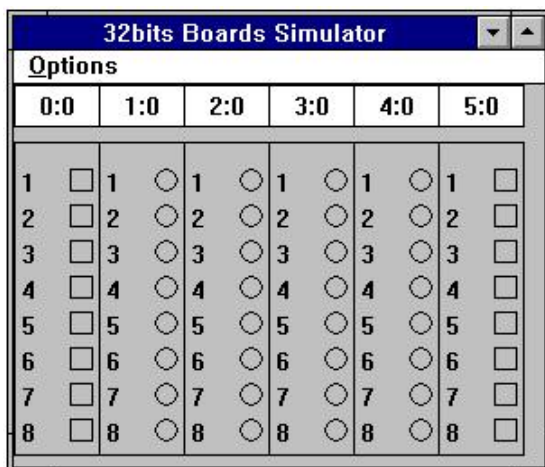
應用的執行模式指出：如果應用透過軟體處理器執行，它以「Software processed」來顯示。假如應用程式被 C 編譯器所編譯過，則顯示「C compiled」。如果沒有應用被執行，顯示「No application」字串。

程式碼大小以位元組為單位。假如執行模式為 « C compiled », 這個欄位為零。

資料大小以位元組為單位，它是執行期內部資料及變數資料庫的總和。

⇒ ISaGRAF NT 目標硬體虛擬板模擬：

當 « Simulate I/O » 選項被選擇，在下次應用程式開始時，會出現如下的視窗：



視你的 I/O 連結組態而定，就會出現多少的板子及變數。在每個板子的頂端有個 « s:b » 的編號，表示槽位編號 (s) 及板子編號 (b)，從零開始計數，且不能變更。

‘32bits Board Simulator’ 視窗會根據應用的開始/停止狀態來運作。所以假如有個包含虛擬板 (或使用模擬器板) 的執行應用程式且 « Simulate I/O » 旗標開啓，這個視窗將出現。相反地，Stop 按鈕按下後，這個視窗立即關閉。它只能用在 I/O 呼叫時。

“Options” 功能表有兩個項目：

Variable names (變數名稱) 選項僅有在符號表先於 TIC 碼之前下載後才會顯示出變數名稱。

目標硬體使用者指南

Hexadecimal values (十六進位值) 選項將以十六進位值取代內定的十進位值來表示整數值。

變數名稱看起來如下：

32bits Boards Simulator											
Options											
0:0		1:0		2:0		3:0		4:0		5:0	
1	☐ ROW0	1	☐ LED00	1	☐ LED10	1	☐ LED20	1	☐ LED30	1	☐ COL0
2	☐ ROW1	2	☐ LED01	2	☐ LED11	2	☐ LED21	2	☐ LED31	2	☐ COL1
3	☐ ROW2	3	☐ LED02	3	☐ LED12	3	☐ LED22	3	☐ LED32	3	☐ COL2
4	☐ ROW3	4	☐ LED03	4	☐ LED13	4	☐ LED23	4	☐ LED33	4	☐ COL3
5	☐	5	☐	5	☐	5	☐	5	☐	5	☐
6	☐	6	☐	6	☐	6	☐	6	☐	6	☐
7	☐	7	☐	7	☐	7	☐	7	☐	7	☐
8	☐	8	☐	8	☐	8	☐	8	☐	8	☐

C.7 “C” 程式設計

C.7.1 概觀

這個章節針對已有 ISaGRAF 觀念及工作平台工具經驗的使用者。你可以使用 ICS Triplex ISaGRAF 提供的標準程式庫內的轉換函數、“C” 函數及功能方塊來發展單純的自動化應用，也可以發展“使用者定義”的轉換函數、“C” 函數與功能方塊。藉由創造新的程式庫允許使用者加強 ISaGRAF 目標硬體 PLC，以及得到硬體平台的最大效益及彈性。

若你有“C”發展系統及“C”程式設計的經驗，這手冊將帶領你設計你的 ISaGRAF 目標硬體 PLC 成為最佳的控制器。如此的發展方式改進了目標硬體 PLC 的性能，亦增進了自動化程式設計者用 ISaGRAF 工作平台的發展品質及便利性。

這個手冊並非只針對專門的目標硬體系統來說明，然而某些特性（如多工能力）不能用在單工系統上。

▣ 標準的 ISaGRAF 工作平台特點

ISaGRAF 工作平台於自動化發展方面提供許多函數來管理“C”元件程式庫。對自動化程式設計而言，“C”轉換函數、函數或功能方塊本身是個“黑箱子”，必須藉由定義它的介面來完成。

ISaGRAF 程式庫管理員用來加入元件到已存在的程式庫中，以及定義“C”實行程式與在 ST/FBD 程式中使用這些元件的介面。ISaGRAF 程式庫管理員亦提供轉換、函數與功能方塊的“C”原始碼自動產生，以及“C”原始碼檔案的編輯工具。參考 **ISaGRAF 使用者指南** 以得到關於程式庫管理員功能的更多說明。

□ “C” 語言發展

ISaGRAF 工作平台不提供任何的“C”編譯器或交互編譯工具。使用者必須擁有用於 ISaGRAF 目標硬體系統的“C”編譯器，用來整合自己的“C”元件到 ISaGRAF 核心程式裡面。

當使用交互編譯器時，ISaGRAF 工作平台提供使用者於 DOS 視窗內執行自己定義的 MS-DOS 命令檔案 (.bat) 的進入點。交互編譯器必須執行於 DOS 模擬視窗中，或者關閉 Windows，進入純 MS-DOS 環境再執行編譯器或連結器。

□ 技術摘要

ISaGRAF 程式庫管理員允許使用者編寫程式庫元件的文字說明，這個技術摘要是“C”元件發展的使用者指南，且助益於自動化程式設計者，它用來敘述於 ISaGRAF 應用程式對應的轉換、函數或功能方塊。

轉換、“C”函數或功能方塊必須確實的定於技術摘要中，以便自動化程式設計者能真的將它當做一個包裝過的 ISaGRAF 函數來使用。對“C”函數而言，技術摘要必須敘述：

函數的詳細功能

呼叫參數的完整說明

回傳值的意義

呼叫及回傳參數的詳細型態

應用內容

對“C”功能方塊而言，技術摘要必須敘述：

功能方塊的詳細功能

呼叫參數的完整說明

回傳參數的意義

呼叫及回傳參數的詳細型態

應用內容

對轉換函數而言，技術摘要必須敘述：

用於輸入變數時的確切轉換意義

用於輸出變數時的確切轉換意義

轉換能處理的上下限值

技術摘要亦可以包含如下的說明：

轉換、函數或功能方塊的完整識別

關於維護及更新的任何訊息

支援的目標硬體系統

特別的多工特性

所需的系統服務、記憶空間、驅動程式...

C.7.2 “C” 轉換函數

ISaGRAF 工作平台包含**線性轉換**的公用程式，用來實行 ISaGRAF 目標硬體 PLC 執行期的簡單類比 I/O 轉換。這個公用程式不需任何的“C”發展程式，但是限制在嚴格的漸增或漸減的連續函數。參考 ISaGRAF 使用者指南，可得知這些工具的完整說明。

轉換函數讓使用者能以“C”語言的特定運算式來做任何複雜的轉換。基本上，轉換函數可定義給**輸入及輸出兩者**來用。即使只使用在單方面（輸入或輸出）上，在整合到 ISaGRAF 核心程式之前必須加以實行及測試，以防止錯誤呼叫時產生系統損毀。

轉換函數以“C”語言所撰寫，編譯後連結到 ISaGRAF 核心程式。於 ISaGRAF 案件中使用新的轉換函數之前，必須將增加功能的核心程式安裝到 ISaGRAF 目標硬體 PLC 內。但是新的轉換函數不能整合到 ISaGRAF 模擬器中。所以在插入非標準的轉換函數**之前**，ISaGRAF 應用必須被模擬好。

ICS Triplex ISaGRAF 所寫的標準轉換的“C”原始碼已安裝於 ISaGRAF 工作平台中，你可以用它當做造新函數的範例。爲了能將標準函數適用到任何的 ISaGRAF 應用程式中，**最好不要變更它**。ISaGRAF 工作平台提供的標準轉換支援 ISaGRAF 模擬器。

警告： 轉換函數於應用程式週期的輸入或輸出時期做**同步**運算，於執行期間由 ISaGRAF I/O 管理員呼叫起來執行。轉換函數所花費的時間包含於 ISaGRAF 應用程式**週期時間**內。使用者必須確定轉換函數沒有設計“等待運算”指令，否則會造成 ISaGRAF 週期時間不必要的延長。

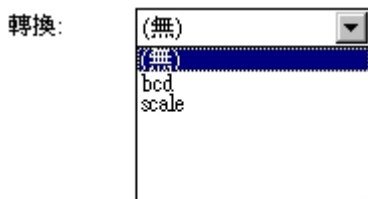
➡ 加入函數到 ISaGRAF 程式庫

ISaGRAF 程式庫管理員用來加入工作平台方面的新的轉換函數到 ISaGRAF 程式庫裡。當轉換函數程式庫被選擇後，“檔案”功能表的“**開新檔**”命令不需在工作平台上定義參數，因為轉換函數使用標準的預定介面。

當新的轉換函數產生後，必須撰寫**技術摘要**。新的轉換函數的“C”原始碼架構將會由 ISaGRAF 程式庫管理員自動產生。

➡ 於 ISaGRAF 案件中使用轉換

定義過的轉換函數可以用來過濾所選案件的輸入或輸出類比變數。欲連結轉換函數至變數，可開啓變數編輯視窗，選擇欲連結的輸入或輸出類比變數，然後編輯它的參數。類比宣告對話盒內的“**轉換**”欄位用來設定類比 I/O 變數的轉換函數：



轉換函數及轉換表皆會出現在表列中，所以轉換函數及轉換表格不能使用相同的名稱。尚未定義或整合到 ISaGRAF 核心程式的轉換函數無法依附至變數上。

▣ 標準 “C” 介面

轉換函數的介面有制式的格式，呼叫及回傳參數透過結構來傳遞，定義於 “TACN0DEF.h” 檔中：

```
/*
檔案名稱: tacn0def.h
目標硬體轉換定義檔
*/

#define DIR_INPUT 0    /* 方向 = 輸入轉換 */
#define DIR_OUTPUT1 /* 方向 = 輸出轉換 */

typedef int32    T_ANA; /* 整數類比型態 */
typedef float    T_REAL; /* 實數類比型態 */

typedef struct { /* 轉換結構 */
    uint16 number; /* 轉換編號 (保留) */
    uint16 direction; /* 轉換方向 */
    T_REAL *before; /* 轉換之前的值 */
    T_REAL *after; /* 轉換之後的值 */
}    str_cnv;

#define ARG_BEFORE (*(arg->before))
#define ARG_AFTER (*(arg->after))
#define DIRECTION (arg->direction)

/* 檔案結束 */
```

“str_cnv” 結構完整地說明介面的格式。“C” 轉換函數的唯一參數是指到這個結構的指標，其中 “number” 欄位是轉換函數的邏輯號碼（可於 ISaGRAF 程式庫中看到），它並不需要用於轉換函數程式中。

目標硬體使用者指南

“**direction**” 欄位指出轉換套用於輸入變數或輸出變數上，可設定成：
DIR_INPUT (用於輸入轉換)，**DIR_OUTPUT** (用於輸出轉換)。

“**before**” 欄位表示轉換之前的值。對於輸入或輸出轉換，所代表的意義不同。當 **direction** 欄位設為 **DIR_INPUT** 時，它代表輸入轉換的電位值 (從輸入設備讀取到的值)。當 **direction** 欄位設為 **DIR_OUTPUT** 時，它代表輸出轉換的物理值 (用於程式式子中)。

“**after**” 欄位表示轉換之後的值。對於輸入或輸出轉換，所代表的意義不同。當 **direction** 欄位設為 **DIR_INPUT** 時，它代表輸入轉換的物理值 (用於程式式子中)。當 **direction** 欄位設為 **DIR_OUTPUT** 時，它代表輸出轉換的電位值 (輸出到輸出設備)。

程式設計者可以透過 “**ARG_BEFORE**” 和 “**ARG_AFTER**” 定義直接進入 “**C**” 轉換函數結構的 **before** 和 **after** 欄位，運算過程以單精度浮點數值來計算。當轉換用於整數型態類比變數上時，運算結果會轉換成長整數值，所以實數或整數類比 I/O 變數都可以使用相同的轉換函數。

ㄟ 原始碼

因為轉換函數可以應用在輸入及輸出類比變數兩者上，所以函數的 “**C**” 原始碼分為兩個主要部分：輸入轉換與輸出轉換。轉換介面結構的 **direction** 欄位用來選擇所應用的轉換型態，當轉換函數創造後，ISaGRAF 程式庫管理員會自動產生完整的函數結構，包含主要的選擇 “**IF**” 結構。下面是轉換函數的標準結構：

```
/*  
conversion function  
name: sample  
  
#include <tasy0def.h>  
#include <tacn0def.h>
```

```
void CNV_sample (str_cnv *arg)
{
    if (DIRECTION == DIR_INPUT) { /* 輸入轉換 */

    }
    else { /* 輸出轉換 */

    }
}
```

/* 下面的函數使用轉換名稱與 ISaGRAF I/O 管理員連結，這個函數完全由 ISaGRAF 程式庫管理員所產生 */

```
UFP cnvdef_sample (char *name)
{
    sys_strcpy (name, "SAMPLE"); /* 給予轉換名稱 */
    return (CNV_sample); /* 回傳函數的執行結果 */
}
```

要完成函數的指定部分，最好的方式是針對輸入轉換及輸出轉換寫兩個分開的區域函數，如上個例子中的註解所示，這些區域函數將在主 **IF** 結構中被呼叫。

ISaGRAF 核心程式中的 “**TASY0DEF.H**” 標頭檔用於與系統有關的定義，它亦包含 **UFP** 型態 (表示無回傳函數的指標) 的定義，可用於宣告函數。

⇒ 案件與 “C” 執行之間的連結

轉換函數的實行與 ISaGRAF 案件的轉換使用間以轉換名稱做為邏輯連結。“宣告” 函數被加入轉換函數的 “C” 原始碼中，這個函數僅在應用程式開始時被呼叫一次，且通知執行函數的轉換函數名稱給 ISaGRAF I/O 管理員知道。下面是宣告函數的標準格式：

目標硬體使用者指南

```
UFP cnvdef_XXX (char *name)
{
    strcpy (name, "XXX");    /* 給予轉換名稱 */
    return (CNV_XXX);    /* 回傳執行的結果 */
}
/* (XXX 是轉換名稱) */
```

上例中，使用於 **strcpy** 敘述的函數名稱必須是**大寫**的，而執行的轉換函數名稱與宣告函數的名稱必須是**小寫**的。

執行函數及定義函數使用 “**CNV_**” 與 “**cnvdef_**” 字首讓使用者以 “C” 語言保留的關鍵字來作轉換的命名，或以 “C” ISaGRAF 程式庫內存在的函數來命名。

其它的敘述式能加入宣告的函數中，用來實行這個轉換任何指定的初始化運算，ISaGRAF 系統會確保應用程式開始後轉換函數僅被呼叫一次。

即使 ISaGRAF 應用程式沒有使用到宣告函數，它依然可以給任何整合的轉換函數所呼叫。假如應用程式使用到的轉換沒有整合到核心程式中，ISaGRAF 核心程式會產生致命的錯誤。

連結新函數到核心程式之前，使用者必須於 “**GRCN0LIB.C**” 中撰寫另外的 “C” 原始檔，且將之（附上可回復的轉換函數）插入於檔案列表中，給連結器 (linker) 知道。 “**GRCN0LIB.C**” 檔案僅包含宣告函數列。這個列表於應用程式初始化時讀取，來製造 “C” 轉換函數的動態連結。檔案內容如下：

```
/* File “GRCN0LIB.c” – Example with conversions of standard library */

#include <asy0def.h> /* required for types definition */

extern UFP cnvdef_scale (char *name);    /* decl function for SCALE
conv */

extern UFP cnvdef_bcd (char *name); /* decl function for BCD conv */
```

```
UFP_LIST CNVDEF[] = { /* array of declaration functions for */
    /* integrated conversion functions */
    cnvdef_scale,
    cnvdef_bcd,

    NULL };

/* end of file */
```

CNVDEF 矩陣必須以 NULL 指標當做結束。當這個條件不滿足時，可能會發生一些衝突。假如 **CNVDEF** 矩陣未定義，於連結新的 ISaGRAF 核心程式時將會發生未解決的參考物錯誤。

藉由撰寫這個檔案的內容，可建立包含所有已存在轉換的新核心程式。也可以僅插入案件用到的轉換於 **CNVDEF** 矩陣中，那麼就可以產生這個案件專用的核心程式。當應用程式碼建立後，ISaGRAF 碼產生器會自動產生“**GRCN0LIB.C**”檔案，它放置於 ISaGRAF 案件目錄下，且僅包含案件所用到的轉換。

限制

ISaGRAF 函數庫最多可包含 **128** 個轉換函數，於轉換函數中可以處理任何型態的運算元。注意，函數於 ISaGRAF 週期中是以**同步進行**方式來呼叫，所以它會直接影響到週期時間。

C.7.3 “C” 函數

“C” 函數能用來補強 **ST** 及 **FBD** 語言的能力。利用 “C” 函數可完成某些特殊的計算、作業系統的函數呼叫、通訊程式、或於多工系統中建立與其他程式的溝通管道。此類函數使用 “C” 語言撰寫、編譯，並與 ISaGRAF 的核心程式連結，完成後的新核心程式需植入 ISaGRAF 的目標硬體 PLC 中，而後才能於 ISaGRAF 新撰寫的案件中使用這些新增加的函數。

這些新撰寫的“C”函數無法於 ISaGRAF 模擬器中產生作用，因此若有模擬的必要，應在使用這些非標準的“C”函數之前模擬所撰寫的 ISaGRAF 程式。

警告：ISaGRAF 中所用到的函數皆為**同步**操作模式，皆經由核心程式於執行時呼叫而驅動，亦即函數執行所花費的時間會累計於核心程式的**週期時間**之中。因此撰寫相關函數時應注意不要在函數中加入“等待 (wait operation)”的指令，以避免核心程式的執行週期時間不必要的延長。

⇒ 在 ISaGRAF 程式庫中加入新的函數

要加入新的“C”函數必須執行工作平台中的 ISaGRAF 程式庫管理員 (ISaGRAF Library Manager)，執行後先選取“C 程式庫”，再於“檔案”功能表中選取“開新檔”，如此便可增加一個新的“C”函數。完成後應在相對的**技術註記**中填入此函數的相關資料。C 原始碼及函數基本結構皆會自動產生，毋需使用者另行編輯。

“編輯”功能表中的“參數”命令可用來定義此函數於呼叫或結束時所要輸入及輸出的參數。

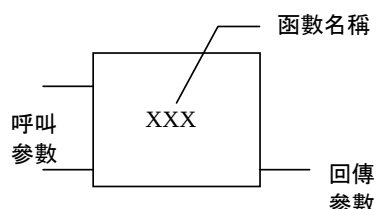
⇒ 在 ISaGRAF 案件中使用“C”函數

在 ISaGRAF 應用程式中任何一個“C”函數皆可視為一個標準的函數，可經由 **ST** 或 **FBD** 語言加以呼叫執行，或由 **SFC** 語言中某些特殊的指令來呼叫執行。

從 **ST** 語言中呼叫“C”函數的語法與呼叫其他函數的語法相同。呼叫時所要輸入的參數可寫在函數名稱後的括弧中，並以逗點隔開，而函數本身便代表其回傳的值。“C”函數適用於任何 **ST** 語言的命令敘述中，可用於給定某個變數的數值，或用於複雜的表示式。以下是個利用呼叫“C”函數來給定某變數數值的例子：

```
result := ProcName (par1, par2, ... parN);
```

FBD 程式亦可呼叫任何的“C”函數，用法就如同使用標準的功能方塊一樣，其中要輸入的參數就連接在功能方塊的左邊，而回傳的值就連接在功能方塊的右邊。以下便是功能方塊的標準外觀：



“C”函數亦可從 **SFC** 的行為方塊中呼叫執行，或於連接至轉移條件的布林判斷中加以呼叫。

▣ 定義“C”函數的介面

程式庫管理員 (ISaGRAF Library Manager) 中“編輯”功能表的“參數”命令可用來定義函數於呼叫或結束時所要輸入及輸出的參數。一個函數在呼叫時最多可輸入 **31** 個參數，並且固定會回傳一個數值。

此視窗最上面方塊中列出函數的參數名稱，其順序依照函數的原型定義而定。第一個是呼叫函數時所要輸入的參數，最後一個是函數回傳的數值。視窗下半部的方塊顯示所選取到參數其細部的描述，包括：

參數名稱

參數的方向 (輸入/輸出)

參數的型態

參數的型態必須使用 ISaGRAF 所支援的變數型態，包括：布林、整數類比、實數類比、計時器或訊息，其中整數與實數類比應區別開來。

以下是 ISaGRAF 變數型態與“C”的比較表：

有	unsigne	不帶符號 32 位元字組: 1=真 / 0=
有	d long	假
無	long	帶符號 32 位元整數字組

字元		
單精度浮點數	float	單精度浮點數值
帶符號 32 位元整數字組 (單位為 1 毫秒)	unsigned long	
字元字串	char *	字元字串

當訊息型態的數值輸入 “C” 函數時，其中不能包含空字元 (null character)，因輸入 “C” 函數的訊息字串是以空字元作為字串的結束字元。此外不要忘了必須將函數回傳的參數值擺在參數列表中的最後一個。以下是替參數命名時所要遵守的規則：

命名的長度不可超過 16 個字元

第一個字元必須是英文字母

其餘的字元必須同樣是英文字母、數字或底線

命名與大小寫無關

同一個函數中不同的參數不可使用相同的名稱，輸入與輸出的參數名稱也不可重複，但不同函數所使用的參數名稱則可以重複。預設的回傳參數名稱為 “Q”，使用者可以任意加以改變。參數名稱的用途是用來辨認 “C” 原始碼中不同的參數。

“插入” 命令是用來在選取參數之前插入一個新的參數，而 “刪除” 命令則是用來刪除目前所選取的參數。“排列” 命令會自動將所有參數重新排序，完成後會將函數回傳的參數擺在參數列表中的最後一位。執行 “確定” 命令會儲存函數界面的相關資料並關閉此對話視窗。執行 “取消” 命令則不會儲存任何資料並直接關閉此對話視窗。

□ C 函數界面

函數的界面是依據其所使用的參數而定。輸入與輸出的參數皆透過 “C” 中的結構來運作。結構定義在 “GRUS0nnn.H” 檔中，其中 “nnn” 代表

ISaGRAF 程式庫中函數的編號。以下是 “SIN” 函數的 “C” 界面範例 (計算 sines 值)：

```
/* File: GRUS0255.h – function “sample” */
```

```
typedef long      T_BOO;
typedef long      T_ANA;
typedef float     T_REAL;
typedef long      T_TMR;
typedef char      *T_MSG;

typedef struct {
    /* CALL */ T_REAL  _param1;
    /* RETURN */ T_REAL _param2;
} str_arg;

#define PARAM1      (arg->_param1)
#define PARAM2      (arg->_param2)

/* end of file */
```

下表是 ISaGRAF 與 “C” 的變數型態對照表。ISaGRAF 的型態定義如同 “C” 一般定義在函數的定義檔中。

布林	T_BOO	long (32 位元)
整數 類比	T_ANA	long
實數 類比	T_REAL	float (32 位元 – 單精度)
計時 器	T_TMR	long
訊息	T_MSG	char * (32 位元 – 字元指標)

	SG	
--	----	--

“str_arg” 結構中的每個欄位皆對應到函數的一個參數。回傳的參數擺在結構中的最後一位，而輸入的參數皆依照函數定義時參數順序而排列。其中英文大寫的參數定義直接指向對應到 “C” 函數結構中的參數，而其名稱是在 ISaGRAF 程式庫管理員中定義函數時所輸入的。

每當函數的界面有所更動時，其對應的 “C” 檔案便會隨之更新，如此便能確保函數內容與 ISaGRAF 應用程式的一致性。

⇒ 原始程式

以下是 “C” 函數的標準架構：

```
/* Example of user function – Number is “255” – Name is “SAMPLE” */
```

```
#include “tasy0def.h” /* ISaGRAF kernel common definitions */
```

```
#include “grus0255.h”      /* interface definition for function 255 */
```

```
void USP_sample (str_arg *arg)
```

```
{
```

```
    /* body of the function */
```

```
}
```

```
/* The following function is used for the initialization of the function and the  
declaration of its implementation. It realizes the link with the ISaGRAF  
kernel, using the name of the function. This function is completely  
generated by the ISaGRAF Library Manager. */
```

```
UFP uspdef_sample (char *name)
```

```
{
```

```
    strcpy (name, “SAMPLE”);    /* gives the name of the function */
```

```
    return (USP_sample);    /* returns the implementation function */
```

```
}

/* end of file */
```

其中 “TASY0DEF.H” 檔必須內含以便載入跟系統有關的定義，此檔亦包含了 **UTP** 型態的定義，而此型態代表指向 **void** 函數的指標，用來宣告函數之用。

⇒ 案件與 “C” 函數的連結

“C” 函數與 ISaGRAF 中的應用程式在邏輯上的連結關係便是此函數的名稱。此函數的 “C” 原始碼中會加入一個特別的 “宣告” 函數 (declaration function)，在應用程式開始執行時會被呼叫一次，用來通知 ISaGRAF 核心程式此 “C” 函數的名字。以下是此宣告函數的標準格式：

```
UFP uspdef_xxx (char *name)
{
    strcpy (name, “XXX”);    /* gives the name of the function */
    return (USP_xxx);    /* returns the implementation function */
}
/* (xxx is the name of the function) */
```

“C” 函數的名稱當被 **strcpy** 命令使用時必須用英文大寫表示。而在函數中及 “宣告” 函數裡則應使用小寫。在撰寫程式及定義程式時，使用 “**USP_**” 及 “**uspdef_**” 作為字首可讓使用者使用一些 “C” 語言中的保留字，或使用 ISaGRAF 程式庫中已經使用過的名稱作為函數的名稱。

在 “C” 函數中的特殊宣告函數裡亦可針對此函數的特性或需要加入一些自定的初始化命令。ISaGRAF 會確保此宣告函數於應用程式開始執行時只被呼叫一次，即使此應用程式並不會用到宣告函數提供的功能。請注意若應用程式使用了未曾被撰寫過及連結的 “C” 函數，則 ISaGRAF 核心程式於執行時會產生嚴重的錯誤。

目標硬體使用者指南

在新的函數與核心程式連結之前，使用者必須編寫另一個 C 的原始程式檔“GRUS0LIB.C”，其中必須插入新的程式名稱以作為連結之用。在“GRUS0LIB.C”檔中只有一個宣告函數的陣列 (array)，而此陣列於應用程式初始化時會被載入，以建立起與函數的動態連結。以下是個範例程式：

```
/* File “GRUS0LIB.c” – Example using trigonometric functions */
```

```
#include <tasy0def.h> /* required for types definition */
```

```
extern UFP uspdef_fc1 (char *name); /* declaration functions */
```

```
extern UFP uspdef_fc2 (char *name);
```

```
extern UFP uspdef_fc3 (char *name);
```

```
extern UFP uspdef_fc4 (char *name);
```

```
UFP_LIST UFPDEF[] = { /*array of declaration functions */
```

```
/* for integrated functions*/
```

```
uspdef_fc1,
```

```
uspdef_fc2,
```

```
uspdef_fc3,
```

```
uspdef_fc4,
```

```
NULL };
```

```
/* end of file */
```

“USPDEF”陣列必須以空字元作為結束字元，否則於程式執行時可能會發生一些不正常的運作。若此“USPDEF”陣列未被定義，則新函數與ISaGRAF核心程式連結時會產生“無法解決的參考物 (unresolved references)”的錯誤。若要發展一些只用於特定應用案例的專屬性核心程式，只需將用到的函數寫入“USPDEF”陣列中，便可將此核心程式製作出來。當應用程式的原始碼被製作時，ISaGRAF的原始碼製作程式會自動產生“GRUS0LIB.C”檔，並放在使用者正在發展的ISaGRAF應用案件目錄下，檔中只包括了應用程式會使用到的函數。

限制

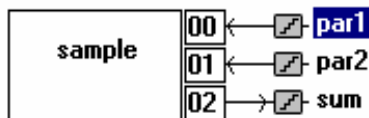
ISaGRAF 程式庫最多可使用 **255** 個 “C” 函數。任何種類的運算或處理皆可於函數中完成，唯一要注意的是這些函數是在一個 ISaGRAF 程式週期中被**同步地**呼叫執行，因此執行函數所需要的時間會直接影響到核心程式的週期時間。

完整的範例

以下是一個取名叫 “sample” 函數的完整程式範例，這函數會執行標準的加法運算。下列是此函數的技術註記：

name:	SAMPLE
description:	簡單的實行整數類比值相加
creation date:	1992 年 7 月 1 日
author:	ICS Triplex ISaGRAF Inc
call:	par1, par2: 整數運算子
return:	sum: 整數相加結果
prototype:	sum := sample (par1, par2);
e:	

以下是此函數的界面：



以下是此函數的 “C” 原始標頭檔：

```
/* File: GRUS0255.h – user C function definitions – Name: sample */
```

```
/* definition of standard ISaGRAF data types */

typedef long T_BOO;
typedef long T_ANA;
typedef float T_REAL;
typedef long T_TMR;
typedef char *T_MSG;

/* definition of the calling and return parameter structure */

typedef struct {
    T_ANA _par1; /* calling parameter #1 */
    T_ANA _par2; /* calling parameter #2 */
    T_ANA _sum; /* return parameter */
} str_arg;

/* identifiers used to access call and return parameters */

#define PAR1    (arg->_par1)
#define PAR2    (arg->par2)
#define SUM     (agr->sum)

/* end of file */


```

以下是此函數的“C”原始程式碼，其中只有以粗黑字體顯示的命令列是使用者必須填入的

```
/* File: GRUS0255.c – user C function – Name: SAMPLE */

#include “tasy0def.h” /*required for types definition */
#include “grus0255.h”    /* C function source header */


```

```

/* C main service: calculates the addition */

void USP_sample (str_arg *arg)
{
    SUM = PAR1 + PAR2;
}

/* declaration service required for dynamic link with ISaGRAF kernel */

UFP uspdev_sample (char *name)
{
    strcpy (name, "SAMPLE");
    return (USP_sample);
}

/* end of file */

```

C.7.4 “C” 功能方塊

“C” 功能方塊包含了運算 (operation) 與靜態資料 (static data)。藉由對靜態物件的處理，“C” 功能方塊可同時具備許多 “C” 函數的功能，並用來增強 **ST** 與 **FBD** 語言的基本能力。不同於函數只對數值做處理，功能方塊可以處理靜態資料，這意味著功能方塊的程式運作能處理資料對時間的變化。

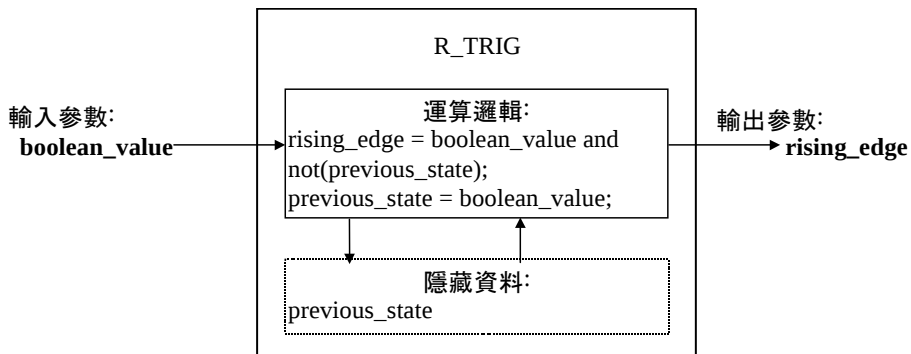
功能方塊用 “C” 語言撰寫，與 ISaGRAF 核心程式一同編譯及連結。在使用新功能方塊前，與新功能方塊連結完成的核心程式必須先植入 ISaGRAF 的目標硬體 PLC 中。這些新撰寫的功能方塊無法於 ISaGRAF 模擬器中產生作用，因此若有模擬的必要，應在使用這些非標準的功能方塊之前模擬所撰寫的 ISaGRAF 程式。

警告：ISaGRAF 中所用到的功能方塊皆為**同步**操作模式，皆經由核心程式於執行時呼叫而驅動，亦即功能方塊執行所花費的時間會累計於核心程式的**週期時間**之中。因此撰寫相關功能方塊時應注意不要在功能方塊中加入 “等

待 (wait operation) ” 的指令，以避免核心程式的執行週期時間不必要的延長。

⇒ 宣告功能方塊樣例

功能方塊是一個包含了運算 (operation) 與靜態資料 (static data) 的物件。下面介紹的是 “R_TRIG” 功能方塊範例，這個功能方塊會偵測布林值的上升邊緣 (rising edge)。以下為這功能方塊的描述：



其中隱藏的變數 “previous_state” 是用來計算上升邊緣之用，而應用程式裡每次使用 “TRIG” 功能方塊時，這個變數皆應不同。ST 語言裡使用的功能方塊必須在變數字典中加以定義；因為功能方塊有內部隱藏的資料，因此複製時皆應給定不同的名稱；方塊的型態名稱可使用程式庫管理員加以命名，而輸出的變數則在變數字典中命名。

FBD 語言中使用的功能方塊無須事先宣告，因為 ISaGRAF 的 FBD 編輯器會自動宣告使用到的方塊，而這些方塊對所編輯的程式而言皆為區域方塊。

⇒ 在 ISaGRAF 程式庫中加入新的功能方塊

在 ISaGRAF 工作平台上要加入一個新的 “C” 功能方塊必須使用到程式庫管理員；於其中選定功能方塊程式庫後，執行 “檔案” 功能表中 “開新檔” 命令，便可新增一個功能方塊。完成後應在相對的技術註記中填入此功能方塊的相關資料，而 C 原始碼及功能方塊基本結構皆會自動產生，毋需使用者

另行編輯。“編輯”功表中的“參數”命令可用來定義此功能方塊於呼叫或結束時所要輸入及輸出的參數。

⇒ 在 ISaGRAF 案件中使用 “C” 功能方塊

任何經整合的 “C” 功能方塊皆可用於 ISaGRAF 應用程式中，經由 **ST** 或 **FBD** 語言加以呼叫。

從 **ST** 語言中呼叫 “C” 功能方塊的語法與呼叫其他功能方塊的語法相同。呼叫時所要輸入的參數可寫在方塊名稱後的括弧中，並以逗點隔開，而回傳的值可按順序依次取得。每個回傳的參數皆由特定名稱來表示，此名稱是由方塊名稱與參數名稱組合而成，並以句點隔開。譬如，下述這名稱：

FBINSTNAME.pname

代表 “**FBINSTNAME**” 功能方塊回傳的 “**pname**” 參數。

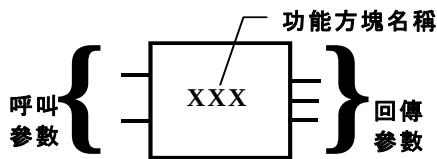
ST 語言裡使用的功能方塊必須在變數字典中加以定義，複製時皆應給定不同的名稱。以下是在 ISaGRAF 變數字典中宣告功能方塊樣例的範例：

樣例：	TRIG1	型	R_TRIG
		態：	
	TRIG2		R_TRIG

以下是在 **ST** 語言中使用這些已宣告過的功能方塊的樣例：

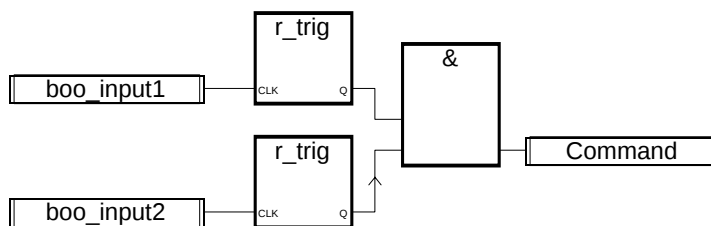
```
TRIG1 (boo_input1);
TRIG2 (boo_input2);
Command := (TRIG1.Q & TRIG2.Q);
```

FBD 程式亦可呼叫任何的 “C” 功能方塊，用法就如同使用標準的功能方塊一樣，其中要輸入的參數就連接在功能方塊的左邊，而回傳的值就連接在功能方塊的右邊。以下便是功能方塊的標準外觀：



FBD 語言中使用的功能方塊無須事先宣告，因為 ISaGRAF 的 FBD 編輯器會自動宣告使用到的方塊，而這些方塊對所編輯的程式而言皆為區域方塊。

以下是前一個例子，但以 FBD 語言來撰寫：



▣ 定義 “C” 功能方塊的介面

ISaGRAF 程式庫管理員 (Library Manager) 中“編輯”選項的“參數”命令可用來定義功能方塊於呼叫或結束時所要輸入及輸出的參數。一個功能方塊最多可有 **32** 個參數，可任意分配給輸入或輸出的參數。不同於“C”函數，功能方塊可擁有許多回傳的參數。

此視窗最上面方塊中列出功能方塊的參數名稱，其順序依照功能方塊的原型定義而定：先是呼叫功能方塊時所要輸入的參數，之後便是功能方塊回傳的參數。視窗下半部的方塊顯示所選取到參數其細部的描述，包括：

參數名稱

參數的方向(輸入/輸出)

參數的型態

參數的型態必須使用 ISaGRAF 所支援的變數型態，包括：布林、整數類比、實數類比、計時器或訊息，其中整數與實數類比應區別開來。

以下是 ISaGRAF 變數型態與“C”的比較表：

布林	unsigned long	不帶符號 32 位元字組：1=真 / 0=假
類比	long	帶符號 32 位元整數字組
實數	float	單精度浮點數值
計時器	unsigned long	不帶符號 32 位元整數字組 (單位為 1 毫秒)
訊息	char *	字元字串

當訊息型態的數值輸入“C”功能方塊時，其中不能包含空字元，因輸入“C”功能方塊的訊息字串是以空字元作為字串的結束字元。此外不要忘了必須將功能方塊回傳的參數值擺在參數列表中的最後一個。以下是替參數命名時所要遵守的規則：

命名的長度不可超過 16 個字元

第一個字元必須是英文字母

其餘的字元必須同樣是英文字母、數字或底線

命名與大小寫無關

同一個功能方塊中不同的參數不可使用相同的名稱，輸入與輸出的參數名稱也不可重複，但不同功能方塊所使用的參數名稱則可以重複。參數名稱的用途是用來辨認“C”原始碼中不同的參數。

“插入”命令是用來在選取參數之前插入一個新的參數，而“刪除”命令則是用來刪除目前所選取的參數。“排列”命令會自動將所有參數重新排序，完成後會將功能方塊回傳的參數擺在參數列表中的最後一位。執行“確定”命令會儲存功能方塊界面的相關資料並關閉此對話框。執行“取消”命令則不會儲存任何資料並直接關閉此對話框。

□ C 功能方塊界面

目標硬體使用者指南

功能方塊的界面是依據其所使用的參數而定。輸入的參數是透過“C”中的結構來運作。結構定義在“GRFB0nnn.H”檔中，其中“nnn”代表 ISaGRAF 程式庫中功能方塊的編號。回傳的參數是由邏輯號碼來表示，亦定義於“GRFB0nnn.H”檔中。以下是“LIM_ALARM”功能方塊的“C”界面範例 (超過限制便產生警告)：

```
/* function block interface – name: sample */
```

```
/* standard ISaGRAF data types */
```

```
typedef    long T_BOO;  
tu[edef    long T_ANA;  
typedef    float T_REAL;  
typedef    long T_TMR;  
typedef    char *T_MSG;
```

```
/* structure of calling parameters */
```

```
typedef struct {  
    /* CALL */ T_BOO _par1;  
    /* CALL */ T_BOO _par2;  
} str_arg;
```

```
/* access to fields of str_arg structure */
```

```
#define    PAR1    (arg->_par1)  
#define    PAR2    (arg->_par2)
```

```
/* return parameter logical numbers */
```

```
#define    FBLPNO_Q1    0  
#define    FBLPNO_Q2    1
```

/* end of file */

下表是 ISaGRAF 與 “C” 的變數型態對照表。ISaGRAF 的型態定義如同 “C” 一般定義在函數的定義檔中。

布林	T_ BO O	Long (32 位元)
類 比	T_ AN A	long
實 數	T_ RE AL	Float (32 位元 – 單精度)
計 時 器	T_ TM R	long
訊 息	T_ M SG	char * (32 位元 – 字元指標)

“str_arg” 結構中的每個欄位皆對應到功能方塊的一個參數。參數皆依照功能方塊定義時的參數順序而排列。其中英文大寫的參數定義直接指向對應到 “C” 結構中的參數，而其名稱是在 ISaGRAF 程式庫管理員中定義功能方塊時所輸入的。

回傳參數的編號順序皆依照功能方塊定義時的參數順序而排列，第一個回傳參數的邏輯編號永遠是 0。

在 “C” 原始程式中，回傳的參數皆以文字名稱加以定義而不使用數字，如此即使功能方塊的介面有所變更修改，程式檔亦可非常容易地重新編譯。

每當功能方塊的界面有所更動時，其對應的“C”檔案便會隨之更新，如此便能確保功能方塊內容與 ISaGRAF 應用程式的一致性。

▣ 原始程式

用“C”語言來撰寫功能方塊可分為三個不同步驟：

函數初始化步驟 (initialization service)

函數運作步驟 (activation service)：輸入參數的處理

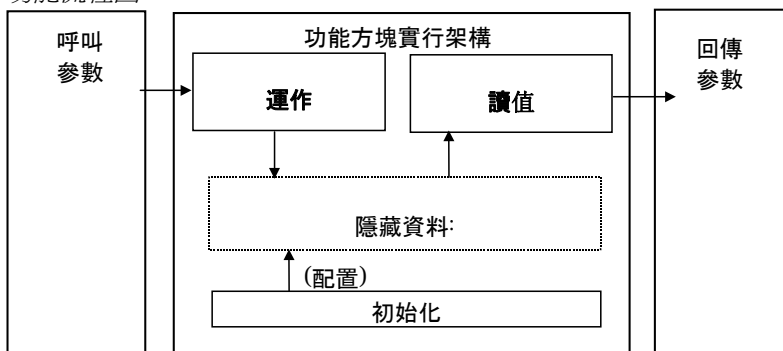
回傳參數讀值步驟 (return parameters read service)

同一種功能方塊使用時會使用相同的程式碼，程式碼並不會被複製；每個被使用的功能方塊皆會伴隨著一個靜態資料，而此靜態資料無法由 ISaGRAF 程式直接取得，且其中並包含了使用到功能方塊中的隱藏變數 (hidden variable)。

每個使用到的功能方塊，函數運作步驟會在每個執行週期（目標硬體週期）中被呼叫執行一次，用來處理輸入參數及更新相關資料。它代表此功能方塊的主要運算邏輯。

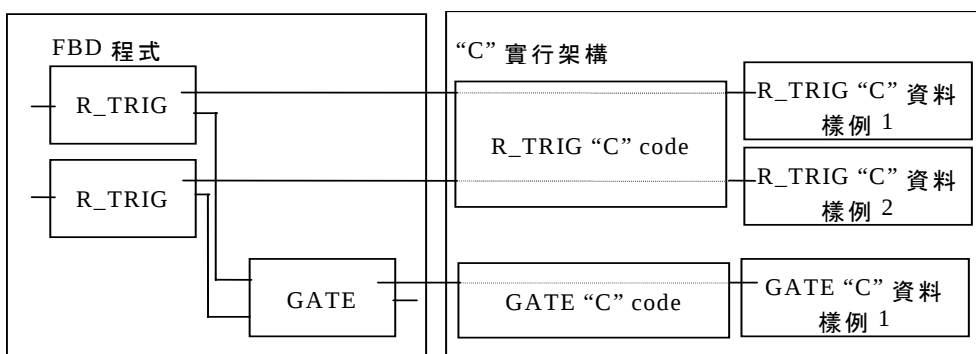
回傳參數讀值步驟會被 ISaGRAF 核心程式呼叫，用來讀取功能方塊中的一個回傳參數值。此部份不需要特別的計算，只負責隱藏資料與 ISaGRAF 應用程式間的傳遞工作。

功能流程圖：



功能方塊靜態資料

功能方塊包含了運算與靜態資料兩部份。同一種功能方塊的每次使用皆會伴隨著一筆資料結構，在 ST 或 FBD 程式中若用到功能方塊，每用一次便會對應到一個功能方塊元件及一筆資料結構。以下的例子說明了“C”資料結構，與 FBD 程式所使用的功能方塊元件間的對應關係：



當應用程式開始執行時，每個功能方塊元件的資料結構所需的記憶體，皆由 ISaGRAF 系統來取得。取得的記憶空間可透過指標傳給運作及讀值兩步驟來使用。

ISaGRAF 程式庫管理員會針對資料結構的型態定義，自動產生“C”原始碼。此資料結構的名稱永遠稱作“**str_data**”，使用者請勿修改，以確保其與標頭檔內容的相容性。隱藏資料中的內部變數一般都會對應著回傳參數來加以產生；功能方塊中讀值的步驟只用來取得回傳參數的值，不應在其中作任何的運算。

函數初始化步驟

功能方塊中初始化步驟於應用程式開始執行時會被 ISaGRAF 核心程式呼叫一次，用來向系統取得功能方塊元件所需的記憶區塊。以下為初始化步驟的標準程式：

```
uint16 FBINIT_xxx (uint16 hinstance)
/* “xxx” is the name of the f. block */
```

目標硬體使用者指南

```
{  
    return (sizeof (str_data));  
}
```

其中“**hinstance**”引數為此元件的邏輯編號，主要保留給 ISaGRAF 內部程式所使用，而初始化步驟的程式中不應用到此引數；這步驟會回傳“一個”元件所需要的記憶位元數目，數目（回傳值）不可超過 **64 K** 位元組，其他運算程式則不應寫在此步驟中。當製作一個功能方塊時，此初始化步驟的“C”原始碼會由 ISaGRAF 程式庫管理員自動產生。

函數運作步驟

應用程式裡每個使用到的功能方塊元件，其運作步驟的部份在每個程式週期都會被呼叫執行一次；此步驟會處理輸入的參數，並執行此功能方塊的主要運算邏輯，以更新隱藏資料與方塊輸出的參數。以下為運作步驟的標準程式：

```
void FBACT_xxx (  
    uint16 hinstance, /* “xxx”式功能方塊的名稱 */  
    /* 樣例的邏輯號碼 */  
    str_data * data, /* data: 樣例資料結構的指標 */  
    str_arg *arg /* 呼叫參數的指標 */  
)  
{  
}
```

其中“**hinstance**”引數為此元件的邏輯編號，主要保留給 ISaGRAF 內部程式所使用，而此步驟的程式中不應用到此引數；其中“**data**”引數是個遠程指標 (far pointer)，指向此功能方塊的資料結構。“**arg**”引數也是個遠程指標，指向輸入參數的結構中。程式師在撰寫程式時應使用此功能方塊的“C”標頭檔中所定義的名稱，來取得“**arg**”結構中的資料。

函數運作步驟中的演算邏輯會處理輸入的參數（存於“**arg**”結構中），並更新“**data**”結構中的資料。以下的例子說明了 **TRIG**（偵測上升邊緣）功能方塊中函數運作步驟的部份：

```

/* definitions stored in the function block “C” header */

typedef struct { /* calling parameters */
    T_BOO _clk; /* trigger input */
} str_arg;

#define CLK      (arg->_clk)

/* function block instance data structure */

typedef struct {
    T_BOO prev_state; /* previous state of the trigger input */
    T_BOO edge_detect; /* edge value: image of return param */
} str_data;

/* activation service */

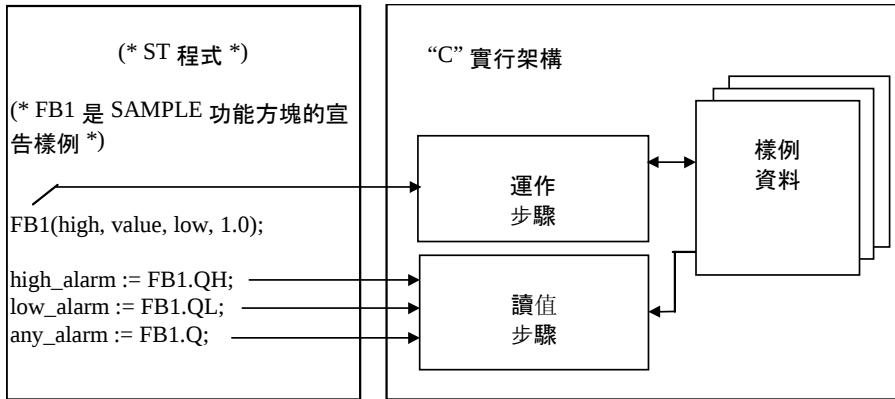
void FBACT_trig (uint16 hinstance, str_data *data, str_arg *arg)
{
    data->edge_detect = (T_BOO)(CLK && !data->prev_state);
    data->prev_state = CLK; /* calling parameter */
}

```

當製作一個功能方塊時，此初始化步驟的“C”原始碼會由 ISaGRAF 程式庫管理員自動產生。

讀取回傳的參數

在 ST 或 FBD 程式裡使用到的功能方塊，若其中回傳參數有執行到更新動作，則此讀值的步驟會被呼叫一次。這步驟是用來讀取一個回傳參數的值，以下的例子說明了當 ST 程式執行時此“讀值”步驟被執行的情況：



因為讀取回傳參數的步驟可能在一個程式週期中被呼叫很多次，因此對相同的回傳參數或相同的功能方塊元件而言，此步驟裡不應有特殊的計算或處理，其所負責的工作應只局限在隱藏資料與 ISaGRAF 應用程式間的資料傳遞上。以下為讀取回傳參數步驟的標準程式：

/ cast operation used to copy the value of a return parameter */*

```

#define BOO_VALUE ((T_BOO *)value)
#define ANA_VALUE ((T_ANA *)value)
#define REAL_VALUE ((T_REAL *)value)
#define TMR_VALUE ((T_TMR *)value)
#define MSG_VALUE ((T_MSG *)value)
  
```

/ return parameters read service: called for each return parameter */*

```

void FBREAD_xxx    /* “xxx” is the name of the function block */
uint16 hinstance, /* logical number of the instance */
str_data *data,   /* pointer to the instance data structure */
uint16 parno,     /* logical number of read parameter */
void *value) /* buffer where to copy the value of the param */
{
    switch (parno) {
  
```

```

case FBLPNO_XX: /* ... */ break;
case FBLPNO_YY: /* ... */ break;
/* .... */
}
}

```

其中“**hinstance**”引數為此元件的邏輯編號，主要保留給 ISaGRAF 內部程式所使用，而此步驟的程式中不應用到此引數；其中“**data**”引數是個遠程指標 (far pointer)，指向此功能方塊的資料結構。

當讀取某回傳參數時，“**parno**”引數可代表此回傳參數的邏輯編號。使用者撰寫程式可使用功能方塊“C”標頭檔中的名稱來區分回傳的參數，此名稱都以“**FBLPNO_**”作為開頭。“**value**”是個遠程指標，指向暫存區，此暫存區複製並儲存使用者要讀取的回傳參數數值，而此遠程指標的型態端賴 ISaGRAF 回傳參數的型態而定。以下的表格說明了 ISaGRAF 型態與此暫存區在“C”中型態的對應關係：

布林		32 位元不帶符號字組 (0=假 / 1=真)
類比		32 位元帶符號字組
實數		32 位元單精度浮點數值
計時器		32 位元不帶符號字組(單位是 1ms)

訊 息		字串陣列

以下的巨集命令 (macro) 根據欲存取回傳參數的型態，可用來對此複製暫存區作存取動作：

```
#define BOO_VALUE ((T_BOO *)value)
#define ANA_VALUE ((T_ANA *)value)
#define REAL_VALUE ((T_REAL *)value)
#define TMR_VALUE ((T_TMR *)value)
#define MSG_VALUE ((T_MSG *)value)
```

這些定義一般用於程式運作中，用來複製數值或參數到 ISaGRAF 緩衝區：

```
/* 對布林參數而言： */
*BOO_VALUE = parameter_value;
/* 對整數類比參數而言： */
*ANA_VALUE = parameter_value;
/* 對實數類比參數而言： */
*REAL_VALUE = parameter_value;
/* 對時間參數而言： */
*TMR_VALUE = parameter_value;
/* 對字串參數而言： */
strcpy (*MSG_VALUE, parameter_value);
```

當製作一個功能方塊時，此步驟的“C”原始碼會由 ISaGRAF 程式庫管理員自動產生。

“C”原始檔的範例：

以下是完整的 “C” 功能方塊標準程式：

```

/* 功能方塊 (xxx 式功能方塊的名稱) */

#include <asy0def.h>
#include <grfb0nnn.h>          /* nnn 程式庫中方塊的編號 */

/* 每一個方塊樣例的隱藏資料結構 */
typedef struct {
    /* fields definition */
} str_data;

/* 初始化步驟：傳回隱藏資料所需的記憶區間大小 */
word FBINIT_xxx (uint16 hinstance)
{
    return (sizeof (str_data));
}

/* 運作步驟：處理呼叫參數 */
void FBACT_xxx (uint16 hinstance, str_data *data, str_arg *arg)
{
    /* ... */
}

/* 投射運作：用來複製回傳參數的值 */
#define BOO_VALUE  ((T_BOO *)value)
#define ANA_VALUE  ((T_ANA *)value)
#define REAL_VALUE ((T_REAL *)value)
#define TMR_VALUE  ((T_TMR *)value)
#define MSG_VALUE  ((T_MSG *)value)

/* 回傳參數讀值步驟：呼叫每一個回傳參數 */

```

目標硬體使用者指南

```
void FBREAD_xxx (uint16 hinstance, str_data *data, uint16 parno, void
*value)
{
    switch(parno)
    {
        case FBLPNO_XX: *???_VALUE = ...; break;
        case FBLPNO_YY: *???_VALUE = ...; break;
        ....
    }
}
```

/ 下面的函數用來初始化功能方塊與宣告它的實行架構，它使用功能方塊的名稱來與 ISaGRAF 核心程式連結。這個步驟的程式完全由 ISaGRAF 程式庫管理員所產生。*/*

```
ABP fbldef_xxx (char *name, IBP *initproc, RBP *readproc)
{
    strcpy (name, "XXX");
    *initproc = (IBP)FBINIT_xxx;
    *readproc = (RBP)FBREAD_xxx;
    return ((ABP)FBACT_xxx);
}
```

/ 檔案結束 */*

其中“**TASY0DEF.H**”檔必須內含，以便載入跟系統有關的定義，此檔亦包含了一些遠程指標的資料型態定義，這些遠程指標會指向所完成的各程式步驟。

⇒ 案件與 C 程式間的連結

C 功能方塊與應用程式間的邏輯連結可利用此函數的名稱來完成。此功能方塊的 C 原始碼中會加入一段“宣告”，此段宣告在應用程式開始執行時會被

呼叫一次，用來通知 ISaGRAF 核心程式此 C 功能方塊的名稱及對應的程式碼。以下是此宣告服務的標準格式：

```
ABP fbldef_xxx (char *name, IBP *initproc, RBP *readproc)
{
    strcpy (name, "XXX");    /* 功能方塊名稱 */
    *initproc = (IBP)FBINIT_xxx;    /* 初始化步驟 */
    *readproc = (RBP)FBREAD_xxx;    /* 讀值步驟 */
    return ((ABP)FBACT_xxx);/* 運作步驟 */
}
/* xxx 是功能方塊的名稱 */
```

其中 **strcpy** 命令用到的功能方塊名稱必須用英文大寫來表示，小寫的部份則用於程式中及宣告服務的名稱裡。

使用 “**FBACT_**”、“**FBINIT_**”、“**FBREAD_**” 和 “**fbldef_**” 作為程式及名稱定義的字首，可幫助使用者以 C 語言的保留字或 ISaGRAF 程式庫中現存的函數來命名功能方塊。此宣告服務中無須再加入任何其他命令。

所有整合的 “C” 功能方塊皆會呼叫此 “宣告” 服務，即使在 ISaGRAF 應用程式中未被使用到。若應用程式使用一個未被整合的 C 功能方塊，則 ISaGRAF 核心程式會於執行時偵測出此重大錯誤。

在連結新功能方塊與核心程式之前，使用者尚必須完成另一個 C 程式檔，“**GRFB0LIB.C**”，並將其加入欲連結的檔案列表中。這個 “**GRFB0LIB.C**” 檔中只包含了一個宣告陣列，這陣列在應用程式初始化時被讀入，用來產生與 “C” 功能方塊間的動態連結。以下是此檔的範例：

```
/* 檔案: grfb0lib.c – 功能方塊的實行架構 */

#include <tasy0def.h>

extern ABP fbldef_fb1(char *name, IBP *init, RBP *read);
```

目標硬體使用者指南

```
extern ABP fbldf_fb2(char *name, IBP *init, RBP *read);
```

```
FBL_LIST FBLDEF[] = {
```

```
    Fbldf_fb1,
```

```
    Fbldf_fb2,
```

```
    NULL};
```

```
/* 檔案結束 */
```

其中 **FBLDEF** 陣列必須以空指標作為結束，否則可能造成當機。當連接新的 ISaGRAF 核心程式時，若此 **FBLDEF** 陣列未被定義，則會出現“參考資料未被定義 (unresolved references) 的錯誤。

藉由撰寫這個檔案可建立新的核心程式，並將會含括所有現存的功能方塊。當製作用於某種特定案件所需的程式時，亦可只將此案件所需的功能方塊填入此 **FBLDEF** 陣列中，來建立此專屬的核心程式。此“**GRFB0LIB.C**”檔會由 ISaGRAF 程式碼產生器 (Code Generator) 於應用程式的原始碼被製作時自動產生，置於 ISaGRAF 案件目錄下，並將此案件用到的功能方塊群聚起來。

限制

ISaGRAF 程式庫最多可含括 **255** 個“C”功能方塊，任何形態的運算皆可在一個函數中完成。一種功能方塊於一個相同的案件中最多可複製使用 **255** 次。

功能方塊是以同步的方式在每個 ISaGRAF 程式週期中被呼叫，因此功能方塊的執行速度對程式週期時間有直接的影響，使用功能方塊時應將此點謹記在心。

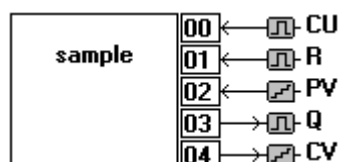
完整的範例

以下是一個功能方塊範例的完整程式，其功能為一上數計數器 (up-counter)。

以下是此功能方塊的技術註記：

name:	SAMPLE
descrip	上數計數器
tion:	
creatio	1994 年 2 月 1 號
n date:	
author:	ICS Triplex ISaGRAF Inc.
call:	CU：計數器輸入 R：歸零命令 PV：最大值
return:	Q：偵測是否發生最大值 CV：計數結果
prototy	SAMPLE (count, reset_command,
pe:	maximum_value); max_detect := SAMPLE.Q; count_result := SAMPLE.CV;

以下是此函數的界面：



以下是此函數的“C” 原始標頭檔 (source header)：

目標硬體使用者指南

/ 功能方塊介面 – 名稱: SAMPLE */*

/ 標準 ISaGRAF 資料型態定義 */*

```
typedef long T_BOO;  
typedef long T_ANA;  
typedef float T_REAL;  
typedef long T_TMR;  
typedef char *T_MSG;
```

/ 呼叫參數結構定義 */*

```
typedef struct {  
    T_BOO _cu;  
    T_BOO _r;  
    T_ANA _pv;  
} str_arg;
```

/ 用於存取呼叫參數的識別字 */*

```
#define CU (arg->_cu)  
#define R (arg->_r)  
#define PV (arg->_pv)
```

/ 回傳參數邏輯號碼 */*

```
#define FBLPNO_Q 0  
#define FBLPNO_CV 1
```

/ 檔案結束 */*

以下是此函數的“C”原始程式碼，其中只有以粗體字顯示的命令列是使用者必須填入的

```

/* 功能方塊 - 名稱: SAMPLE */

#include <tasy0def.h> /* 資料型態定義需要 */
#include <grfb0255.h> /* 功能方塊的 C 原始標頭檔 */

/* 包含一個樣例資料的結構定義 */

typedef struct {
    T_BOO overflow; /* 真: 計數值 >= 最大值 */
    T_ANA value; /* 目前的計數值 */
} str_data;

/* 初始化步驟: 樣例資料所需的記憶體大小 */

word FBINIT_sample (uint16 hinstance)
{
    return (sizeof(str_data));
}

/* 運作步驟: 上數計數邏輯運算 */

void FBACT_sample (uint16 hinstance, str_data *data, str_arg *arg)
{
    if (R) data->value = 0;
    else if (CU && data->value < PV) (data->value)++;
    data->overflow = (data->value >= PV) ? (T_BOO)1 : (T_BOO)0;
}

/* 投射運作: 用來複製參數至 ISaGRAF 緩衝區 */

#define BOO_VALUE ((T_BOO *)value)
#define ANA_VALUE ((T_ANA *)value)

```

```
#define REAL_VALUE ((T_REAL *)value)
#define TMR_VALUE ((T_TMR *)value)
#define MSG_VALUE ((T_MSG *)value)

/* 讀值步驟: 得到一個回傳參數 */

void FBREAD_sample (uint16 hinstance, str_data *data, uint16 parno,
void *value)
{
    switch (parno) {
        case FBLPNO_Q : *BOO_VALUE = data->overflow; break;
        case FBLPNO_CV : *ANA_VALUE = data->; break;
    }
}

/* 宣告服務: 用於與ISaGRAF 核心程式作動態連結 */

ABP fbldef_sample (char *name, IBP *initproc, RBP * readproc)
{
    strcpy (name, "SAMPLE");
    *initproc = (IBP)FBINIT_sample;
    *readproc = (RBP)FBREAD_sample;
    return((ABP)FBACT_sample);
}

/* 檔案結束 */
```

C.7.5 編譯與連結技術

ISaGRAF 工作平台並未包含任何的編譯及連結工具，然而此章節依然會講解一些主要技術，教導使用者如何操作 ISaGRAF 程式庫管理員所製作出來的檔案，並搭配其他的編譯及連結工具一起使用。

☐ “C” 原始檔

ISaGRAF 程式庫管理員會將轉換表、函數及功能方塊的“C”原始檔置於 **ISAWINLIB\DEFS** 及 **ISAWINLIB\SRC** 目錄下，其名稱會依據轉換表、函數或功能方塊在 ISaGRAF 程式庫中的數目順序而定。以下為使用到的檔案名稱：

<code>\isawin\lib\defs\TACN0</code>	所有轉換函數的定義檔
<code>DEF.H</code>	
<code>\isawin\lib\src\GRCN0n</code>	轉換函數的原始檔
<code>nn.H</code>	
<code>\isawin\lib\defs\CRUS0</code>	函數的定義檔
<code>nnn.C</code>	
<code>\isawin\lib\src\GRUS0n</code>	函數的原始檔
<code>nn.C</code>	
<code>\isawin\lib\defs\GRFB0</code>	功能方塊的定義檔
<code>nnn.H</code>	
<code>\isawin\lib\src\GRFB0n</code>	功能方塊的原始檔
<code>nn.C</code>	

(**nnn** 是轉換、函數或功能方塊的編號)

警告：當重新命名或複製程式庫中的元件時，ISaGRAF 程式庫管理員並不會根據新的名稱及邏輯編號將其對應的文字敘述及程式碼自動更新，此部份必須以人工方式自行處理。

在 **ISAWINLIB\USPNUMS** 檔中會定義 ISaGRAF 程式庫中的“C”函數名稱與邏輯編號間的關係，以下舉例此檔的內容：

```

1    funct_A
10   funct_B
     funct_C

```

目標硬體使用者指南

在 **ISAWIN\LIB\FBLNUMS** 檔中會定義 ISaGRAF 程式庫中的“C”功能方塊其名稱與邏輯編號間的關係，以下舉例此檔的內容：

```
0    fbl_A
1    fbl_B
2    fbl_C
```

在 **ISAWIN\LIB\CNVNUMS** 檔中會定義 ISaGRAF 程式庫中的轉換函數其名稱與邏輯編號間的關係，以下舉例此檔的內容：

```
0    SCALE
1    BCD
```

每當轉換表、函數或功能方塊被產生、更名、複製或消除時，這些檔皆會被 ISaGRAF 程式庫管理員自動地更新。而當一個建立一個應用時，ISaGRAF 程式碼產生器 (Code Generator) 則會自動地產生下列這些檔：

<code>\sawin\apl\ppp\GRCN0</code>	宣告案件所有用到的轉換函數
<code>LIB.C</code>	
<code>\sawin\apl\ppp\GRUS0</code>	宣告案件所有用到的函數
<code>LIB.C</code>	
<code>\sawin\apl\ppp\GRFB0</code>	宣告案件所有用到的功能方塊
<code>LIB.C</code>	

(**ppp** 是 ISaGRAF 案件的名稱)

當欲建立一個新的專屬用於某案件的 ISaGRAF 核心程式時，可以使用這些檔案來做連結，其中只需含括了此案件用到的轉換表、函數及功能方塊。

⇒ 將原始檔下載至原始系統中

若 ISaGRAF 目標系統支援基本的編譯工具，ISaGRAF 程式庫管理員所產生的“C”原始檔及相關定義檔案可下載至此系統中，此時可藉助 Windows 系統提供的標準終端機程式來達成。

若這些程式原始檔要在目標系統中重新整理編譯的話，每次以 ISaGRAF 程式庫管理員做任何函數介面的更動，都必須重新下載一次更新過的定義檔。

下載檔案的命令列可加以群聚整合至批次檔 (batch file) 中，並可從工作平台中的工具選單中加以執行 (參考使用者指南：程式管理)。

⇒ 使用跨平台編譯器

若目標硬體為 PC，或具備跨平台編譯器，可在同一台 PC 執行並產生目標系統的執行碼，則可直接在這台 PC 上編譯管理這些程式原始檔。

在這種情況下，使用者可以利用 ISaGRAF 程式庫管理員來完成轉換表、函數、或功能方塊原始檔的編修工作。

執行編譯及連結動作的命令列可加以群聚整合至批次檔 (batch file) 中，並可從工作平台中的工具選單中加以執行 (參考使用者指南：程式管理)。

當轉換表、函數、及功能方塊在 PC 上重新編譯過，使用者在執行應用程式之前，必須先將新產生的 ISaGRAF 核心程式 (與新元件重新連結過) 下載至目標系統中。若目標系統為另一台 PC，則新產生的 ISaGRAF 核心程式可藉由磁片或網路傳輸至目標硬體上。

⇒ 與 ISaGRAF 核心程式的程式庫做連結

警告：

以下的資料可能並不完全搭配你的目標系統。

任何情況你皆可查閱在目標系統磁片中的 readme 或 .TXT 檔。

ISaGRAF 目標系統磁片包含很多公程式檔，可用來將轉換表、函數、及功能方塊與 ISaGRAF 程式庫編譯連結在一起。

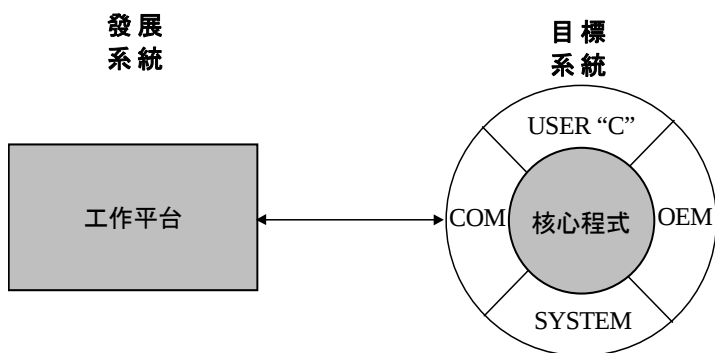
目標硬體使用者指南

有兩種執行模式：

單工 ISaGRAF：所有函數皆在同一個程式中執行

多工 ISaGRAF：一個獨立的程式（或程式緒 (thread)）專門負責通訊部份

在這兩種情況中，“C” 元件皆群聚於相同的程式庫中：對寫“C” 程式的人而言，單工與多工之間並沒有差別。在單工的版本中，使用者的“C” 程式庫會與單工程式連結在一起（通常叫做 **isa**），而在多工版本中，程式庫會與核心程式連結在一起（通常叫做 **isaker**）。



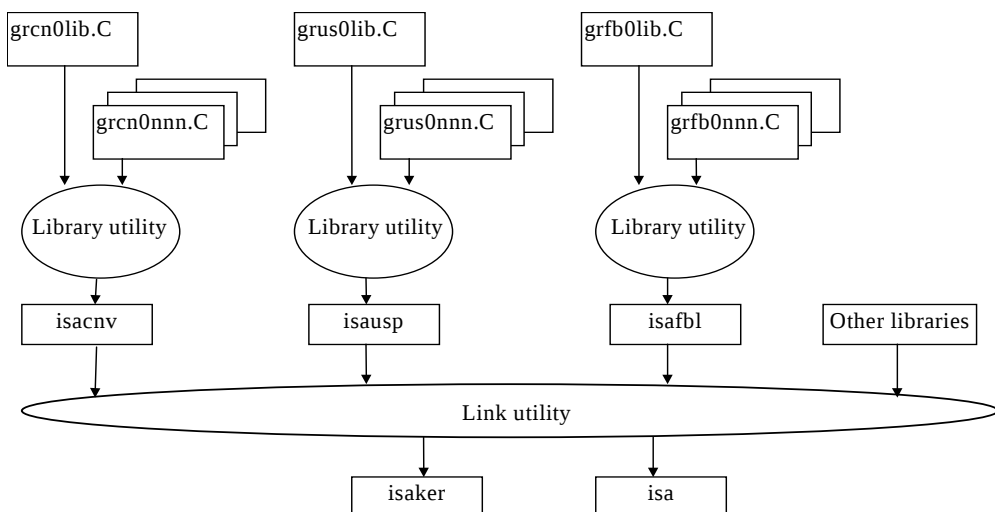
ISaGRAF 軟體的內部核心與硬體無關，可執行 IEC 語言並擁有自己的變數資料。

要與核心程式做連結時，第一步為建立案件用的所有轉換表、函數及功能方塊的程式庫：

程 式 庫	內容
ISA US P	- GRUS0LIB 目的碼檔 (函數宣告矩陣) - 每一個整合的函數物件檔
ISA FBL	- GRFB0LIB 目的碼檔 (功能方塊宣告矩陣) - 每一個整合的功能方塊物件檔

ISA	- GRCN0LIB 目的碼檔 (轉換宣告矩陣)
CN	- 每一個整合的轉換函數物件檔
V	

接著必須將這些新的程式庫與其他目的碼檔 (object files)，及 ISaGRAF 核心程式的程式庫連結在一起。以下為使用者 “C” 程式的發展整合架構圖：



以下為程式連結時必須加入的目的模組 (object modules)，以及程式庫的完整列表：

建立 **isaker**：

目的模組：**tast0mai**

目的模組：**tats0com**

核心程式庫：**isaker**

核心程式庫：**isaoem**

使用者自訂程式庫：**isausp** 使用者定義函數

使用者自訂程式庫：**isafbl** 使用者定義功能方塊

目標硬體使用者指南

使用者自訂程式庫：**isacnv** 使用者定義轉換函數

核心程式庫：**isasy**

系統程式庫：（參考你的“C”編譯器手冊）

建立 isa：

目的模組：**tast0mai**

目的模組：**tast0com**

核心程式庫：**isaker**

核心程式庫：**isatst**

核心程式庫：**isaoem**

使用者自訂程式庫：**isausp** 使用者定義函數

使用者自訂程式庫：**isafbl** 使用者定義功能方塊

使用者自訂程式庫：**isacnv** 使用者定義轉換函數

核心程式庫：**isasy**

系統程式庫：（參考你的“C”編譯器手冊）

程式員最好遵循上表中目的模組與程式庫的順序來做，其名稱皆會根據不同的目標系統來定義延伸檔名（“.lib”，“.obj”，“.l”，“.r”...）。

□ 編譯與連結時所需的選項

在編譯與連結程式時可使用方便的參數選項，端視於對轉換表、函數及功能方塊所需的處理型態而定，在連結時有些處理會需要其他的系統程式庫（math、graphics...）。

所有 ISaGRAF 核心程式的 “C” 原始檔皆以 LARGE 記憶模式編譯過，因此編譯轉換表、函數及功能方塊時皆需以此模式來進行。

在編譯 “C” 程式庫元件時，有一個特別的常數必須加以定義。這常數指出了目標系統及處理器的類型，如此便可將轉換表、函數及功能方塊原始檔完全改為與系統無關的形式。以下是這些常數的名稱：

DOS DOS 系統 (INTEL 處理器)

ISAWNT Windows-NT 系統 (INTEL 處理器)

OS9 OS9 系統 (MOTOROLA 處理器)

VxWorks VxWorks 系統 (MOTOROLA 處理器)

在 ISaGRAF 目標系統軟體所提供的公用程式檔中，描述了如何在編譯器命令中，加入這些常數值的定義。

□ 已支援的編譯器

以下的編譯器已被支援，可用於轉換、函數及功能方塊的發展，及與 ISaGRAF 核心程式的連結：

Microsoft MSC 7.00 compiler	用於 DOS based 目標硬體
Microsoft MSVC 4.00 compiler	用於 Windows-NT based 目標硬體
Microware ULTRA-C compiler	用於 OS-9 目標硬體
Tornado 1.0; GNU Toolkit 2.6	用於 VxWorks 目標硬體

若需使用其他的編譯器，可與 ICS Triplex ISaGRAF Inc. 聯絡。

□ 總結

以下是發展新的轉換、函數及功能方塊時所需的流程總結：

- ⇒1. 使用 ISaGRAF 程式庫管理員，製作新的元件：給定名稱與註解文字。
“C” 原始檔會自動由程式庫管理員產生。

目標硬體使用者指南

- ⇒2. 若此元件為函數或功能方塊，則使用 ISaGRAF 程式庫管理員，定義所需介面 (呼叫與回傳參數)。“C” 原始標頭檔會自動由程式庫管理員產生。
- ⇒3. 使用 ISaGRAF 程式庫管理員，輸入此元件的詳細技術註記 (使用者指南)。
- ⇒4. 使用 ISaGRAF 程式庫管理員，完成“C” 原始檔，輸入轉換、函數及功能方塊所需的程式碼，至此完成原始程式的編輯工作。注意此時應會使用到其他的文字編輯工具。
- ⇒5. 在程式庫管理員中，選取“顯示邏輯號碼”選項，可得知此新元件的邏輯編號。此編號會被用來命名“.C”及“.H”原始檔的路徑名稱。
- ⇒6. 將“.C”及“.H”原始檔複製 / 下載至目標系統中 (若系統使用自己的編譯器)，或至其他相對的環境 (若使用跨平台編譯器)，此目標系統必須已植入 ISaGRAF 程式庫及相關程式。
- ⇒7. 執行“C”編譯器編譯新的原始檔，並更正語法錯誤。
- ⇒8. 在定義了元件名稱陣列的“GR??0LIB.C”檔中，加入新元件宣告服務的名稱。
- ⇒9. 執行“C”編譯器編譯“GR??0LIB.C”檔。
- ⇒10. 將目的碼模組 (object module) 的名稱加入目的碼檔案的列表中，用來製作相對的程式庫。
- ⇒11. 執行“C”程式庫連結程式，建立新的 ISaGRAF 核心程式。
- ⇒12. 將新建好的核心程式植入目標硬體上。
- ⇒13. 寫一個 ISaGRAF 應用程式，測試新元件的執行效果與其介面。

C.8 Modbus 連結

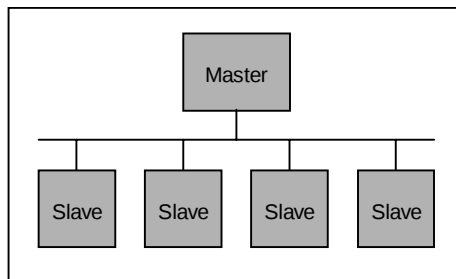
應用程式發展測試完成後，你可將目標系統與程序監控軟體做通訊連結。

ISaGRAF 為開放式的架構，具備各種通訊方法的可能性。一種最簡單的工業通訊為 MODBUS/MODICON 通訊協定標準，幾乎所有的程序監控軟體都支援這種通訊協定，並且只需串列連結即可 (RS232、RS485、Current Loop)。

ISaGRAF 除測程式其通訊協定亦與 MODBUS 相容，可由 MODBUS master 讀寫目標系統上的程式變數數值。

C.8.1 MODBUS 網路與協定

MODBUS 網路是由一個 master (通常為程序監控系統) 及一或多個 slave 站 (通常為 PLCs) 組成。



Master 站一次送出一個詢問命令給一個 slave 站 (使用 slave 編號)，並在下次送出其他命令之前等待其回應，至於其他無關的 slave 站則不會回應任何訊息。

每一個封包皆內含一個 slave 編號、一個詢問命令編號與相對資料，及一個 16 位元的檢查碼 (CRC checksum)。

目標硬體使用者指南

若經過一段通訊失敗 (time-out) 等待時間，slave 站還未回答，則詢問命令還會被重複送出數次，之後 master 站便會宣告此 slave 站“斷線”。至於通訊失敗時間與重複送出的次數則可在 master 站中加以調整，以搭配 slave 站的執行狀況（視應用程式或其他狀況等等）。

在詢問的過程中若有錯誤發生，則 slave 站不會回傳對應的資料封包，而會送出一個錯誤訊息給 master 站。

MODBUS 是 Modicon 制訂的通訊協定，並不是國際標準，因此市面上有許多與 MODBUS 相容的協定存在，其間可能有需多差異，例如：

功能碼 (function codes) 的差異

位址的對應

RTU(二元碼)格式或 ASCII 格式

等等...

C.8.2 ISaGRAF 上的通訊協定

⇒ 存取應用程式中的變數

在 ISaGRAF 的通訊中可辨認下列五種 Modbus 功能碼：

1	read N bits (讀取 N 個位元)
3	read N words (讀取 N 個字組)
5	write 1 bit (寫入 1 個位元)
6	write 1 word (寫入一個字組)
16	write N words (寫入 N 個字組)

ISaGRAF 應用程式中的變數資料可經由‘網路位址’的定義來辨認取得，當然這些位址必須先在工作平台中的字典中定義過。這些變數可為：

布林或類比變數

輸入、輸出或內部變數

區域或全域變數

要寫入一個布林變數，可使用功能碼 5、6 或 16。任何非 0 值皆被視為真 (TRUE)。

要讀取一個布林變數，可使用功能碼 1 或 3。用功能碼 1，數值以位元方式回傳。用功能碼 3，數值以位元組方式回傳 (真 (TRUE) 以 0xFFFF 表示)。

要寫入一個類比變數，可使用功能碼 6 或 16。傳回數值為介於-32768 與 +32767 之間的 16 位元整數。

要讀取一個類比變數，可使用功能碼 3。傳回數值亦為介於-32768 與+32767 之間的 16 位元整數。在目標系統上，類比變數為 32 位元，因此在目標系統上超過 16 位元範圍的數值會以 16 位元最大的範圍值讀回 (正值或負值)。

實數變數無法以 Modbus 協定存取。

警告：

ISaGRAF 的 MODBUS 通訊程式不處理錯誤碼，例如像‘無法認知的 modbus 位址’。

縮寫列表：

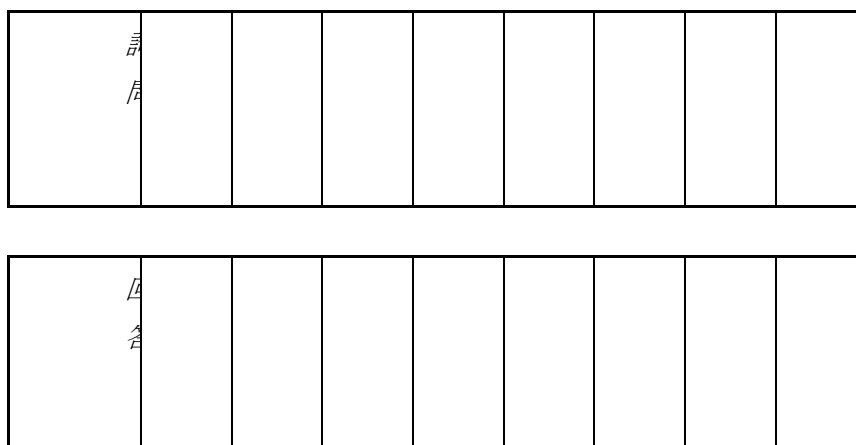
s	slave 編號 (slave number)
l	
v	
n	字組數目 (number of words)
b	
w	

目標硬體使用者指南

<i>n</i>	位元組數目 (number of bytes)
<i>b</i>	
<i>b</i>	
<i>n</i>	位元數目 (number of bits)
<i>b</i>	
<i>i</i>	
<i>a</i>	網路位址 (network address) (高位元組, High Byte)
<i>d</i>	
<i>d</i>	
<i>H</i>	
<i>a</i>	網路位址 (network address) (低位元組, Low Byte)
<i>d</i>	
<i>d</i>	
<i>L</i>	
<i>v</i>	數值 (value) (高位元組, High Byte)
<i>H</i>	
<i>v</i>	數值 (value) (低位元組, Low Byte)
<i>L</i>	
<i>V</i>	位元組數值 (Byte value)
<i>b</i>	位元欄 (bit filed) (nbb Bytes)
<i>f</i>	
<i>d</i>	
<i>c</i>	檢查碼 (checksum) (高位元組, High Byte)
<i>r</i>	
<i>c</i>	
<i>H</i>	
<i>c</i>	檢查碼 (checksum) (低位元組, Low Byte)
<i>r</i>	
<i>c</i>	
<i>L</i>	

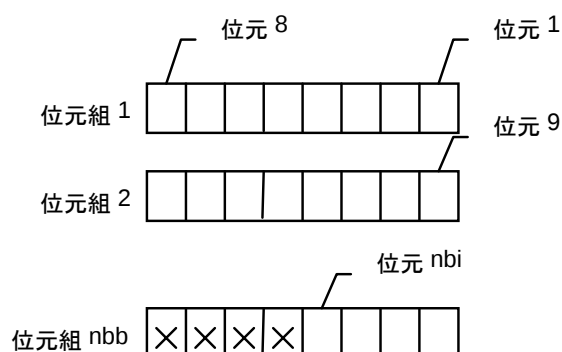
功能碼 1：讀 *N* 個位元

從網路位址 addH/addL 開始，讀取 nbi 個位元 (布林) 數值



位元組 1 位元組 nbb

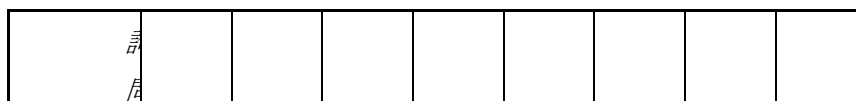
bfd 是 nbb 個位元組中的一個位元欄 (bit field)，其格式如下：



位元 1 對應到網路位址為 addH/addL 的變數值，位元 nbi 對應到網路位址為 $\text{addH}/\text{addL} + \text{nbi} - 1$ 的變數值。X 代表未定義的數值。

功能碼 3：讀 N 個字組

從網路位址 addH/addL 開始，讀取 nbw 個字組數值



目標硬體使用者指南

--	--	--	--	--	--	--	--	--

位元組 地址								
-----------	--	--	--	--	--	--	--	--

nbb 對應到 *vH* , *vL* 位元組的數目。

功能碼 5：寫 1 個位元

在網路位址 *addH/addL*，寫入一個位元 (布林) 數值

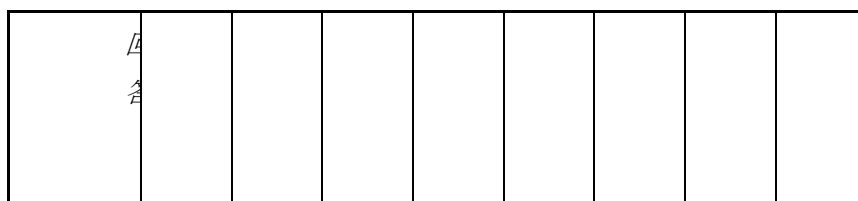
位元組 地址								
-----------	--	--	--	--	--	--	--	--

位元組 地址								
-----------	--	--	--	--	--	--	--	--

功能碼 6：寫 1 個字組

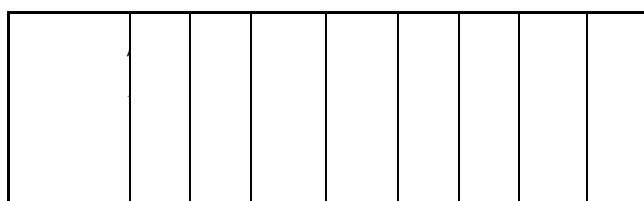
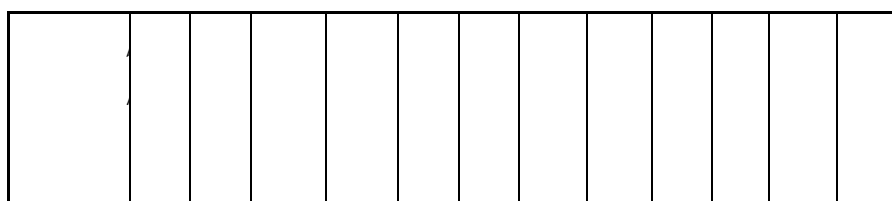
在網路位址 *addH/addL*，寫入一個字組數值

位元組 地址								
-----------	--	--	--	--	--	--	--	--



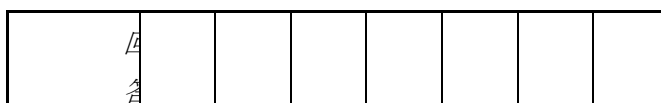
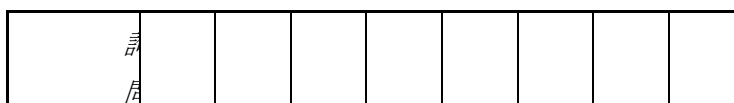
功能碼 16：寫 N 個字組

從網路位址 addH/addL 開始，寫入 nbw 個字組數值 (nbd=2nbw)



範例：

- 功能碼 1：從 slave1，網路位址 0x1020 開始，讀取 15 個位元



讀回數值為 0x0012，相對於二位元為 00000000 00010010。在這個例子中，只有變數位址為 0x1029 及 0x102C 的兩個變數其值為真，其他皆為假。

- 功能碼 16：從 slave1，網路位址 0x2100 開始，寫入 2 個字組數值，分別為 0x1234 及 0x5678。

16														
----	--	--	--	--	--	--	--	--	--	--	--	--	--	--

16								
----	--	--	--	--	--	--	--	--

☐ 檔案傳輸

與其他更先進的場域通訊 (field bus) 比起來，Modbus 協定所能提供的功能非常貧瘠，尤其當硬體製造廠商沒有新增任何其他功能碼的時候。

在這種情況下，在一個功能強大、可變性高的電腦上執行 ISaGRAF，對於 Modbus 協定而言，將會有兩種限制：

其只具備存取 ISaGRAF 變數值的能力
很難快速傳遞大量的資料

這就是為何 ISaGRAF 在 Modbus 功能碼中提供檔案傳遞的功能，或者可稱之為‘遠端檔案管理’協定。這種能力可讓系統執行：

下載二進位 (binary) 或文字 (ASCII) 格式的檔案
上載二進位 (binary) 或文字 (ASCII) 格式的檔案
透過虛擬或實際的共享檔案作動態資料交換

如此，透過 ISaGRAF 的 Modbus 通訊，任何「與 ISaGRAF 無關」的應用皆可很容易地與遠端的目標系統通訊。

這種協定遵循下列三種概念：

在 ISaGRAF 目標系統上的檔案稱為**遠端檔** (remote file)

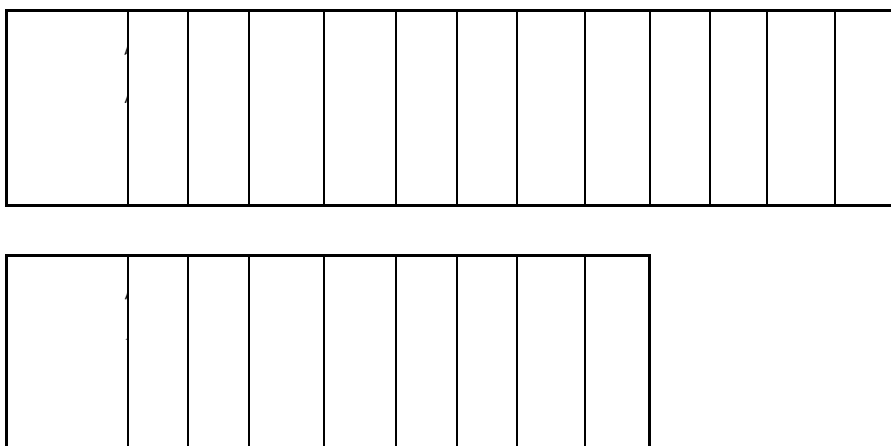
在 ISaGRAFmaster 電腦上的檔案稱為**本端檔** (local file)

檔案中的每一個位元組皆可藉由32 位元基礎位址 (**base address**) 與16 位元組位址 (**byte address**) 的定址方式來存取

有對應的詢問命令來選擇遠端檔的名稱，選擇基礎位址及使用 16 位元組位址來讀取遠端檔的資料。

功能碼 17：寫入資料

nbb 對應到 vH，vL 位元組的數目



詢問命令所代表的意義隨著位址範圍 addH/addL 的變動而不同：

0xF000：將遠端檔的名稱初始化

nbb 對應到檔案名稱的字元數目，也就是 vH，vL 欄位的數目（在這種情況下每個 vH 或 vL 數值皆代表一個字元，High 及 Low 不具任何意義），並包含 10 作為字串結尾字元。若這個檔案不存在，則會以 可寫 (writable) + 可讀 (readable) + 可執行 (executable) 的屬性加以產生。

0xF002：更改基礎位址 (base address) 的數值

nbb 此時應等於 4。第一個 vH/vL 位元組(byte)對應到基礎位址數值中的高字組 (High word) 部份。任何的 32 位元數值皆會被接受。

之後的所有讀寫命令皆會以此基礎位址為準。此基礎位址在不使用時的預設值為 0。

目標硬體使用者指南

0xF004：刪除檔案

nbb 此時應等於 0。

此檔案若存在且允許被刪除的話，則會被刪除。

大於 0xF004：保留

小於 0xF000：寫入位元組

要寫入位元組的相對位址存於 addH/addL 處，其值必須小於 F000。被寫入的數值（共 nbb 個位元組，其值存於 vH vL 欄中，此時 High 及 Low 不具任何意義）以固定的順序（從左到右）寫入之前給定名稱的遠端檔案中。要寫入數值的位址，為之前選定的基礎位址加上相對位址。若此位址超過檔案大小，則此檔繼續延伸。你不能減少檔案的大小。

功能碼 18：讀取資料

	讀 入							
	位 元							

	位 元							
	組							

要讀取位元組的相對位址存於 addH/addL 處，其值必須小於 F000。此功能碼會從之前給定名稱的遠端檔案中讀取給定數量 (nbb) 的位元組，起始位址為之前選定的基礎位址加上相對位址 (addH/addL 中任何 16 位元的數值)。傳回資料（從左到右存於 V 欄中）的順序會依循從檔案中讀出的順序。

範例：

選取遠端檔案的名稱：‘target.fil’。

讀												
寫												

讀								
寫								

選取基礎位址：0x10000

讀												
寫												

讀								
寫								

寫入 4 個位元組：絕對位址為 0x107D0，數值為 01，02，03，04。

讀												
寫												

讀								
寫								

讀取 4 個位元組：絕對位址為 0x107D0。

讀								
寫								

讀								
寫								

C.9 電源中斷管理

C.9.1 基本觀念

電源中斷的處理常常隨著應用程式而不同，有三個原因：

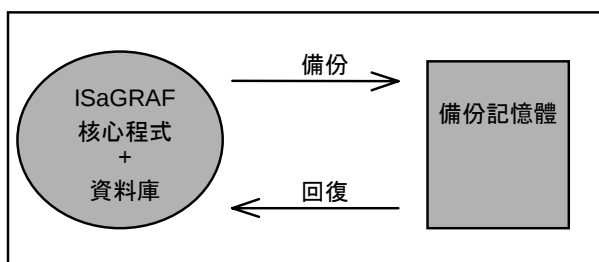
其視程序的特性而定

其視硬體的能力而定

其視程式的方法而定

因此，ISaGRAF 對電源中斷的處理方式不是一個完整且絕對廣泛的方法，而是針對不同應用或硬體，將一些條列、方法與工具以一種特定的方法加以組合使用。

要讓一個流程控制系統在斷電後能正確地重新啓始，有 3 個問題必須解決：



將資料備份

於啓始時能偵測到之前發生的斷電情況

重新載入備份的資料

其中的第二項問題，並沒有一個標準的軟體解決方案，一般都依賴系統供應者提供一些必要的工具，能夠從 ISaGRAF 應用程式中或 C 程式中讀取硬體的狀態來解決。

再者，一件更重要的事是決定何種資料必須加以備份。這裡我們定義兩種資料：

應用程式的變數：

例如程序變數，像處理過的項目數目、斷電的日期、應用程式的參數值等。

例如程式中的變數，像計數器 (counter)、計時器 (timer)、中介變數值 (intermediate values) 及旗標 (flag) 等。

程式狀態：

例如執行到的步數、每一個 C 程式的狀態等。

這兩種情況將在以下的章節中討論到，看看 ISaGRAF 能夠提供何種解答。

C.9.2 應用程式的變數備份

⇒ 可保留變數

在 ISaGRAF 工作平台中的變數編輯器裡，針對內部變數提供了‘可保留’的選項 (非 IO 變數)。

在每次目標程式的週期結數時，可保留變數的值會被複製到特定的記憶位置，此記憶位置通常為以電池為備份電源的 RAM。

在一開始，如果有一個以上的變數具備‘可保留’屬性，ISaGRAF 會尋找這些記憶變數：

如果相同的應用程式之前執行過，則 ISaGRAF 會辨認出這些儲存值，並將之存入每一個‘可保留’變數中。

如果之前執行不同的應用，或者之前沒有其他的應用程式，則 ISaGRAF 會辨認出這些記憶的數值並不正確，然後將之全部清除為 null。

用來儲存不同型態可保留變數的記憶空間，其規格可由工作平台中加以設定，在製作功能表：**執行期選項；可保留變數**。

指定的字串必須遵循下列的格式：

目標硬體使用者指南

`boo_add,boo_size,ana_add,ana_size,tmr_add,tmr_size,msg_add,msg_size`

其中：

boo_add：用來儲存布林變數的十六進位位址。其值不可為 0。

boo_size：在此位址下可使用的位元組容量大小，為十六進位值。一個布林變數使用一個位元組。

ana_add：用來儲存類比變數的十六進位位址。其值不可為 0。

ana_size：在此位址下可使用的位元組容量大小，為十六進位值。一個類比變數使用四個位元組，此值最少為四個位元組。

tmr_add：用來儲存計時器變數的十六進位位址。其值不可為 0。

tmr_size：在此位址下可使用的位元組容量大小，為十六進位值。一個計時器變數使用五個位元組。

msg_add：用來儲存訊息變數的十六進位位址。其值不可為 0。

msg_size：在此位址下可使用的位元組容量大小，為十六進位值。一個訊息變數使用 256 個位元組。

需求：

每一種型態的每一個欄位皆需加以指定，即使你不需將所有型態的變數備份。在此種情況，你必須將不需要保留的變數其大小指定為零（除了類比變數你必須將大小指定為四個位元組），並任意指定不為零的位址值。

範例：

假設我們必須備份以下資料：

20 個布林變數

0 個類比變數

0 個計時器變數

3 個訊息變數

資料備份的位址在 0xA2F200。

我們假設：

布林變數會從位址 0xA2F200 開始儲存，大小為 20 個位元組。

類比變數最少需要 4 個位元組，從位址 0xA2F214 開始儲存。

計時器變數假定的儲存位址為 0xA2F200，大小為 0 個位元組。

訊息變數會從位址 0xA2F218 開始儲存，大小為 256*3 個位元組。

如此輸入的字串應為：

A2F200,14,A2F214,4,A2F200,0,A2F218,300

二 系統函數呼叫

t 如果應用程式裡大多數的變數皆需要儲存，則應使用系統函數來處理一整組的變數（系統函數更詳細的說明請參閱使用者指南）。此處提醒一點，資料的備份與回復是由程式設計者在應用層次上來加以管理。

首先你必須定義一種特定型態或所有型態變數的備份位置：

```
<new_address> := SYSTEM(SYS_INITxxx,<address>);
```

其中：

<address> 為備份資料的儲存位址（16# 表示十六進位的格式），必須為偶數數值，否則此命令失效。

SYS_INITxxx 可為：

*SYS_INITBOO 用來定義所有布林變數的儲存位址。

*SYS_INITANA 用來定義所有類比變數的儲存位址。

*SYS_INITTMR 用來定義所有計時器變數的儲存位址。

*SYS_INITALL 用來定義所有布林、類比及計時器變數的儲存位址。

<new_address> 會取得下一個空白的位址，也就是根據 SYS_INITxxx 來取得<address>+ 備份變數所需的容量大小（位元組）。此函數可用來檢查備份所需的記憶體大小，若此函數發生錯誤，則 <new_address> 取得的值將為 0。

接著你便可使用備份功能。這程序可在應用程式中的任何地方加以呼叫，備份的動作會在目前這個程式週期的最後被執行一次。在系統斷電的時候，若

目標硬體使用者指南

硬體可送出一個布林輸入訊號，或有 C 函數可通知使用者此情況的發生，並且有足夠的延遲時間，可讓 ISaGRAF 在斷電關機前多執行一個以上的程式週期，則資料備份的工作在偵測到斷電時可由下式完成：

```
<error> := SYSTEM(SYS_SAVxxx,0);
```

其中：

SYS_SAVxxx 可為：

*SYS_SAVBOO 來要求所有布林資料的備份

*SYS_SAVANA 來要求所有類比資料的備份

*SYS_SAVTMR 來要求所有時間資料的備份

*SYS_SAVALL 來要求所有布林、類比及時間資料的備份

<error>會於操作失敗時回傳不為0的數值 (SYS_INITxxx 未被呼叫執行過)。

最後，也許你想要重新載入備份的資料。這程序可在應用程式中的任何地方加以呼叫，重新載入的動作會在目前這個程式週期的最後被執行一次。為了確保備份資料的正確，類比變數必須設成已定義好的定值：

```
<error>:=SYSTEM(SYS_RESTxxx,0);
```

其中：

SYS_RESTxxx 可為：

*SYS_RESTBOO 重新載入所有布林資料的備份

*SYS_RESTANA 重新載入所有類比資料的備份

*SYS_RESTTMR 重新載入所有時間資料的備份

*SYS_RESTALL 重新載入所有布林、類比及時間資料的備份

<error>會於操作失敗時回傳不為0的數值 (SYS_INITxxx 未被呼叫執行過)。

以下是系統函數的命令總表，用來處理備份的變數：

命令		意義
預定的關鍵字	值	
SYS_INITBOO	16#20	初始布林備份

SYS_SAVBOO	16#21	儲存布林變數
SYS_RESTBOO	16#22	回復布林變數
SYS_INITANA	16#24	初始類比備份
SYS_SAVANA	16#25	儲存類比變數
SYS_RESTANA	16#26	回復類比變數
SYS_INITTMR	16#28	初始時間備份
SYS_SAVTMR	16#29	儲存計時器變數
SYS_RESTTMR	16#2A	回復計時器變數
SYS_INITALL	16#2C	初始所有型態備份
SYS_SAVALL	16#2D	儲存所有型態變數
SYS_RESTALL	16#2E	回復所有型態變數

命令 (預定的關鍵字)	參數	回傳值
SYS_INITxxx	記憶體位址	下一個自由位址
SYS_SAVxxx	0	若可行則回傳 0
SYS_RESTxxx	0	若可行則回傳 0

☞ 製作指定規格的應用

最後，你應可以使用 C 函數或功能方塊來發展特定的程序，存取以電池備份的記憶體中資料，並可在應用程式中的任何時刻存取變數。

範例：

針對某個特定應用的備份程序：

`backup , restore_temp , restore_date , restore_cpt` 為一些使用者的 C 程序。

backup(temperature, date, cnt); 儲存 3 筆臨界資料

目標硬體使用者指南

`temperature := restore_temp();` 回復 temperature

`date := restore_date();` 回復 date

`cnt := restore_cnt();` 回復 counter

針對一般需求的備份程序：

`backup_init`，`backup`，`backup_link`，`restore` 為一些使用者的 C 程序。

`save_id := backup_init(address, size);` 取得一塊備份所需的記憶區塊

`backup(save_id, cpt1, 3);` 將 `cpt1` 儲存成第 3 個備份元件

`rest_id := backup_link(address, size);` 連結至備份資料的記憶區塊

`cpt1 := restore(rest_id, 3);` 重新載入 `cpt1` 的備份數值

C.9.3 程式狀態備份

儲存每個應用程式的每個執行狀態並不困難，不過重新載入程式之前的執行狀態似乎有些危險，至少有下列 3 個原因：

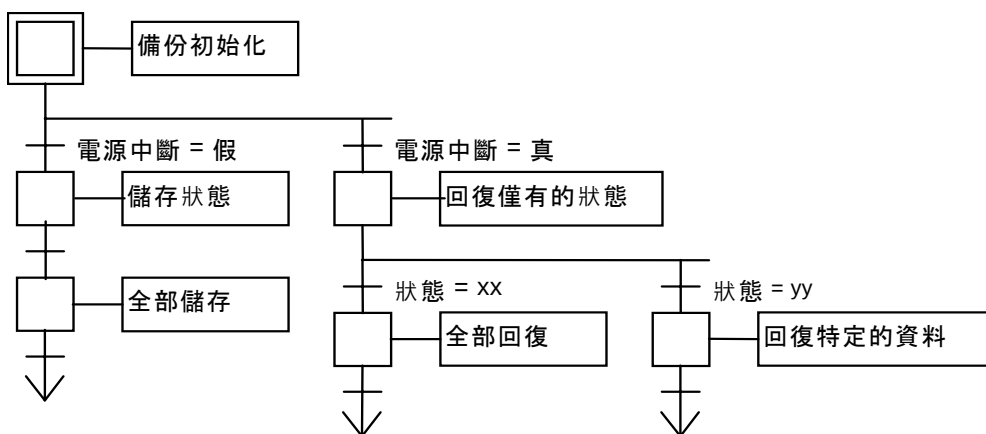
某些程序在重新啓始前必須先執行一些特定的操作

處理一整個應用的每一個狀態非常的冗長繁雜

一些外部的資源，像 C 程式、電腦周邊裝置等，無法自動重新啓始。

最好的方法似乎是將程序的狀態以布林或類比變數備份起來，當寫程式的人在思考重新啓始的階段所需處理的事時，便可以使用到這些備份的資料。而程式重新啓始時，便可以使用這些不全然完整但卻聰明的對應狀態資料，來停止或暫停 SFC 程式，將變數初始化，使應用程式跳到一個適當的執行狀態。ISaGRAF 並不提供自動的備份啓始程序。

範例：



C.10 附錄：錯誤碼列表與描述

錯誤碼列表：

訊息	型態
無法配置執行期資料庫存取所需的記憶體 (cannot allocate memory for run time data base)	系統
不正確的應用程式資料 (Motorola/Intel) (incorrect application data base)	應用程式
無法配置通訊郵包 (cannot allocate communication mailbox)	系統
無法連結核心程式資料庫 (cannot link kernel data base)	系統
傳送通訊詢問至核心程式逾時 (time-out sending question to kernel)	系統
等待核心程式回答逾時 (time-out waiting answer from kernel)	系統
無法將通訊初始化 (cannot init communication)	系統
無法配置記憶變數所需的記憶體 (cannot allocate memory for retained variables)	應用程式
應用程式終止 (application stopped)	應用程式
太多同時的 N 或 P 行為	應

	(too many simultaneous N or P actions)	用 程 式
	太多同時的設定行為 (too many simultaneous setting actions)	應 用 程 式
	太多同時的重設行為 (too many simultaneous resetting actions)	應 用 程 式
	未知的 TIC 指令 (unknow TIC instruction)	應 用 程 式
	無法回答讀值詢問命令 (cannot answer read data request)	系 統
	無法回答寫值詢問命令 (cannot answer write data request)	系 統
	無法回答除錯程式溝通詢問命令 (cannot answer debugger session request)	系 統
	無法回答 modbus 詢問命令 (cannot answer modbus request)	系 統
	無法回答除錯程式應用詢問命令 (cannot answer debugger application request)	系 統
	無法回答除錯程式 (cannot answer debugger)	系 統
	未知的詢問命令碼 (unknown request code)	系 統
	乙太網路通訊錯誤 (Ethernet communication error)	系 統
	通訊同步錯誤	系

目標硬體使用者指南

	(communication synchro error)	統
	無法配置應用程式所需記憶體 (cannot allocate memory for application)	系 統
	無法配置應用程式更新所需記憶體 (cannot allocate memory for application update)	系 統
	未知的 OEM key 碼 (unknown OEM key code)	應 用 程 式
	無法初始化布林輸入 IO 板 (cannot init boolean input board)	應 用 程 式
	無法初始化類比輸入 IO 板 (cannot init analog input board)	應 用 程 式
	無法初始化訊息輸入 IO 板 (cannot init message input board)	應 用 程 式
	無法初始化布林輸出 IO 板 (cannot init boolean output board)	應 用 程 式
	無法初始化類比輸出 IO 板 (cannot init analog output board)	應 用 程 式
	無法初始化訊息輸出 IO 板 (cannot init message output board)	應 用 程

		式
	無法輸入布林 IO 板 (cannot input boolean board)	應 用 程 式
	無法輸入類比 IO 板 (cannot input analog board)	應 用 程 式
	無法輸入訊息 IO 板 (cannot input message board)	應 用 程 式
	無法輸出布林輸出變數 (cannot output boolean output variable)	應 用 程 式
	無法輸出類比輸出變數 (cannot output analog output variable)	應 用 程 式
	無法輸出訊息輸出變數 (cannot output message output variable)	應 用 程 式
	無法操作布林變數 (cannot operate boolean variable)	應 用 程 式
	無法操作類比變數 (cannot operate analog variable)	應 用 程

目標硬體使用者指南

		式
	無法操作訊息變數 (cannot operate message variable)	應 用 程 式
	無法開啓IO 板 (cannot open board)	應 用 程 式
	無法關閉IO 板 (cannot close board)	應 用 程 式
	無法覆寫布林輸出變數 (cannot overwrite boolean output variable)	程 式
	無法覆寫類比輸出變數 (cannot overwrite analog output variable)	程 式
	無法覆寫訊息輸出變數 (cannot overwrite message output variable)	程 式
	未知的系統詢問命令碼 (unknown system request code)	程 式
	取樣週期溢位 (sampling period overflow)	程 式
	使用者函數未完成 (user function not implemented)	應 用 程 式
	整數除以零 (integer divided by zero)	程 式
	轉換函數無法執行 (conversion function not implemented)	應 用 程

		式
	功能方塊無法執行 (function block not implemented)	應 用 程 式
	標準函數無法執行 (standard function not implemented)	應 用 程 式
	實數除以零 (real divided by zero)	程 式
	無效的操作參數 (invalid operate parameters)	應 用 程 式
	應用程式符號無法變更 (application symbols cannot be modified)	應 用 程 式
	無法更新：不同的布林變數序列 (cannot update: different set of boolean variables)	應 用 程 式
	無法更新：不同的類比變數序列 (cannot update: different set of analog variables)	應 用 程 式
	無法更新：不同的計時器變數序列 (cannot update: different set of timer variables)	應 用 程 式
	無法更新：不同的訊息變數序列	應

目標硬體使用者指南

	(cannot update: different set of message variables)	用 程 式
	無法更新：找不到新的應用 (cannot update: cannot find new application)	應 用 程 式
	特定的 OEM 錯誤碼，詢問提供廠商以取得進一步資料	

有 3 種型態的錯誤對應到不同的問題來源：

系統錯誤：

此類問題也許肇因於目的軟體或硬體，並非因應用的設定或程式的執行而造成。

試著將目標系統做冷開機動作 (power off)，並試著執行其他的應用程式。這種錯誤應將之報告給你的 ISaGRAF 提供廠商。

應用程式錯誤：

此類問題肇因於應用程式的參數、容量或內容。

當載入一個已知並且之前驗證過的應用程式，這種錯誤應會消失。若問題還在，則此問題應屬上列的系統錯誤。

程式錯誤：

此類問題肇因於錯誤程式中特別的程序。

當應用程式以單步執行模式啓始，或當相關對應的程式停止時，這類錯誤應會消失。

錯誤描述：

1. 無法配置執行時資料庫存取所需的記憶體	系統
-----------------------	----

記憶體不足，檢查硬體。

2. 不正確的應用程式資料 (Motorola/Intel)	應用程式
--------------------------------	------

下載或備份的應用程式檔案不正確。若應用程式以 INTEL 模式編譯，卻下載至 MOTOROLA 硬體（及上載），或檔案被修改過，則會發生此種錯誤。

3. 無法配置通訊郵包	系統
-------------	----

若通訊程式無法配置足夠的空間給不同程式間的內部通訊使用，則會發生此種錯誤。

4. 無法連結核心程式資料庫	系統
----------------	----

若通訊程式發現核心程式的 slave 編號與其命令列所給的不同，則會發生此種錯誤。

5. 傳送 ^{通訊詢問} 至核心程式逾時	系統
-------------------------------	----

若通訊程式無法送出一個詢問命令給核心程式，則會發生此種錯誤。此時核心程式可能未執行，或處於忙碌狀態。

6. 等待核心程式回答逾時	系統
---------------	----

若通訊程式無法收到核心程式的回答，則會發生此種錯誤。此時核心程式可能未執行，或處於忙碌狀態。

7. 無法將通訊初始化

系
統

當通訊軟體層無法將實際硬體連線初始化，則會發生此種警告。若通訊路徑未被指定，則此警告亦會顯示。這狀況不會影響目的程式正常的運作，只是不能通訊而已。

8. 無法配置記憶變數所需的記憶體

應
用
程
式

表示 ISaGRAF 此時無法管理記憶變數。這問題可能有兩種原因：
傳給主程式的參數字串語法不對
每個儲存區塊所需的記憶體不足
你必須檢查你‘記憶變數’參數的語法是否正確，以及試著減少記憶變數的數量。

9. 應用程式終止

應
用
程
式

每次從除錯程式裡停止應用程式，便會產生此種告警。

10. 太多同時的 N 或 P 行為

應
用

	程 式
--	--------

若一個程式週期中必須執行太多的非儲存 (non stored)、脈衝式 (pulse) 行為或週期性方塊 (cyclic block)，則會發生此種錯誤。可使用 CC (Cycle by Cycle) 模式找出問題所在。每個 SFC 程式最大容許的同步行為數量為 2+4。

11. 太多同時的設定行為	應 用 程 式
---------------	------------------

若一個程式週期中必須執行太多的設定行為（於一步驟變為執行狀態時執行），則會發生此種錯誤。

12. 太多同時的重設行為	應 用 程 式
---------------	------------------

若一個程式週期中必須執行太多的重設行為（於一步驟變為非執行狀態時執行），則會發生此種錯誤。

13. 未知的 TIC 指令	應 用 程 式
----------------	------------------

核心程式偵測到一些應用程式碼發生錯誤（稱為目標硬體 Independent Code）。有兩種可能的解釋：

外部的程式可能對應用程式碼做寫入的動作。試著從 CC 模式中找出問題所在，並應確定 IO 介面沒有給錯參數。

目標硬體使用者指南

你的目標系統只能提供較少的指令，而你的應用使用到不合法的指令或變數型態。

16. 無法回答讀值詢問命令	系 統
----------------	--------

表示在回答 ISaGRAF Modbus 詢問命令功能碼 18 時 (檔案讀取) 發生通訊錯誤。請檢查主電腦與目標系統的連線與系統設定。

17. 無法回答寫值詢問命令	系 統
----------------	--------

表示在回答 ISaGRAF Modbus 詢問命令功能碼 17 時 (檔案寫入) 發生通訊錯誤。請檢查主電腦與目標系統的連線與系統設定。

18. 無法回答除錯程式溝通詢問命令	系 統
--------------------	--------

表示在回答除錯程式詢問命令時發生通訊錯誤。請檢查主電腦與目標系統的連線與系統設定。

19. 無法回答 modbus 詢問命令	系 統
----------------------	--------

表示在回答 Modbus 詢問命令時發生通訊錯誤。請檢查主電腦與目標系統的連線與系統設定。

20. 無法回答除錯程式應用詢問命令	系 統
--------------------	--------

表示在回答除錯程式詢問命令時發生通訊錯誤。請檢查主電腦與目標系統的連線與系統設定。

21. 無法回答除錯程式	系統
--------------	----

表示在回答除錯程式詢問命令時發生通訊錯誤。請檢查主電腦與目標系統的連線與系統設定。

23. 未知的詢問命令碼	系統
--------------	----

除錯程式詢問命令無意義。

24. 乙太網路通訊錯誤	系統
--------------	----

每當除錯程式關閉，通訊終止時，則會發生此種錯誤：代表系統作業正常。否則，此錯誤表示偵測到乙太網路通訊的錯誤狀況。請檢查主電腦與目標系統的連線與系統設定。

其中第二欄可為：

- 1：收送時的錯誤
- 2：產生 socket 時的錯誤
- 3：連結或讀取 socket 時的錯誤
- 4：接受一個新客戶端時的錯誤

25. 通訊同步錯誤	系統
------------	----

在目標系統與主電腦間的通訊程式發生同步錯誤。請檢查主電腦與目標系統的連線與系統設定 (通訊參數)。

28. 無法配置應用程式所需記憶體	系統
-------------------	----

記憶體不足。根據應用程式的大小檢查硬體。

29. 無法配置應用程式更新所需記憶體	系統
---------------------	----

記憶體不足。根據應用程式的大小檢查硬體。

30. 未知的 OEM key 碼	應用程式
-------------------	------

應用程式使用到一塊無法辨認硬體製造廠商編號的 IO 卡。檢查工作平台中的 IO 連結部份，並使用‘虛擬 (VIRTUAL)’屬性選項來找出錯誤的 IO 卡。你的工作平台程式庫可能沒有對應到你的目標硬體的版本。

31. 無法初始化布林輸入 IO 板	應用程式
--------------------	------

初始化布林輸入 IO 板失敗。檢查工作平台中的 IO 連結部份，以及布林輸入 IO 板的參數。

32. 無法初始化類比輸入 IO 板	應用程式
--------------------	------

初始化類比輸入 IO 板失敗。檢查工作平台中的 IO 連結部份，以及類比輸入 IO 板的參數。

33. 無法初始化訊息輸入 IO 板	應 用 程 式
--------------------	------------------

初始化訊息輸入 IO 板失敗。檢查工作平台中的 IO 連結部份，以及訊息輸入 IO 板的參數。

34. 無法初始化布林輸出 IO 板	應 用 程 式
--------------------	------------------

初始化布林輸出 IO 板失敗。檢查工作平台中的 IO 連結部份，以及布林輸出 IO 板的參數。

35. 無法初始化類比輸出 IO 板	應 用 程 式
--------------------	------------------

初始化類比輸出 IO 板失敗。檢查工作平台中的 IO 連結部份，以及類比輸出 IO 板的參數。

36. 無法初始化訊息輸出 IO 板	應 用 程 式
--------------------	------------------

初始化訊息輸出 IO 板失敗。檢查工作平台中的 IO 連結部份，以及訊息輸出 IO 板的參數。

37. 無法輸入布林IO 板	應 用 程 式
----------------	------------------

當更新一塊布林輸入卡數值時偵測到錯誤。檢查工作平台中的IO 連結部份，以及IO 板的參數。

38. 無法輸入類比IO 板	應 用 程 式
----------------	------------------

當更新一塊類比輸入卡數值時偵測到錯誤。檢查工作平台中的IO 連結部份，以及IO 板的參數。

39. 無法輸入訊息IO 板	應 用 程 式
----------------	------------------

當更新一塊訊息輸入卡數值時偵測到錯誤。檢查工作平台中的IO 連結部份，以及IO 板的參數。

40. 無法輸出布林輸出變數	應 用 程 式
----------------	------------------

當更新一塊布林輸出卡數值時偵測到錯誤。檢查工作平台中的IO 連結部份，以及IO 板的參數。

41. 無法輸出類比輸出變數	應 用 程 式
----------------	------------------

當更新一塊類比輸出卡數值時偵測到錯誤。檢查工作平台中的 IO 連結部份，以及 IO 板的參數。

42. 無法輸出訊息輸出變數	應 用 程 式
----------------	------------------

當更新一塊訊息輸出卡數值時偵測到錯誤。檢查工作平台中的 IO 連結部份，以及 IO 板的參數。

43. 無法操作布林變數	應 用 程 式
--------------	------------------

對布林變數執行 OPERATE 呼叫時偵測到錯誤。檢查你的 OPERATE 參數以及 IO 板的使用資料。

44. 無法操作類比變數	應 用 程 式
--------------	------------------

對類比變數執行 OPERATE 呼叫時偵測到錯誤。檢查你的 OPERATE 參數以及 IO 板的使用資料。

45. 無法操作訊息變數	應 用 程 式
--------------	------------------

對訊息變數執行 **OPERATE** 呼叫時偵測到錯誤。檢查你的 **OPERATE** 參數以及 IO 板的使用資料。

46. 無法開啓 IO 板	應 用 程 式
---------------	------------------

應用程式使用到目標系統不認識的 IO 板參考資料 (reference)。檢查工作平台中的 IO 連結部份。你的工作平台程式庫可能沒有對應到你的目標硬體的版本。

47. 無法關閉 IO 板	應 用 程 式
---------------	------------------

應用程式使用到目標系統不認識的 IO 板參考資料 (reference)。檢查工作平台中的 IO 連結部份。

50. 無法覆寫布林輸出變數	程 式
----------------	--------

兩個 **SFC** 程序在相同的程式週期中寫入相同的布林輸出變數。這種情況應避免，以免對 IO 做出危險的動作。若發生此種衝突的情況，則以階層較高的程式為優先。若兩個 **SFC** 程式在同一個階層，則結果將不可預測。

51. 無法覆寫類比輸出變數	程 式
----------------	--------

兩個 SFC 程序在相同的程式週期中寫入相同的類比輸出變數。請參考上文。

52. 無法覆寫訊息輸出變數	程 式
----------------	--------

兩個 SFC 程序在相同的程式週期中寫入相同的訊息輸出變數。請參考上文。

61. 未知的系統詢問命令碼	程 式
----------------	--------

程式在使用 SYSTEM 呼叫時用到無效碼。

62. 取樣週期溢位	程 式
------------	--------

目的程式週期週期較工作平台中給定的值為長。在多工系統中，這意味著執行一個程式週期所需的 CPU 時間不足，即使‘目前的週期週期 (current cycle duration)’比給定的值少。

在一個單工系統中，這意味著一個程式週期週期中執行了太多的命令。

有很多可行的方法移除這種警告：

減少偵測到警告元件中的指令數目。

減少權杖記號 (tokens) 的數目，條件判斷式 (transition) 的數目，將複雜程序最佳化等。

減少其他程式佔用 CPU 的時間，讓 ISaGRAF 有更多的執行時間。

減少通訊佔用的時間，讓 ISaGRAF 有更多的執行時間。

使用動態週期週期的調節方式，使週期週期能適應不同的程序階段。

目標硬體使用者指南

將週期週期設為 0，使 ISaGRAF 核心程式能以最快的速度執行，不做週期溢位的檢查。

63. 使用者函數無法執行	應 用 程 式
---------------	------------------

應用程式使用到目標系統不認識的 C 函數。你的工作平台程式庫可能沒有對應到你的目標硬體的版本。

64. 整數除以零	程 式
-----------	--------

程式中發生整數除以零的情況。你的應用程式應避免這種情況發生，以免產生不可預測的結果。

當這情況發生時，ISaGRAF 會將結果值設為最大的類比值。若運算結果為負值，則結果正好相反。

65. 轉換函數無法執行	應 用 程 式
--------------	------------------

應用程式使用到目標系統不認識的轉換函數。你的工作平台程式庫可能沒有對應到你的目標硬體的版本。

當此情況發生時，ISaGRAF 將不會轉換數值。

66. 功能方塊無法執行	應 用 程 式
--------------	------------------

應用程式使用到目標系統不認識的 C 功能方塊。你的工作平台程式庫可能沒有對應到你的目標硬體的版本。

67. 標準函數無法執行	應 用 程 式
--------------	------------------

應用程式使用到目標系統不認識的功能方塊，即使其應已被大多數的目標系統所支援。請詢問你的供應商。

68. 實數除以零	程 式
-----------	--------

程式中發生實數除以零的情況。你的應用程式應避免這種情況發生，以免產生不可預測的結果。

當這情況發生時，ISaGRAF 會將結果值設為最大的實數類比值。若運算結果為負值，則結果正好相反。

69. 無效的操作參數	應 用 程 式
-------------	------------------

你的應用程式使用 OPERATE 呼叫時參數錯誤。這問題通常可由編譯器檢測出來，可能是錯誤的計時器參數，或是非數入輸出的變數所造成。

72. 應用程式符號無法變更	應 用 程 式
----------------	------------------

目標硬體使用者指南

當欲更新應用程式時，因符號不同，變動過的程式無法啓始，與目前的應用程式比較，新增或移除了一個以上的變數或功能方塊元件。

73. 無法更新：不同的布林變數序列	應 用 程 式
--------------------	------------------

更動過的應用程式無法啓始，因與目前的應用程式比較，新增或移除了一些布林變數。

74. 無法更新：不同的類比變數序列	應 用 程 式
--------------------	------------------

更動過的應用程式無法啓始，因與目前的應用程式比較，新增或移除了一些類比變數。

75. 無法更新：不同的計時器變數序列	應 用 程 式
---------------------	------------------

更動過的應用程式無法啓始，因與目前的應用程式比較，新增或移除了一些計時器變數。

76. 無法更新：不同的訊息變數序列	應 用 程 式
--------------------	------------------

更動過的應用程式無法啓始，因與目前的應用程式比較，新增或移除了一些訊息變數。

77. 無法更新：找不到新的應用	應 用 程 式
------------------	------------------

在記憶體中找不到更動過的應用程式，可能在下載的過程中發生錯誤。

D. 字彙表

Action	行爲	當 SFC 程式的步驟動作時，敘述式集以指定的方法執行。
Action (FC)	行爲 (F C)	流程圖的符號。行爲包含了一組指令，這些指令是當流程執行到此行爲符號時將被執行的指令。
Activity of a step	步驟 屬性	步驟的屬性；步驟動作時，其上會標以一個 SFC 權杖記號，所附屬的行爲也會根據它的動作狀態來執行。
Analogue	類 比	變數的型態；為連續的整數或實數變數。
Attribute	屬 性	變數的類別；可用的屬性有：內部、輸入及輸出。
Beginning section	開 始 區	週期程式的一種族群；會於每個目標硬體週期開始時執行。

字彙表

Beginnings step	序 始 步 驟	巨集步驟本體的第一步。它之前不會連結任何的轉移條件。
Boolean	有 本	變數的型態；僅能為真或假等數值。
Boolean action	有 本 行 為	SFC 行為的一種；此行為之布林變數會以步驟的動作狀態來指定它的值。
Breakpoint	中 斷 點	除錯時，使用者於 SFC 步驟或轉移條件上所做的記號，當 SFC 權杖記號移至中斷點時，目標硬體的執行會停止於此。
C function	C 函 數	以“C”語言所寫的函數，以同步的方式由 ISaGRAF 程式（以其他程式所寫的）所呼叫。C 函數是由 ICS Triplex ISaGRAF Inc. 公司所附送的，或者由使用者自行發展。
C language	C 語 言	高階文字式語言；可用來敘述電腦的運算（如 C 函數、轉換函數等）。
C source code	C 原 始 碼	包含 C 函數或轉換函數的“C”原始碼的一種文字檔。

C source header	C 源 頭 部 分	包含 C 函數或轉移函數設計時所需的“C”定義及型態的一種文字檔。
Cell	有 關	SFC、FBD 或 LD 等圖形式語言的最小圖形矩陣區域。
Child SFC program	S F C 子 程 式	由另一個 SFC 父程式所控制的 SFC 程式。
Clearing a transition	消 除 轉 移 作 用	執行期的操作：所有於此前一步驟的權杖記號皆會被移除，然後於緊接它之後的每一個步驟放上權杖記號。
Coil	繼 電	LD 程式的圖形元件；用於表示輸出變數的指定方法。
Comment	註 解	程式內的一種文字，但對於程式的執行沒有影響。

字彙表

Comment (SFC)	  (S F C)	附屬於 SFC 步驟或轉移條件的一種文字，但對於程式的執行沒有影響。
Common	 	定義字的範圍；此種物件能用於任何案件的任何程式。
Condition (for a transition)	  (      )	依附於 SFC 轉移條件的布林表示式。當轉移條件的條件為假時，無法清除轉移條件。
Connector (FC)	   (F C)	FC 圖形元件，它表示從流程圖的一點連結到一個 FC 行為或測試。跳躍的圖形符號是一個小圓圈，其編號為目的元件的參考號碼。

Constant expression	常 量 表 示 式	文字表示式；用於描述常數值，且表示式內只能使用一種型態。
Contact	接 點	LD 程式的圖形元件；代表輸入變數的狀態。
Conversion	轉 換	依附於類比輸入或輸出變數的過濾器。每一次變數輸入或輸出時，依附的轉換會自動應用於其上。
Conversion function	轉 換 函 數	以“C”語言撰寫的函數；用來描述轉換。此種轉換可以依附於任何的類比輸入或輸出變數上。
Conversion table	轉 換 表	用於定義線性轉換的點集合。此種轉換可以依附於任何的類比輸入或輸出變數上。
Cross reference	交 叉 參 照	由 ISaGRAF 工作平台所計算關於案件內所用的變數與它的相關位置的一種資訊。

字彙表

Current result (IL)	IL 程式的指令結果。目前的結果能由指令所變更，或用來設定變數。
Cycle timing	目標硬體執行週期的時間。
Cycle to cycle mode	執行模式：於此模式中，會根據使用者於除錯模式下的命令，一次執行一個週期。
Cyclic	程式的屬性；表示每個週期皆會執行。
Defined word	相等的識別字；用來取代任何程式所用的表示式。
Defined word	在程式中用來取代任何表示式的唯一識別字。

Delay operation (IL)	延 延 延 集	IL 程式的一種運算；當程式中碰到“(”指令時，會先執行“(”與“)”內的敘述。
	(I L)	
Diary	日 記	一種文字檔，包含一個程式改變時所有做過的註記。每一個註記由編輯日記時所完成。
Dictionary	字 典	用於案件內的程式的內部變數、輸入變數、輸出變數與定義字的集合。
Edge	邊 緣	布林變數的改變。上升邊緣表示從假變為真，下降邊緣表示由真變為假。
End section	期 牙 區	週期程式的一種族群；會於每個目標硬體週期結束時執行。
Ending step	期 牙 步 驟	巨集步驟本體的最後一步。它之後不會連結任何的轉移條件。
Expression	運 算 子	運算元與運算子的集合，用來表示數值的求取。

字彙表

Father SFC program	S F C 多 種 正	控制其他 SFC 子程式的 SFC 程式。
FBD	F B D	功能方塊語言
FC	F C	即“流程圖”。
Flow Chart	流 程 圖	用來設計流程的繪圖式語言。它的圖形包含將被執行的行為及允許流程中各種不同路徑間選擇的決策。流程圖語言可以讓迴路的圖樣在連續的週期中執行。
Function block	功 能 方 塊	FBD 程式的圖形元件；用來表示 ISaGRAF 程式庫的函數。
Functional Block Diagram	功 能 方 塊 圖	圖形式語言：將 ISaGRAF 程式庫元件以方塊形式表示，且各方塊連結在一起，形成方程式。
Global	全 局	變數或定義字的範圍。這些物件可以用於一個案件的任何程式中。

Hierarchy	在 此 系 存	案件的架構，於一些程式所組成。樹狀結構表示父程式與子程式（或副程式）之間的連結關係。
I/O board	I / C 在	硬體資源。I/O 板具有同型態及同方向的特性，它的參數可以描述於 ISaGRAF 程式庫中。
I/O channel	I / C 這 系 無	I/O 板的單連結點。I/O 板可以接受一個 I/O 變數。
I/O connection	I / C 這 系	定義應用程式變數與目標硬體 I/O 板上的連結點之間的連結關係。
I/O variable	I / C 變 量	連結到輸入或輸出裝置的變數。I/O 變數必須連結至 I/O 板的連結點上。
Identifier	識 別 符	於程式中用來表示變數或常數表示式的唯一名稱。

字彙表

IL	I L	指令集語言。
Initial situat ion	初 始 狀 態	SFC 程式的初始步驟集，它表示程式開始的內容。
Initial step	初 始 步 驟	SFC 程式的特別的步驟，程式開始時它會變為動作的狀態。
Input	輸 入	變數的屬性。此種變數將連結至輸入裝置。
Instr uctio n	指 令	IL 程式基本運算，以文字列的方式輸入。
Instr uctio n List	指 令 集	低階的文字式語言，以基本的運算列集所組成。
Integ er	整 數	類比變數的類別，以 32 位元不帶符號整數格式儲存。
Inter nal	內 部	變數的屬性，它不連結至輸入或輸出裝置。

Jump to a step	    	SFC 圖形元件，表示轉移條件至步驟之間的連結。以箭頭形狀來代表，上面標上目的步驟的參考號碼。
Keyword	  	語言的保留字。
Label (IL)	     	放置於 IL 指令列前端的識別字，它代表此行指令，能夠用於 JMP 運算元之運算子名稱上。
Ladder Diagram	  	混合接點及線圈之圖形式語言，主要用於布林表示式。
LD	 	階梯圖。
Level 1 of the SFC	     	SFC 程式的主要描述，包含圖形（步驟及轉移條件）及註解。

字彙表

Level 2 of the SFC	S F C 身 二 雇	SFC 程式的細節描述，為步驟的行為描述及轉移條件依附的布林條件。
Library	系 立 馬	硬體及軟體的資源集合，能直接用於任何的應用程式中。
Local	區 埠	變數或定義字的範圍。此物件僅能用於單一案件的單一程式中。
Locked I/O	鎖 住 I / C	使用者可以於除錯器模式下執行“鎖住”命令，則會中斷輸出入變數與相對的 I/O 裝置之間的關係。
Macro step	巨 身 步 勝	SFC 圖形元件。巨集步驟是一個單獨的步驟及轉移條件的群組，於主流程中以單一的符號來表示。
Matrix	矩 陣	當編輯圖形式語言程式時，以長方形格點將編輯區劃分為許多的邏輯區間。
Message	訊 息	變數型態。此種變數包含可變長度的字元字串。

Modbus	主從 (Master-Slave) 架構的通訊格式。 ISaGRAF 目標硬體為一個 Modbus slave，可以用來連結外部的系統（如監控系統）。
Modifier (IL)	放置於 IL 運算元關鍵字之後的單一字元，可用來變更運算元的意義。
	(I L)
Network address	可選擇的 16 進位位址，用來定義變數。當目標系統連結至外部系統時，這個位址用於 Modbus 通訊中。
Non-store d action	SFC 行為；當相對應的步驟動作時，它的敘述式集於每個目標硬體週期皆會執行。

字彙表

OEM	C	16 進位的 16 位元碼；定義 ISaGRAF 程式庫
key	E	中的 I/O 板編號。OEM 碼可以分辨不同供應
code	N	商的 I/O 板。
(I/O	參	
board	變	
)	數	
	(	
	I	
	/	
	C	
	板	
	)	
OEM	C	由 I/O 板設計者所定義的參數，它可以是一個
para	E	常數，或 I/O 板連結期間使用者輸入的變動參
meter	N	數。
(I/O	參	
board	變	
)		
	(	
	I	
	/	
	C	
	板	
	)	

Oper and (IL)	這 隻 子 (I L)	由 IL 指令所處理的變數或常數表示式。
Oper ation (IL)	這 隻 子 (I L)	IL 語言的基本指令。於指令中，運算元通常與運算子一起使用。
Outp ut	轉 占	變數的屬性。這些變數將連結到目標機器的輸出裝置上。
Para mete r (C funct ion)	參 數 (C 語 言 隻)	“C” 函數的輸入值。參數需具備一種型態。

字彙表

Parameter (I/O board)	參 數 (I / C 板)	I/O 板的常數參數或使用者定義的參數。使用者定義的參數於 I/O 連結時，由程式設計者輸入。
Parent program	父 程 式	以任何語言寫成的程式，可以控制 (呼叫) 另一個非 SFC 程式 (稱為它的副程式)。
Power rail	負 入 電	階梯圖形中，位於兩端的主要的左及右垂直電力軌。
Program	程 式	案件的基本程式設計單元。一個程式以一種語言來描述，且被放進案件的樹狀結構中。
Project	身 份	程式設計區，包含一個 ISaGRAF 應用程式的所有資訊 (程式、變數、目標碼...)
Pulse action	脈 衝 行 為	SFC 行為：它是一種敘述式，當它的相對步驟動作時，它僅會執行一次。
Range	範 圍	物件能被程式集使用的程度。預定的 ISaGRAF 範圍有共同、全域及區域。

Real	眞 婁	類比變數的類別。以 IEEE 標準單精度 32 位元浮點數格式儲存。
Real board	眞 眞 尤	真正連到目標機器 I/O 裝置的 I/O 板。
Real time mode	眞 眞 尤 厶	正常的即時執行模式：目標硬體週期由程式的週期時間所觸發。
Reference number (SFC)	眞 ㄣ 易 ㄩ (S F C)	SFC 程式中的步驟或轉移條件的 10 進位識別號碼 (從 1 到 65535)。
Register (IL)	眞 ㄗ ㄟ (I L)	IL 指令序列的目前結果。

字彙表

Return value of a sub-program	副程式節結束時傳回的值。回傳值可以被呼叫程式的運算式所利用。
Run time error	ISaGRAF 目標系統於執行期所偵測到的應用程式錯誤。
Section	以相同動態規則執行的程式群組。
Separator	文字式語言中用來分開識別字的特殊字元 (或字元組)。
Sequential Function Chart	圖形式語言：由步驟集 (之間以轉移條件連結) 所描述的程序。步驟會附上行為，而轉移條件則以布林條件為內容。
Sequential section	案件的程式群組。這些程式以 SFC 語言的動態規則執行。

SFC	S F C	順序式功能圖。
ST	S T	結構化文字。
State ment	糸 辺 エ	ST 的基本運算式。
Step	タ 馬	SFC 語言的基本圖形元件。步驟表示程序不變的情況，以方塊表示。步驟具有參考號碼。步驟的動作用於控制相對行為的執行。
Strin g	弓 与	儲存於訊息變數的字元集。
Struc tured Text	糸 存 ハ マ 弓	高階結構化文字式語言，包含指令、高階結構 (如 If/Then/Else) 及函數呼叫。
Sub- progr am	畠 糸 エ	以 SFC 以外的語言所撰寫的程式，可以由另一個程式所呼叫。
Targ et t	厩 有 也 量	可以支援 ISaGRAF 核心程式軟體的 ISaGRAF 目標機器。

字彙表

Target cycle		每次 ISaGRAF 目標系統動作時設定運算集執行。週期由設計的週期時間所觸發。
Technical note		ISaGRAF 程式庫元件 (C 函數、功能方塊、轉換函數或 I/O 板) 的使用者指南。技術註記由元件的設計者所撰寫。
Test (FC)		法 (亦稱為決定) 流程圖符號，以布林運算式來表示其狀態，然後根據此狀態來做為輸出 (指向 YES 或 NO 符號)。
Timer		變數型態。此變數包含時間值，於執行期時由 ISaGRAF 自動更新。

Toke n (SFC)	有 本 言 易 (S F C)	圖形記號；用於顯示 SFC 程式的動作步驟。
Tool box	工 具 盒	圖形編輯工具視窗的小的子視窗，包含圖形元件的主按鈕。
Top level progr am	頂 層 程 式	位於樹狀結構頂層的程式。頂層程式由系統自動呼叫起來執行。
Tran sition	轉 移 條 件	基本的 SFC 圖形元件。轉移條件表示不同 SFC 步驟之間的條件。轉移條件具有參考號碼。每一個轉移條件會附上一個布林條件。
Type	型 息	相同變數的變數類別。基本的型態有布林、類比、計時器及訊息。

字彙表

Validi ty of a trans ition	真 性 條 件 轉 移 交 代	轉移條件的屬性。當所由之前連結的步驟動作時，此轉移條件有效。
Varia ble	變 量	用於案件程式中的元素資料的唯一表示字。
Virtu al boar d	虛 擬 板	未連結到目標機器 I/O 裝置的 I/O 板。

E. 索引

符號

-, B-102

\$ 序列, B-12

%, A-95, B-14

&, B-98

) 運算符號 (IL), B-91

*, B-103

/, B-104

:=, A-149

:= (ST 指定), B-69

+, B-101

<, B-108

<=, B-109

<>, B-113

=, B-112

=1, B-100

>, B-110

>=, B-111

>=1, B-99

1

1 gain, B-96

4

4 個輸入多路選擇器, B-164

8

8 個輸入多路選擇器, B-165

A

ABS, B-147

ACOS, B-152

Action, D-1

Activity of a step, D-1

ANA, B-115

Analog, D-1

AND, B-98

AND_MASK, B-105

AnyTarget, A-109

appli.tst, C-7, C-20, C-36, C-45

appli.x6m, C-20, C-36

appli.x8m, C-7, C-45

ARCREATE, B-181

ARREAD, B-182

ARWRITE, B-183

ASCII, B-169

ASIN, B-152

ATAN, B-153

Attribute, D-1

AVERAGE, B-138

B

Begin section, D-1

Beginning step, D-1

BinaryFile, A-108

Bitmap, A-131

BLINK, B-144

BOO, B-114

Boolean, D-2

Boolean action, D-2

索引

Breakpoint, D-2

BY, B-75

C

C function, D-2

C language, D-2

C source code, D-2

C source header, D-2

C 功能方塊, A-164, C-55

C 函數, A-164, C-55, C-63, D-2

C 原始標頭檔, C-58, C-66, C-76, C-92,
D-2

C 原始碼, A-105, C-60, C-67, C-78,
C-92, D-2

C 語言, C-55, C-58, C-60, C-66, C-76,
C-78, C-92, D-2

C 編譯器, C-55, C-92

CAL 運算元 (IL), B-93

CASE, B-72

Cat, B-119

Cell, D-2

CHAR, B-170

Child SFC program, D-3

Clearing a transition, D-3

CLKRATE, C-24

CMP, B-135

Coil, D-3

Comment, D-3

Comment (SFC), D-3

Common, D-3

Condition (for a transition), D-3

Connector, D-4

Constant expression, D-4

Contact, D-4

Conversion, D-4

Conversion function, D-4

Conversion table, D-4

COS, B-154

Cross references, D-4

CTD, B-130

CTU, B-129

CTUD, B-131

Current result (IL), D-5

Cycle, A-151

Cycle timing, D-5

Cycle to cycle mode, D-5

Cyclic, D-5

C 程式碼, A-158

D

DAY_TIME, B-180

DDE, A-126

DDE (NT 目標硬體), C-46, C-50, C-53

Decision, D-5

Defined word, D-5

Delayed operation (IL), D-5

DELETE, B-171

DERIVATE, B-143

Diary, D-6

Dictionary, D-6

DO, B-73, B-75

E

Edge, D-6

ELSE, B-71, B-72

ELSIF, B-71

EN, A-53
End, A-154
End section, D-6
END_CASE, B-72
END_FOR, B-75
END_IF, B-71
END_REPEAT, B-74
END_WHILE, B-73
Ending step, D-6
ENO, A-53
EPROM, C-19, C-35
EQ 運算元 (IL), B-112
Ethernet, A-29
EXIT, B-76
Expression, D-6
EXPT, B-147

F

F_CLOSE, B-186
F_EOF, B-187
F_ROPEN, B-184
F_TRIG, B-127
F_WOPEN, B-185
FA_READ, B-189
FA_WRITE, B-190
Father SFC program, D-6
FBD, A-61, B-44, C-64, C-74, D-6
FBD 註解, A-64
FBD 編輯器, A-61, A-73
FC, A-42, B-38, D-6
FC 副程式, B-40
FC 連結, A-45
FC 連結器, A-45

FC 註解, A-45
FC 編輯器, A-42
FC 副程式, A-18
FEDGE, B-68
FIND, B-172
Flow Chart, D-7
FM_READ, B-193
FM_WRITE, B-195
FOR, B-75
From, A-110
Function block, D-7
Functional Block Diagram, D-7

G

gain 1, B-96
GE 運算元 (IL), B-111
GFREEZE, B-37, B-81
GKILL, B-37, B-81
Global, D-7
Goto, A-153
GRST, B-37, B-82
GSTART, B-37, B-80
GSTATUS, B-37, B-82
GT 運算元 (IL), B-110

H

Hierarchy, D-7
HYSTER, B-140

I

I/O, A-25, A-92, A-93, A-95, A-115,
A-119, A-142, A-159, A-160, A-161,
A-176, A-178
I/O board, D-7

索引

I/O channel, D-7
I/O connection, D-8
I/O variable, D-8
I/O 板, A-161, D-7
I/O 特定動作, B-39
I/O 特定動作, B-41
I/O 接點操作, B-122
I/O 組態, A-13, A-159
I/O 連結, A-25, A-92, D-8
I/O 連結點, D-7
I/O 複合設備, A-160
I/O 變數, D-8
Identifier, D-8
If, A-153
IF, B-42, B-71
IL, A-71, A-129, B-32, B-34, B-84, D-8
IL 中呼叫函數, B-92
IL 語言中呼叫功能方塊, B-93
IL 語言中呼叫副程式, B-92
IL 編輯器, A-73
Initial situation, D-8
Initial step, D-8
Input, D-8
INSERT, B-173
Instruction, D-8
Instruction List, D-9
Integer, D-9
INTEGRAL, B-141
Internal, D-9
IO 變數, C-57, C-58
ISA 工作 (OS9), C-10
ISA.EXE, C-4
ISA.O (VxWorks), C-23, C-24

isa_main, C-26, C-30
isa_register_slave, C-25
ISAGRAF.INI (NT 目標硬體), C-39
ISAKER 工作 (OS9), C-12
ISAKERET.O (VxWorks), C-23, C-27
ISAKERSE.O (VxWorks), C-23, C-27
ISAMOD (VxWorks), C-23
ISAMOD.EXE, C-5
ISANET 工作 (OS9), C-14
ISASSR.O (VxWorks), C-23
ISATST 工作 (OS9), C-13
ISAx0, C-19
ISAx1, C-6, C-18, C-19
ISAx1 (NT 目標硬體), C-45
ISAx1 (VxWorks), C-34
ISAx2, C-19
ISAx3, C-19
ISAx4, C-19
ISAx5, C-19
ISAx6, C-6, C-18, C-19
ISAx6 (NT 目標硬體), C-45
ISAx6 (VxWorks), C-35

J

JMP 運算元 (IL), B-89
Jump to a step, D-9

K

Keyword, D-9

L

Label, A-152
Label (IL), D-9
Ladder Diagram, D-9

LD, A-39, A-49, A-51, A-61, B-49, D-9

LD 運算元 (IL), B-87

LD 編輯器, A-51

LE 運算元 (IL), B-109

LEFT, B-174

Level 1 of the SFC, D-10

Level 2 of the SFC, D-10

Library, D-10

LIM_ALARM, B-140

LIMIT, B-162

Local, D-10

Locked I/O, D-10

LOG, B-148

LT 運算元 (IL), B-108

M

Macro step, D-10

Matrix, D-10

MAX, B-161

Message, D-10

Metafile, A-131

MID, B-175

MIN, B-160

MLen, B-176

MOD, B-163

Modbus, D-11

MODBUS, A-86, C-100

Modification tracking, A-68

Modifier (IL), D-11

MSG, B-118

MUX4, B-164

MUX8, B-165

N

N 修飾語, A-38

NE 運算元 (IL), B-113

NEG, B-97

Network address, D-11

Non stored, A-38

Non-stored action, D-11

NOT, A-63, A-65

NOT_MASK, B-108

NT (保護鎖), A-5

O

ODD, B-166

OEM key code, D-11

OEM parameter (I/O board), D-11

OEM 參數, A-161

OEM 參數 (I/O 板), D-11

OEM 識別碼, A-161, D-11

OF, B-72

Operand (IL), D-12

Operation (IL), D-12

OR, A-62, B-99

OR_MASK, B-106

OS9 命令列, C-22

Output, D-12

P

P 修飾語, A-37

P0 修飾語, A-38

P1 修飾語, A-38

Parameter (C function), D-12

Parameter (I/O board), D-12

Parent program, D-12

索引

POW, B-149

Power rail, D-12

Print, A-150

PrintTime, A-150

Program, D-13

Project, D-13

PROM, C-19, C-35

Pulse, A-37

Pulse action, D-13

Pulse 計時, B-135

Q

Quick LD, A-39, A-49, A-51

Quick LD 編輯器, A-73

R

R (reset) 運算元 (IL), B-88

R_TRIG, B-126

RAND, B-167

Range, D-13

Real, D-13

REAL, B-116

Real board, D-13

Real time mode, D-13

REDGE, B-67

Reference number, D-13

Register (IL), D-14

REPEAT, B-42, B-74

REPLACE, B-177

RET 運算元 (IL), B-90

RETURN, B-45, B-59, B-70

Return value, D-14

RIGHT, B-179

ROL, B-157

ROR, B-157

RS, B-125

Run time error, D-14

S

S (set) 運算元 (IL), B-88

Section, D-14

SEL, B-168

SEMA, B-128

Separator, D-14

Sequential Function Chart, D-14

Sequential section, D-15

SFC, A-30, A-103, A-121, A-172, B-20,
C-64, D-15

SFC 子程式, A-39, B-3, B-37, D-3

SFC 父程式, B-37, D-6

SFC 的動態規則, B-36

SFC 的第一層, B-20, B-22

SFC 的第二層, B-26

SFC 倉庫, A-40

SFC 第一層, D-10

SFC 第二層, D-10

SFC 編輯器, A-30, A-73

SFC 子程式, A-18

SHL, B-158

SHR, B-159

SIG_GEN, B-144

SIN, B-155

Slave 號碼, C-11, C-12, C-13, C-14,
C-24, C-39, C-50, C-52

Slave 編號, A-28, C-5

SlavesLink, C-32

SQRT, B-150
 SR, B-124
 SSR[x][1].space, C-35
 ST, A-39, A-71, A-129, B-63, C-64, C-74,
 D-15
 ST 運算元 (IL), B-87
 ST 編輯器, A-73
 STACKINT, B-137
 Statement, D-15
 Step, D-15
 String, D-15
 Structured Text, D-15
 Style, A-68
 ST 中指定
 =, B-69
 Sub-program, D-15
 SYSTEM, B-120
 System clock rate (VxWorks), C-24

T

TAN, B-156
 Target, A-109, D-16
 Target cycle, D-16
 Technical note, D-16
 Test, D-16
 TextFile, A-108
 THEN, B-71
 Timer, D-16
 TMR, B-118
 To, A-110
 TO, B-75
 TOF, B-134
 Token (SFC), D-16

TON, B-133
 Toolbox, D-16
 Top level program, D-17
 TP, B-135
 Transition, D-17
 TRUNC, B-151
 TSK_FUNIT, C-25, C-29
 TSK_NBTCKSCHEd, C-25, C-29, C-37
 tst_main_ex, C-30
 TSTART, B-78
 TSTOP, B-79
 Type, D-17

U

ULongData, A-107
 UNTIL, B-74

V

Validity of a transition, D-17
 Variable, D-17
 VarList, A-107
 Virtual board, D-17

W

Wait, A-151
 WHILE, B-42, B-73
 WISAKER.EXE (NT), C-38

X

XOR, B-100
 XOR_MASK, B-107

二劃

二元, A-128

索引

二元選取器, B-168

十進位法, B-10

三劃

下降邊緣偵測接點, B-53

下載, A-118

上升邊緣偵測接點, B-53

上載, A-137, A-138

上載 (選項), A-138

上載原始碼, A-138

大於, B-110

大於或等於, B-111

子程式, A-18, B-3

小於, B-108

小於或等於, B-109

工具功能表, A-26

工具盒, D-16

四劃

不等於, B-113

中斷點, A-119, A-121, D-2

內容表, A-170

內部, D-9

分隔符號, B-63, D-14

化名, A-59

反正切, B-153

反正弦, B-152

反相, B-97

反相 (FBD), B-47

反相接點, B-52

反相連結, A-63, A-65

反相線圈, B-55

反餘弦, B-152

文件, A-14, A-25, A-170

文字編輯器, A-71

文字顯示, A-131

日記, A-21, D-6

比較, B-135

父程式, D-12

五劃

加, B-101

功能方塊, A-18, A-21, A-52, A-62, A-67,

A-79, A-83, A-158, B-5, B-44, C-72,

D-7

功能方塊圖, B-44, D-7

功能方塊樣例, C-73

可保留, C-110

失敗時間, A-28

巨集步驟, A-32, A-34, B-25, D-10

巨集步驟程式主體, B-26

布林, A-83, B-15, D-2

布林行爲, A-38, B-27, D-2

平方根, B-150

正切, B-156

正在行動的步驟, B-78

正在動作的步驟, B-21, B-36

正弦, B-155

正常樣式, A-69

正接點, B-53

正線圈, B-57

目前的結果 (IL), D-5

目前結果 (IL), B-84, B-85

目標硬體, A-102, D-16

目標硬體結構, C-3

目標硬體週期, D-16

目錄, A-182

六劃

交互查詢, A-25, A-114, D-4

全域, D-7

全域的, B-12

共用, D-3

共同資料, A-168

列印, A-14, A-25, A-81, A-170, A-172

同位元, A-29

向下計數, B-130

向上/向下計數, B-131

回存, A-159, A-167, A-169, A-176

回復, A-16

多工應用, C-18, C-34, C-44

字串, B-11, D-15

字串長度, B-176

字串相加, B-119

字典, A-21, A-73, A-78, A-115, A-164,

C-58, C-73, D-6

字型, A-173

存取 I/O 接點, B-122

存取層級, A-174

安裝, A-2

收斂, A-31, A-33, A-34, B-23

曲線, A-131, A-132, A-136

有效的轉移條件, B-36

行爲, B-26, B-32, D-1

行爲 (FC), D-1

七劃

位元向右旋轉, B-157

位元向右移位, B-159

位元向左旋轉, B-157

位元向左移位, B-158

位元欄, A-133

刪除 FBD, A-66

刪除 FC, A-47

刪除 LD, A-57

刪除 SFC, A-34

刪除子字串, B-171

刪除文字, A-71

刪除板子, A-93

刪除程式, A-23

刪除程式庫, A-157

刪除樣式, A-69

即時, A-24, A-119

即時錯誤, A-120

序列埠連結, A-29

技術註記, A-94, A-158, C-64, C-73,

D-16

技術摘要, C-56, C-58

改變焦點大小, A-133

更名程式庫, A-157

更新, A-24, A-102, A-118

步驟, A-30, A-36, A-121, B-20, D-15

步驟的動作狀態, D-1

決定, A-42, A-46, A-48, B-39

沒有動作的步驟, B-21

系統函數, C-112

八劃

其它的程式, A-75

函數, A-18, A-21, A-158, A-163, B-4

取代, A-36, A-47, A-58, A-66, A-71

取出子字串 (中間), B-175

索引

取出子字串 (右邊), B-179
取出子字串 (左邊), B-174
取消群組, A-134
呼叫函數 (ST), B-65
呼叫副程式 (ST), B-65
定義字, A-79, A-83, B-18, D-5
延遲計算 (IL), D-5
延遲時間, B-133, B-134
延遲執行 (IL), B-91
延遲運算 (IL), B-85
放大, A-50, A-58, A-68
板子, A-92, A-93
板子型態, A-93
板子參數, A-94, A-161
版本號碼, A-118
直接表示變數, A-95
直接接點, B-52
直接線圈, B-54
初始步驟, B-21, B-36, D-8
初始狀態, D-8
初始情形, B-21, B-36
表示式, D-6
返回, A-63
長條圖, A-132
非儲存, A-38
非儲存行為, B-29, D-11

九劃

信號, B-128
保留字, B-13
保護, A-14, A-97, A-157, A-174
保護層級, A-174
保護鎖, A-5

前置字 (prefix), B-10
型態, A-78, A-80, A-92, A-114, A-161,
B-9, D-17
宣告, A-21, A-78
指令, B-84, D-8
指令集, D-9
指令集語言, B-84
指定, B-96
指數, B-147
拷貝, A-159, A-167, A-176
拷貝磁碟機, A-168
括弧, B-64, B-85
架構, B-2
流程, A-45, B-39, B-41, B-43
流程圖, A-42, B-38, D-7
流程圖編輯器, A-42
界面, A-164
科學記號表示法, B-10
背景圖, A-131
計時器, A-83, B-16, D-16
計數, B-129
負接點, B-53
負線圈, B-58
重新編號, A-47
重置線圈, B-56
重編參考號碼, A-35

十劃

乘, B-103
乘冪計算, B-149
修改變數, A-82
修飾字 (IL), B-85, B-86, D-11
倉庫, A-40

- 原始碼, A-158
 時間, B-11
 時間單位, B-11
 案件, A-12, A-167, D-13
 案件分隔線, A-12
 案件文件, A-14, A-25, A-170
 案件列表, A-12, A-14
 案件描述, A-13, A-25
 案件群組, A-14
 案件管理, A-12
 格點, A-54, D-2
 真, A-83, B-9
 真實板, D-13
 真實板子, A-93
 真實模式, D-13
 矩陣, D-10
 脈衝, A-37
 脈衝行為, B-28, D-13
 脈衝計時, B-135
 草稿, A-146, A-148
 記憶體, A-2
 訊息, A-83, A-128, B-11, B-17, D-10
 訊息長度, B-176
 訊息連結, B-119
 訊號產生器, B-144
 閃爍, B-144
 除, B-104
 除錯, A-27
 除錯工作空間, A-27
 除錯器, A-116, A-141
 十一劃
 停止, A-117
 假, A-83, B-9
 剪下 FBD, A-66
 剪下 FC, A-47
 剪下 LD, A-57
 剪下 SFC, A-34
 剪下文字, A-71
 剪下變數, A-82
 副程式, A-18, B-4, B-31, B-35, B-40,
 B-47, D-15
 副程式回傳值, D-14
 動作, A-42, A-48, B-39, B-41
 區域, A-17, A-164, D-10, D-14
 區域的, B-12
 參考號碼, B-20, B-21, B-22, B-26
 參考號碼 (SFC), D-13
 參數, A-21, A-164
 參數 (C 函數), C-65, D-12
 參數 (I/O 板), D-12
 參數 (功能方塊), C-75
 基底, B-10
 執行一個週期, A-119
 執行期, A-24
 執行期錯誤, A-24, B-121, D-14
 執行週期, A-24, C-2
 執行順序, A-67
 密碼, A-14, A-97, A-157, A-174
 常數表示式, D-4
 常數表示法, B-9
 控制面板, A-116
 接點, A-51, A-62, A-94, A-95, A-96,
 A-161, B-51, D-4
 接點型態, A-57
 接點註解, A-94

索引

排列, A-82
啓動, A-118
敘述, B-63
敘述式, D-15
條件, B-39
條件 (對轉移條件而言), D-3
清除轉移條件, B-36, D-3
產生程式碼, A-24
移動 FBD, A-65
移動 FC, A-46
移動 SFC, A-35
移動板子, A-93
移動案件, A-12
移動焦點, A-133
移動程式, A-22
第二層, A-36, A-48
符號 (應用符號), C-6
符號表, A-184
終端模式, C-22
設定線圈, B-55
通訊, A-28, A-120, A-138, A-181, C-4,
 C-10, C-13, C-14, C-17, C-23, C-30,
 C-39, C-50, C-53
通訊程式邏輯號碼, C-14, C-15
連接 (LD), B-49
連接器 (FC), D-4
連結, A-28, A-63, A-65, A-66, A-120,
 A-138, A-181, B-39, B-41, B-43
連結 (FBD), B-45
連結 (SFC), B-22
連結器, A-45, B-41
連結點, A-176
頂層, A-17

頂層程式, D-17

十二劃

備份, A-16, A-159, A-167, A-169, A-176
備份檔, A-169
備份檔案路徑 (VxWorks), C-25, C-29
最大值, B-161
最小值, B-160
最佳化, A-103
創造矩陣, B-181
單步, A-24, A-119
單步模式, D-5
單頁, A-172
尋找, A-36, A-47, A-58, A-66, A-71, A-77
尋找子字串, B-172
循序, B-20
循序式功能圖, B-20
循序的, B-2
描述, A-12, A-25
插入 FBD, A-67
插入 FBD 元件, A-64
插入 FC 元件, A-44
插入子字串, B-173
插入空槽, A-93
插入接點, A-55
插入線圈, A-55
插入導軌, A-57
插入檔案, A-72
插入變數, A-37
減, B-102
測試, A-42, A-46, A-48, A-116, A-142,
 B-39, D-16
測試整數的奇或偶, B-166

焦點, A-131
 無效的轉移條件, B-36
 發散, A-31, A-33, A-34, B-23
 發散 (FBD), B-45
 硬碟, A-2
 程式, A-17, A-73, A-145, B-2, D-13
 程式列印, A-75
 程式的註解, A-21
 程式庫, A-16, A-23, A-93, A-94, A-114,
 A-156, A-167, C-55, D-10
 程式庫管理員, A-156, C-55, C-58, C-64,
 C-73
 程式管理員, A-17
 程式語法, A-73
 等於, B-112
 結束, A-17, B-38
 結束步驟, A-34, B-26, D-6
 結束區, D-6
 結構, A-17, A-22, B-36
 結構化文字, B-63, D-15
 絕對值, B-147
 虛擬板, D-17
 虛擬板 (NT 目標硬體模擬), C-50,
 C-53
 虛擬板 (在 NT 模擬), C-42
 虛擬板子, A-93, A-178
 註解, B-18, B-42, B-63, B-84, D-3
 註解 (SFC), B-20, B-22, D-3
 診斷, A-141
 貼上 FBD, A-66
 貼上 FC, A-47
 貼上 LD, A-57
 貼上 SFC, A-34

貼上文字, A-71
 貼上變數, A-82
 週期, B-2, B-7
 週期的, B-2, D-5
 週期時間, A-119, A-145, B-121, C-8,
 C-21, C-37, C-48, C-58, C-63, C-72,
 D-5
 週期控制結束 (VxWorks), C-25, C-29
 週期描繪器, A-145
 開始, A-17, A-118, B-38
 開始步驟, A-32, A-34, B-26, D-1
 開始區, D-1
 開啓案件, A-13
 開啓程式, A-21, A-115
 開啓檔案, B-184, B-185
 階梯圖, B-49, D-9
 順序, A-17
 順序式功能圖, D-14
 順序區, D-15

十三劃

亂數值, B-167
 微分, B-143
 新的功能方塊, A-20
 新的函數, A-20
 新的案件, A-13
 新的程式, A-20
 新的程式庫元件, A-156
 新導軌, A-54
 新變數, A-81
 置換子字串, B-177
 群組, A-3, A-14, A-134
 解除, A-119

索引

資料庫, A-143
資源, A-25, A-106
資源定義檔, A-106
跳回, A-53
跳至, A-36, A-47, A-71
跳躍, A-52, A-63, B-46, B-59
跳躍至步驟, A-32, B-22, D-9
運算 (IL), B-86
運算子 (IL), B-84, D-12
運算元 (IL), B-84, B-86, D-12
電力軌, A-51, A-52, A-61, B-49, D-12

十四劃

圖示, A-4, A-132
圖形, A-131, A-136
實數, A-83, B-10, D-13
對數, B-148
截去小數部分, B-151
網路位址, A-80, A-83, A-86, D-11
製作, A-24, A-101
語言, A-19, B-7

十五劃

寫入矩陣, B-183
寫入檔案, B-190, B-195
暫存器 (IL), B-84, D-14
樣式, A-134
樣例, A-79, A-83
標籤, A-63, B-46, B-59
標籤 (IL), B-84, D-9
槽位, A-93, A-96
模擬, A-27, A-146
模擬器, A-142, A-145, C-57, C-63, C-72

碼產生器, A-101
範圍, A-78, A-80, D-13
編輯案件描述, A-13
編譯, A-24, A-101, A-158, A-163
編譯訊息, A-105
編譯器選項, A-138
編譯選項, A-24, A-102
線上, A-27, A-116
線上變更, A-118, A-122
線圈, A-52, A-62, B-51, D-3
線圈型態, A-57
複製 FBD, A-66
複製 FC, A-47
複製 LD, A-57
複製 SFC, A-34
複製文字, A-71
複製程式, A-23
複製程式庫, A-157
複製變數, A-82
餘弦, B-154
餘數, B-163

十六劃

導軌, A-51, A-52, A-58
導軌的註解, A-54
導軌的標籤, A-55
導軌註解, A-58
導線, A-65
整數, A-83, B-10, D-9
整數位元遮罩 (互斥或), B-107
整數位元遮罩 (且), B-105
整數位元遮罩 (或), B-106
整數位元遮罩 (補數), B-108

整數值的堆疊, B-137
 樹狀結構, D-7
 歷史, A-26
 歷史紀錄, A-13
 積分, B-141
 輸入, A-87, A-92, A-115, A-142, A-145,
 A-161, B-7, D-8
 輸入功能方塊, A-23
 輸入函數, A-23
 輸出, A-87, A-92, A-115, A-142, A-161,
 B-7, D-12
 輸出功能方塊, A-23
 輸出函數, A-23
 輸出視窗, A-77
 選取 FC 元件, A-44
 選擇 FBD 元件, A-64
 選擇焦點, A-133
 遲滯, B-140
 錯誤, A-105
 錯誤訊息, A-77
 鮑率, A-29

十七劃

優先權, C-51
 優先權準位 (NT 目標硬體), C-42
 儲存變數集, A-127
 壓縮, A-168
 應用程式碼大小, C-53
 應用程式碼大小限制, C-9
 檔案：測試檔案結尾, B-187
 檢測, A-127, A-129, A-131
 檢測變數, A-127
 縮放 FC, A-46

點, A-99

十八劃

轉角, A-64
 轉移條件, A-30, A-36, B-22, D-17
 轉移條件有效性, D-17
 轉移條件的判斷條件, B-33, B-34
 轉換, A-99, D-4
 轉換 ASCII -> 字元, B-170
 轉換字元 -> ASCII, B-169
 轉換成布林型態, B-114
 轉換成計時器型態, B-118
 轉換成訊息型態, B-118
 轉換成實數型態, B-116
 轉換成整數型態, B-115
 轉換函數, A-165, C-55, C-57, D-4
 轉換表, A-98, A-100, D-4
 轉換條件, A-121
 鎖住, A-119, A-178
 鎖住 I/O, D-10
 離開鍵 (NT 目標硬體), C-49
 離開鍵 (目標硬體), C-8

十九劃

識別字, D-8
 邊緣, D-6
 關閉檔案, B-186
 關鍵字, D-9
 關鍵字 (IL), B-86
 類比, B-10, B-16, C-57, C-58, D-1

二十一劃

屬性, D-1

索引

二十二劃

權杖記號 (SFC), B-21, D-16

讀取陣列, B-182

讀取檔案, B-189, B-193

二十三劃

變更樣式, A-69

變數, A-21, A-37, A-56, A-63, A-67, A-72,

A-78, A-114, A-115, A-121, A-164,

A-176, B-12, B-44, D-17

變數名稱, B-12

變數的直接表示法, B-14

變數集合, A-127, A-129, A-134

驗證, A-24, A-101, A-163