HIWIN®
Motion Control and System Technology

史卡拉机器手臂控制器连线函式库

使用观念与说明

www.hiwin.tw

HIWIN®
Motion Control and System Technology

工业4.0 优质伙伴

INDUSTRIE 4.0 Best Partner

多轴机器人
Multi Axis Robot

取放作业/组装/整列与包装/半导体/
光电业/汽车工业/食品业

- 关节式机器手臂
- 并联式机器手臂
- 史卡拉机器手臂
- 晶圆机器人
- 电动夹爪
- 整合型电爪
- 旋转接头

单轴机器人
Single Axis Robot

高精密产业/半导体/
医疗自动化/FPD面板搬运

- KK, SK
- KS, KA
- KU, KE, KC

直驱马达回转工作台
Direct Drive Rotary Table

航太/医疗/汽车工业/工具机/产业机械

- RAB系列
- RAS系列
- RCV系列
- RCH系列

滚珠丝杆
Ballscrew

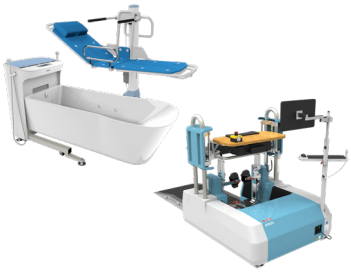
精密研磨/精密转造

- Super S 系列 (高 Dm-N 值/高速化)
- Super T 系列 (低噪音/低振动)
- 微小型研磨级
- E2 环保润滑模组
- R1 螺帽旋转式
- C1 节能温控丝杆
- RD 高DN节能重负荷
- 滚珠花键

直线导轨
Linear Guideway

精密机械/电子半导体/生技医疗

- 滚珠式—
  HG重负荷型, EG低组装, WE宽幅型,
  MG微小型, CG扭矩型
- 静音式—
  QH重负荷型, QE低组装型,
  QW宽幅型, QR滚柱型
- 其他—
  RG滚柱型, E2自润型, PG定位型,
  SE金属端盖型, RC强化型



医疗设备
Medical Equipment

医疗院所/复健中心/疗养中心

- 下肢康复训练机
- 沐浴水疗系统
- 内窥镜扶持机器手臂

特殊轴承
Bearing

工具机产业/机械手臂

- 交叉滚柱轴承
- 滚珠丝杆轴承
- 精密线性轴承
- 轴承座

AC伺服电机&驱动器
AC Servo Motor & Drive

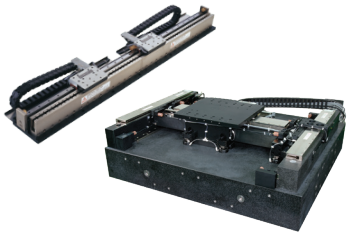
半导体设备/包装机/SMT机台/
食品业机台/LCD设备

- 驱动器—D1, D1-N, D2T
- 伺服电机—50W~2000W

动力刀座
Driven Tool Holders

各式刀塔

- VDI系统
  轴向动力刀座, 轴向偏心动力刀座,
  径向动力刀座, 径向缩头动力刀座, MT
- BMT系统
  DS, NM, GW, FO, MT, OM, MS



线性马达
Linear Motor

自动化搬运/AOI光学检测/
精密加工/电子半导体

- 铁心式直线电机
- 无铁心式直线电机
- 棒状直线电机
- 平面电机
- 空气轴承定位平台
- X-Y平台
- 龙门系统

转矩马达
Torque Motor
(Direct Drive Motor)

检测设备/工具机/机器人

- 旋转平台系列-TMS,TMY,TMN
- 水冷式系列-TMRW
- 高转速水冷系列-TMRI

版权声明：

本手册内容仅供授权合法持有本手册的客户与厂商学习、参考与使用。非经上银科技股份有限公司正式授权，不得以任何形式复制、抄袭、转载或透过网络散布其内容。

版权所有，翻印必究

© HIWIN Technologies Corp. All rights reserved

目录

1. 史卡拉机械手控制器联机函式库使用观念与说明

最大联机数：256

每个联机规格

最大档案清单数目：300

循环(Polling)命令队列个数：64 直接(Direct)命令队列个数：64

直接(Direct)命令与循环(Polling)命令 观念说明：

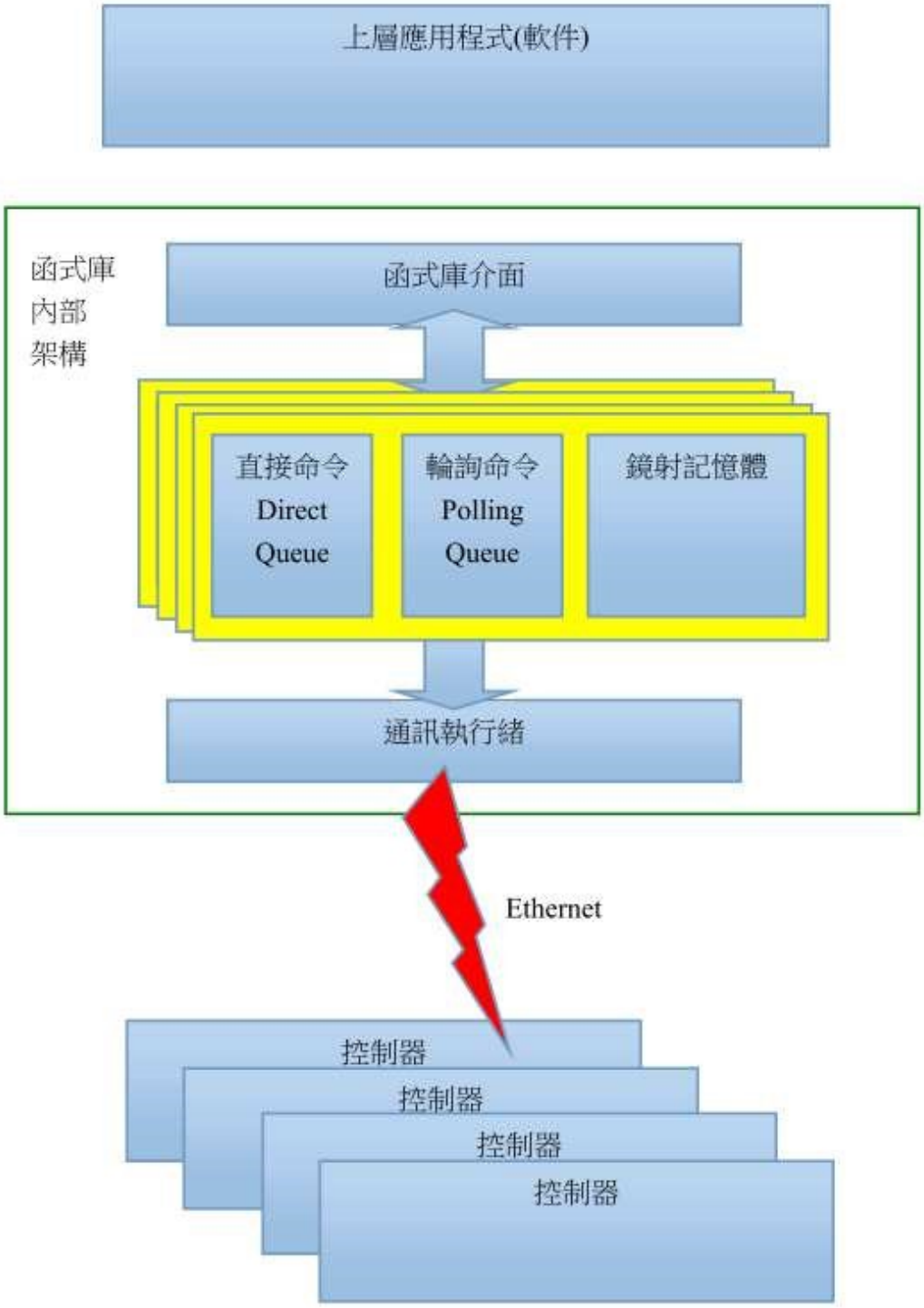
直接命令(Direct)：此类型的命令，会被存放到「直接命令队列 Direct Queue」中，当函式库成功执行该队列中的命令后，就会将该命令自队列中移除，也就是该命令只会被执行一次。

循环命列(Polling)：此类型的命令，会被存放到「循环命令队列 Polling Queue」中，函式库会重复循的执行该队列中的命令。直到主程序下「清除队列命令 scif_cmd_ClearAll」。循环命令又可分为两区部份，DEFAULT 及 POLLING。

命令的优先级

「直接(Direct)命令」的优先权是较「循环(Polling)命令」高的，因此只有在「直接命令」已完全被执行完毕后，「循环命令」才有机会被执行。

1.1. 函式库架构


上層應用程式(軟件)
函式庫
內部
架構
函式庫介面
直接命令
Direct
Queue
輪詢命令
Polling
Queue
鏡射記憶體
通訊執行緒
Ethernet
控制器
控制器
控制器
控制器

1.2. 函式库初始化说明

为了管控制造商只能联机到属于他的控制器，及一些其他的设定，函式库需先执行初始化动作，初始化的内容包含：

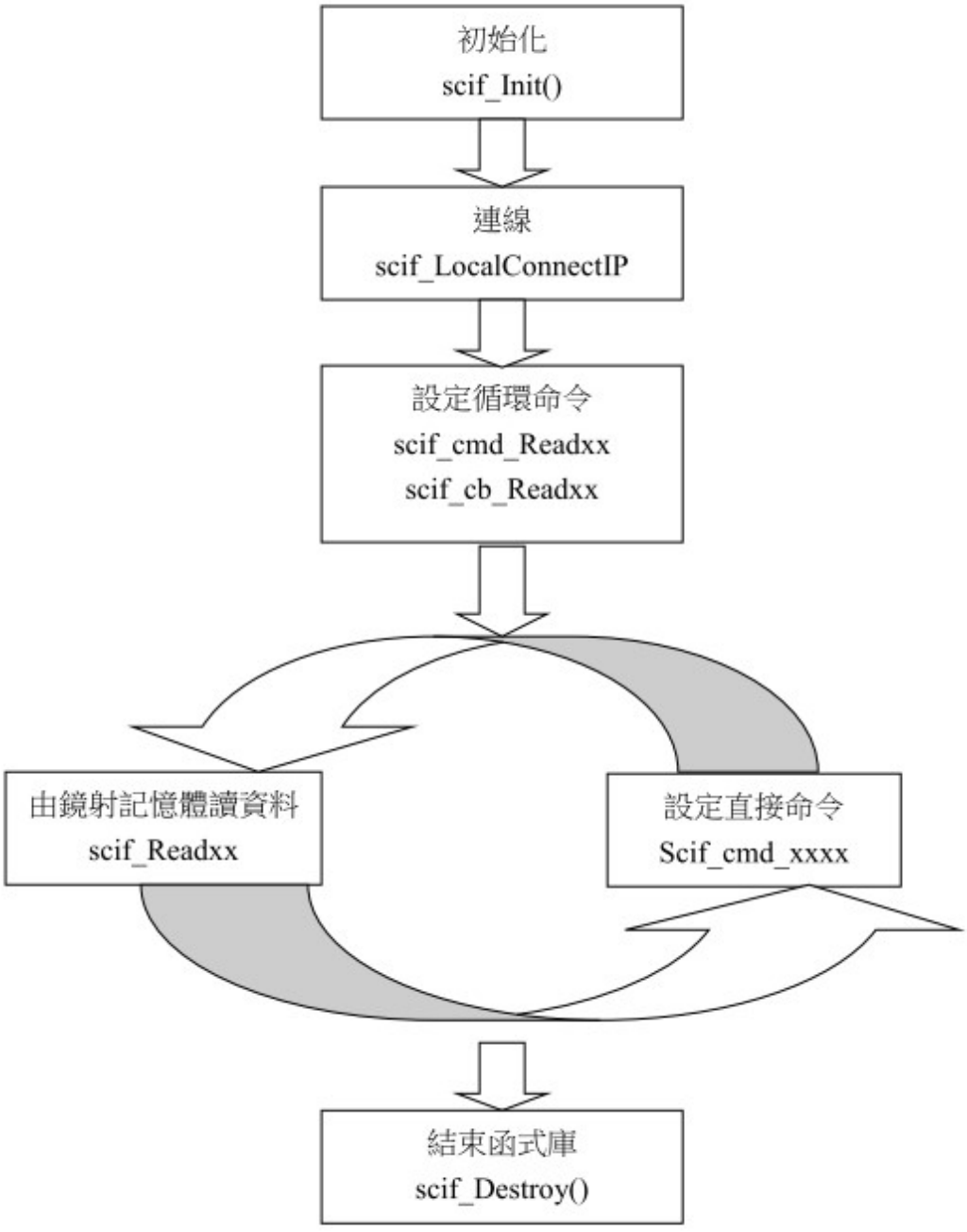
(1) 所使用的控制器联机 1~5：每台控制器最多可同时支持五个联机，软件联机时需设定其要连入的联机编号。

(2) 联机数目：联机的控制器数量，当用于监控使用，或以两台控制器同时组成系统时，会需要同时联机到多台控制器，此数值即代表要联机的控制器量。

(3) 各项资源使用数量：为了函式库的使用方便，会在 PC 端为每个联机建立一个镜射内存，用以存放自控制器读回的数据，此宣告要开启的内存小，当联机目量多时，更该特别注意此设定值，以免将 PC 的内存全部耗用完毕。

(4) 制造商识别密码字符串：用来确认制造商身份，以确认能够连入属于其的控制器，若密码字符串错误，初始化将会失败，函式库将无法正确运作。初始化成后，函式库才会建立通讯的线程。

1.3. 一般性的程序流程


初始化
scif_Init()
連線
scif_LocalConnectIP
設定循環命令
scif_cmd_Readxx
scif_cb_Readxx
由鏡射記憶體讀資料
scif_Readxx
設定直接命令
Scif_cmd_xxxx
結束函式庫
scif_Destroy()

2. 控制器联机方式

此章节会说明如何将我们的函式库初始化，并根据提供的参数作为函式库中的使用设定，在这章节介绍的函式为如何侦测局域网络内的控制器，包括其数量与控制器信息，也会说明与控制器的联机方式。

2.1. scif_Init

此函式为建立在本机端所需的镜射内存与线程，并根据提供的参数作为函式库中的使用设定。

语法

int scif_Init (DLL_USE_SETTING *pUseSetting, int MakerID, char *pEncString);

参数

pUseSetting：

函式库的使用设定，可在 scif_define.h 檔中找到其 struct 的内容。其中的 SoftwareType 代表软件种类，TalkInfoNum 代表联机数目。

MakerID：制造商编号，此参数会由原厂宝元数控提供。

pEncString：

加密字符串，包含信息有 MakerID、局域网络侦测功能、是否使用镜射内存、可否数据写入、是否有文件传输功能，此参数亦会由原厂宝元数控提供。

回传值

0：初始化失败。

10：初始化成功、但解密功能字符串失败。

100：成功。

范例

```
DLL_USE_SETTING DllSetting;
DllSetting.SoftwareType = 3;      //软件种类
DllSetting.TalkInfoNum = 5;       //联机数目
```

```
//以下参数的设定代表在 PC 端所宣告镜射内存所能读取各参数值地址的
//范围，需要特别注意的为 MemSizeR 的大小，如果将其设定成与控制器预设
//R 值(6000000)大小相同的话，则会在 PC 端使用约 25MB 的大小，再乘以连
//线数目，在本范例中则会使用约 125MB，故我们在本范例中特地将
//MemSizeR 设为 10000，而非默认值，但设定为 10000 后，则 R 地址值超过
//10000 以上的数值，不会储存在镜射内存中，请用户根据自己所需要的
//各参数地址范围来设定以下的 MemSize 值。


  DllSetting.MemSizeI    =    4096;
  DllSetting.MemSizeO    =    4096;
  DllSetting.MemSizeC    =    4096;
  DllSetting.MemSizeS    =    4096;
  DllSetting.MemSizeA    =    4096;
  DllSetting.MemSizeTT   =    256;
  DllSetting.MemSizeCT   =    256;
  DllSetting.MemSizeR    =    10000;  //控制器端预设大小为 6000000
  DllSetting.MemSizeTS   =    256;
  DllSetting.MemSizeTV   =    256;
  DllSetting.MemSizeCS   =    256;
  DllSetting.MemSizeCV   =    256;
  DllSetting.MemSizeF    =    100000;

int makerid, Status;
char pencstring[64];
Status = scif_Init(&DllSetting, makerid, pencstring);
```

2.2. scif_Destroy

此函式为终结建立的内存与线程。

语法

void scif_Destroy();

参数

无。

回传值

无。

范例

scif_Destroy();

2.3. scif_LocalDetectControllers

此函式会自动侦测局域网络内有多 控制器，并读取其控制器信息，并在函式库内依序建立每个控制器的数据索引，若无呼叫此函式，则呼叫 scif_LocalReadControllerCount 与 scif_LocalReadController 两函式时，不会有正确的回传值。

语法

```
int scif_LocalDetectControllers();
```

参数

无。

回传值

局域网络内侦测到的控制器数量。

范例

```
int Count;
Count = scif_LocalDetectControllers();
```

2.4. scif_LocalReadControllerCount

此函式为读取在函式库内记录的控制器数据笔数，在呼叫此函式前，必须先呼叫 scif_LocalDetectControllers 函式，才会有正确的回传值。

语法

```
int scif_LocalReadControllerCount();
```

参数

无。

回传值

函式库中记录侦测到的控制器数量。

范例

```
int Count;
Count = scif_LocalReadControllerCount();
```

2.5. scif_LocalReadController

此函式会根据传入的控制器数据索引，将每个控制器的数据存放入函式库的控制器数据结构中，但在呼叫此函式前，必须先呼叫 scif_LocalDetectControllers 函式，才会有正确的回传值。

语法

```
int scif_LocalReadController(unsigned short Index,
LOCAL_CONTROLLER_INFO *Info);
```

参数

Index：

函式库内记录的区域控制器数据索引。

Info：

函式库的使用设定，可在 scif_define.h 檔中找到其 struct 的内容。内容中会有控制器 IP、名称等信息。

```
typedef struct tag_LOCAL_CONTROLLER_INFO1
{
        unsigned int    IPLong;
        char            IP[16];
        char            Name[16];
}LOCAL_CONTROLLER_INFO;
```

回传值

0：失败。

1：成功。

范例

```
//在 LOCAL_CONTROLLER_INFO 的 struct 中，会存放控制器的 IP 和 Name，
//其中 Name 为使用者在设定控制器时，对控制器命名的名称，使用者可经由
//控制器的名称搭配控制器 IP 判断此联机的控制器是否为其所想联机的对象。

LOCAL_CONTROLLER_INFO Info;
int Status;
unsigned short controllerindex;
Status = scif_LocalReadController(controllerindex, &Info);
```

2.6. scif_ConnectLocalList

此函式为与函式库内记录的控制器数据索引进行联机设定，在呼叫此函式前，必须呼叫过 scif_LocalDetectControllers 函式，才会有正确的控制器数据索引。执行此函式成功只代表联机设定成功，有无真正建立起联机，必须呼叫 scif_GetTalkMsg 函式来检查联机状态。

语法

int scif_ConnectLocalList(char ServerIdx, unsigned short Index);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

Index：

函式库内记录的区域控制器数据索引。

回传值

0：设定指令失败。

1：设定指令被接受。

范例

```
char serverindex;  unsigned
short controllerindex;
int Status;
Status = scif_ ConnectLocalList (serverindex, controllerindex);
```

2.7. scif_LocalConnectIP

此函式为直接输入控制器 IP 进行联机设定。执行此函式成功只代表联机设定成功，有无真正建立起联机，必须呼叫 scif_GetTalkMsg 函式来检查联机状态。

语法

int scif_LocalConnectIP(char ServerIdx, char *IP);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

IP：

欲联机的控制器 IP。

回传值

0：设定指令失败。

1：设定指令被接受。

范例

```
char serverindex;
char ip[32];
int Status;
Status = scif_ LocalConnectIP(serverindex, ip);
```

2.8. scif_Disconnect

呼叫此函式中断与控制器的联机。

语法

int scif_Disconnect(char ServerIdx);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

回传值

0：设定指令失败。

1：设定指令被接受。

范例

```
char serverindex;
int Status;
Status = scif_ Disconnect(serverindex);
```

2.9. scif_GetTalkMsg

呼叫此函式可取得联机通讯的信息。

语法

unsigned int scif_GetTalkMsg(char ServerIdx, char id);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

id：

scif_define.h 文件中有可填入的 id 信息，如填入 SCIF_CONNECT_STATE 为取得联机的状态。

回传值

信息内容；scif_define.h 文件中有回传信息内容代表的意义。

范例

```
char serverindex;
int Status;
Status = scif_GetTalkMsg(serverindex, SCIF_CONNECT_STATE);
```

2.10. 联机状态补充

可使用同一个 IP 对控制器建立多个联机，但前提是必须使用不同 SoftwareType(软件种类)。

联机时可能的状态有：

```
#define SC_CONN_STATE_DISCONNECT        0    //联机关闭
#define SC_CONN_STATE_CONNECTING        1    //联机中
#define SC_CONN_STATE_FAIL              2    //联机失败
#define SC_CONN_STATE_OK                3    //联机正常
#define SC_CONN_STATE_NORESPONSE        4    //联机无回应
```

- 刚启动函式库或要求中断联机后，状态为 SC_CONN_STATE_DISCONNECT。
- 要求联机后，状态变为 SC_CONN_STATE_CONNECTING。
- 若联机失败，状态变为 SC_CONN_STATE_FAIL，之后变成 SC_CONN_STATE_NORESPONSE。
- 若联机成功，状态变为 SC_CONN_STATE_OK。
- 若断线或控制器关机，状态变为 SC_CONN_STATE_NORESPONSE。
- 在非 SC_CONN_STATE_DISCONNECT 状态下，会自动尝试重新联机。

与控制器的联机设定完成后，必须等待到联机状态回传为联机正常，才可以确定与控制器的联机真的成功。

3. 通讯数据设定与读取方式

此章节会说明如何建立与控制器间的通讯命令，并可根据使用者的需求分为连续和离散两种通讯方式，连续的通信设置意指读取连续区间地址的设定(如地址区间为 0~10)，而离散的通讯方式则可一次输入不相连地址的设定(如将地址设定为 1、4、13)。并提供镜射内存和指针结构两种方式来读取所需的数据。在处理通讯数据的方面，若所要读取值的地址太过分散、设定读取函式时所读取的数量太 ，造成呼叫了多次读取函式向控制器读值，这种情况下会造成大量的通讯封包，且在轮询数据时会较无效益，故我们提供了封包组合函式，有效的减命令封包数量。在读取数据方面，也提供了函式来组合读取回来的数据，提供更好的数据处理效益。

3.1. 通讯命令型态说明

SC_DEFAULT_CMD：

用于同时要监看多个联机控制器的数据时，此种命令会
SC_POLLING_CMD 一同被轮流执行。

SC_POLLING_CMD：
用以持续同步更新控制器与 PC 端的数据。

SC_DIRECT_CMD：
此种命令执行过一次后即被删除，且会被优先处理。

通讯命令为通讯数据设定中的参数，当通讯命令被设为 SC_DEFAULT_CMD 和 SC_POLLING_CMD 时，所设定要被读取的地址数据，将会持续被更新，若通讯命令设为 SC_DIRECT_CMD，则要读写的地址数据只会被执行一次，在函式库中，用来写入控制器中数据的函式，默认的通讯命令皆为 SC_DIRECT_CMD。

3.2. scif_cmd_ReadI,scif_cmd_ReadO,scif_cmd_ReadC, scif_cmd_ReadS,scif_cmd_ReadA,scif_cmd_ReadTMR, scif_cmd_ReadCNT

这些函式皆是用来作为连续数据读取的设定，根据数据存放类型的不同，分为不同的函式去读取，但所需输入参数的意义皆同。以函式 scif_cmd_ReadI 为例来说明。

语法

int scif_cmd_ReadI(char type, char ServerIdx, unsigned int addr, unsigned int num);

参数

type：

命令型态 SC_DEFAULT_CMD，SC_POLLING_CMD，

SC_DEFAULT_CMD。

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

起始地址。

num：

读取数量，num 的最大值为 MAX_BIT_NUM(scif_define.h 档中定义)。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。(若该设定会被重新组合，回传值为 1，之后无法用此指标判断命令是否已被执行。)

范例

```
char serverindex;
unsigned int addr, num;
int pTran;
pTran = scif_cmd_ReadI(SC_POLLING_CMD, serverindex, addr, num);
```

3.3. scif_cmd_ReadR, scif_cmd_ReadTMRV, scif_cmd_ReadTMRS, scif_cmd_ReadCNTV, scif_cmd_ReadCNTS

这些函式皆是用来作为连续数据读取的设定，根据数据存放类型的不同，分为不同的函式去读取，但所需输入参数的意义皆相同。以函式 scif_cmd_ReadR 为例来说明。

语法

int scif_cmd_ReadR(char type, char ServerIdx, unsigned int addr, unsigned int num);

参数

type：

命令型态 SC_DEFAULT_CMD，SC_POLLING_CMD，

SC_DEFAULT_CMD。

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

起始地址。

num：

读取数量，num 的最大值为 MAX_INT_NUM(scif_define.h 档中定义)。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。(若该设定会被重新组合，回传值为 1，之后无法用此指标判断命令是否已被执行。)

范例

```
char serverindex;
unsigned int addr, num;
int pTran;
pTran = scif_cmd_ReadR(SC_POLLING_CMD, serverindex, addr, num);
```

3.4. scif_cmd_ReadF, scif_cmd_ReadP

这些函式皆是用来作为连续数据读取的设定，根据数据存放类型的不同，分为不同的函式去读取，但所需输入参数的意义皆相同。以函式 scif_cmd_ReadF 为例来说明。

语法

int scif_cmd_ReadF(char type, char ServerIdx, unsigned int addr, unsigned int num);

参数

type：

命令型态 SC_DEFAULT_CMD，SC_POLLING_CMD，

SC_DEFAULT_CMD。

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

起始地址。

num：

读取数量，num 的最大值为 MAX_FIX_NUM(scif_define.h 档中定义)。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。(若该设定会被重新组合，回传值为 1，之后无法用此指标判断命令是否已被执行。)

范例

```
char serverindex;
unsigned int addr, num;
int pTran;
pTran = scif_cmd_ReadF(SC_POLLING_CMD, serverindex, addr, num);
```

3.5. scif_cb_ReadI,scif_cb_ReadO,scif_cb_ReadC, scif_cb_ReadS,scif_cb_ReadA,scif_cb_ReadTMR, scif_cb_ReadCNT

这些函式皆是用来作为离散数据读取的设定，根据数据存放类型的不同，分为不同的函式去读取，但所需输入参数的意义相同。以函式 scif_cb_ReadI 为例来说明。

语法

int scif_cb_ReadI(char type, char ServerIdx, unsigned int num, unsigned int *addr);

参数

type：

命令型态 SC_DEFAULT_CMD，SC_POLLING_CMD，

SC_DEFAULT_CMD。

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

num：

读取数量，num 的最大值为 MAX_CB_BIT_NUM(scif_define.h 档中定义)。

addr：

要读取的地址数组指针，在数组中填入所要读取数据的地址。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。(若该设定会被重新组合，回传值为 1，之后无法用此指标判断命令是否已被执行。)

范例

```
char serverindex;
unsigned int num; unsigned int addr[32];
int pTran;
pTran = scif_cb_ReadI(SC_POLLING_CMD, serverindex, num, addr);
```

3.6. scif_cb_ReadR,scif_cb_ReadTMRV,scif_cb_ReadTMRS, scif_cb_ReadCNTV, scif_cb_ReadCNTS

这些函式皆是用来作为离散数据读取的设定，根据数据存放类型的不同，分为不同的函式去读取，但所需输入参数的意义皆同。以函式 scif_cb_ReadR 为例来说明。

语法

```
int scif_cb_ReadR(char type, char ServerIdx, unsigned int num, unsigned int *addr);
```

参数

type：

命令型态 SC_DEFAULT_CMD，SC_POLLING_CMD，

SC_DEFAULT_CMD。

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

num：

读取数量，num 的最大值为 MAX_CB_INT_NUM(scif_define.h 档中定义)。

addr：

要读取的地址数组指针，在数组中填入所要读取数据的地址。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。(若该设定会被重新组合，回传值为 1，之后无法用此指标判断命令是否已被执行。)

范例

```
char serverindex;
unsigned int num; unsigned int addr[32];
int pTran;
pTran = scif_cb_ReadR(SC_POLLING_CMD, serverindex, num, addr);
```

3.7. scif_cb_ReadF, scif_cb_ReadP

这些函式皆是用来作为离散数据读取的设定，根据数据存放类型的不同，分为不同的函式去读取，但所需输入参数的意义皆同。以函式 scif_cb_ReadF 为例来说明。

语法

int scif_cb_ReadF(char type, char ServerIdx, unsigned int num, unsigned int *addr);

参数

type：

命令型态 SC_DEFAULT_CMD，SC_POLLING_CMD，

SC_DEFAULT_CMD。

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

num：

读取数量，num 的最大值为 MAX_CB_FIX_NUM(scif_define.h 档中定义)。

addr：

要读取的地址数组指针，在数组中填入所要读取数据的地址。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。(若该设定会被重新组合，回传值为 1，之后无法用此指标判断命令是否已被执行。)

范例

```
char serverindex;
unsigned int num;
unsigned int addr[32];
int pTran;
pTran = scif_cb_ReadF(SC_POLLING_CMD, serverindex, num, addr);
```

3.8. scif_GetTranState

在呼叫完连续或离散通讯数据读取设定的函式后，如果通讯封包没有设定被重新组合，且欲以指针结构方式来读取数据，则必须呼叫此函式取得通讯命令状态，才能得知通讯命令有无被正确执行。

语法

unsigned char scif_GetTranState(int pTran);

参数

pTran：

连续或离散通讯数据命令设定时函式回传的通讯封包指针。

回传值

命令状态。

```
#define SC_TRANSACTION_PENDING          0       //等待处理中
#define SC_TRANSACTION_PORCESSING       1       //处理中
#define SC_TRANSACTION_FINISH           2       //完成
#define SC_TRANSACTION_INVALID          3       //无效的索引
```

范例

```
unsigned char command_status;
int pTran;
command_status = scif_GetTranState(pTran);
```

3.9. scif_GetDefaultQueueDataPointer

若欲以指针结构方式来读取数据，则必须呼叫此函式来取得 Default Queue 通讯命令数据指针，但在呼叫此函式前，必须呼叫 scif_GetTranState 函式确认通讯命令被正确执行。

语法

SC_DATA* scif_GetDefaultQueueDataPointer(char ServerIdx, unsigned char TranIdx);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

TranIdx：

Default 命令在设定时的顺序，由 0 开始。

回传值

数据结构指针，在 scif_define.h 档中定义。

```
typedef union tag_SC_DATA
{
    unsigned char     Bytes[MAX_BIT_NUM];        //bit 或 byte 资料
    unsigned short    Words[MAX_WORD_NUM];       //word 资料
    unsigned int      Ints[MAX_INT_NUM];         //整数
    double            Fixs[MAX_FIX_NUM];         //double
    SC_POINTER        Ptrs[MAX_PTR_NUM];         //指标的值
}SC_DATA;
```

范例

```
char serverindex;
unsigned char tranindex;
SC_DATA*sc_data;
sc_data = scif_GetDefaultQueueDataPointer(serverindex, tranindex);
```

3.10. scif_GetDataPointerByTranPointer

若欲以指针结构方式来读取数据，则必须呼叫此函式由交易封包指针取得通讯命令数据指针，但在呼叫此函式前，必须呼叫 scif_GetTranState 函式确认通讯命令被正确执行，并执行完 scif_GetDefaultQueueDataPointer 函式。

语法

SC_DATA* scif_GetDataPointerByTranPointer(int TranPointer);

参数

TranPointer：

连续或离散通讯数据命令设定时函式回传的通讯封包指针。

回传值

数据结构指针，在 scif_define.h 档中定义。

```
typedef union tag_SC_DATA
{
   unsigned char      Bytes[MAX_BIT_NUM];      //bit 或 byte 资料
   unsigned short     Words[MAX_WORD_NUM];     //word 资料
   unsigned int       Ints[MAX_INT_NUM];       //整数
   double             Fixs[MAX_FIX_NUM];       //double
   SC_POINTER         Ptrs[MAX_PTR_NUM];       //指标的值
 }SC_DATA;
```

范例

```
SC_DATA *sc_data;
int pTran;
sc_data = scif_GetDataPointerByTranPointer(pTran);
```

补充说明

每笔通讯命令的数据是一个联集的数据内容，依据命令的数据型态不同，存放在不同的成员中(但其实是同一份内存)。

I,O,C,S,A,TMR,CNT → Bytes。

R,TMRV,TMRS,CNTV,CNTS → Ints。

F → Fixs。

P → Ptrs。

3.11. scif_SetMirror

在呼叫完连续或离散通讯数据读取设定的函式后，若欲以镜射内存方式来读取数据，则必须先呼叫此函式来做镜射内存的设定。在考虑会联机多台控制器的情况下，透过设定的方式，让每个控制器对应到各自的镜射内存，请注意在呼叫读取镜射内存的函式，如 scif_ReadXX 之前，务必要呼叫此函式，经由此函式的 ServerIdx 参数，来对应所要读取该联机控制器的镜射内存。

语法

int scif_SetMirror(char ServerIdx);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

回传值

0：设定失败。

1：设定成功。

范例

```
int Status;
char serverindex;
Status = scif_SetMirror(serverindex);
```

3.12. scif_ReadI, scif_ReadO, scif_ReadC, scif_ReadS, scif_ReadA, scif_ReadTMR, scif_ReadCNT

这些函式皆是用来读取镜射内存数据的值，根据数据存放类型的不同，分为不同的函式去读取，但所需输入参数的意义皆相同。以函式 scif_ReadI 为例来说明。因考虑会有联机到多个控制器的情况下，在呼叫这些函式读取数据前，必须先呼叫 scif_SetMirror 函式，来选择所要读取控制器的镜射内存。

语法

char scif_ReadI(unsigned int addr);

参数

addr：

数据地址。

回传值

数据内容。

范例

```
char data;
unsigned int addr;
data = char scif_ReadI(addr );
```

3.13. scif_ReadR, scif_ReadTMRV, scif_ReadTMRS, scif_ReadCNTV, scif_ReadCNTS

这些函式皆是用来读取镜射内存数据的值，根据数据存放类型的不同，分为不同的函式去读取，但所需输入参数的意义皆相同。以函式 scif_ReadR 为例来说明。因考虑会有联机到多个控制器的情况下，在呼叫这些函式读取数据前，必须先呼叫 scif_SetMirror 函式，来选择所要读取控制器的镜射内存。

语法

unsigned int scif_ReadR(unsigned int addr);

参数

addr：

数据地址。

回传值

数据内容。

范例

```
unsigned int data, addr;
data = char scif_ ReadR (addr );
```

3.14. scif_ReadF

此函式是用来读取镜射内存数据的值，因考虑会有联机到多个控制器的情况下，在呼叫此函式读取数据前，必须先呼叫 scif_SetMirror 函式，来选择所要读取控制器的镜射内存。

语法

double scif_ReadF(unsigned int addr);

参数

addr：

数据地址。

回传值

数据内容。

范例

unsigned int addr;
double data;
data = char scif_ ReadR (addr);

3.15. scif_ReadRString

此函式是由镜射内存中读取字符串，因考虑会有联机到多个控制器的情况下，在呼叫此函式读取数据前，必须先呼叫 scif_SetMirror 函式，来选择所要读取控制器的镜射内存。

语法

unsigned int scif_ReadRString(unsigned int addr, unsigned int BufSize, char *Buf);

参数

addr：

数据地址。

BufSize：

要读取的数量 (Bytes)。

Buf：

要回传的字符串内容。

回传值

要回传的字符串内容的数量(Bytes)。

范例

```
unsigned int num, addr, bufsize;
char buf[32];
num = scif_ReadRString( addr, bufsize, buf);
```

3.16. scif_cmd_WriteRString

此函式是写入字符串到控制器中的 R 值。

语法

int scif_cmd_WriteRString(char ServerIdx, unsigned int addr, unsigned int BufSize, char *Buf);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

addr：

要写入的数据地址。

BufSize：

要写入的数量 (Bytes)。

Buf：

要写入的字符串内容。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status;
char serverindex;
unsigned int addr, bufsize;
char buf[32];
Status = scif_cmd_WriteRString(serverindex, addr, bufsize, buf);
```

3.17.scif_cmd_WriteI,scif_cmd_WriteO,scif_cmd_WriteC, scif_ cmd_WriteS, scif_cmd_WriteA, scif_cmd_WriteTMR, scif_ cmd _WriteCNT

这些函式皆是用来作为单笔资料的写入，根据资料存放类型的不同，分为不同的函式去写值，但所需输入参数的意义皆相同。以函式 scif_cmd_WriteI 为例来说明。

语法

int scif_cmd_WriteI(char ServerIdx, unsigned int addr, char val);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

要写入的地址。

val：

要写入的值。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status;  char
serverindex;
unsigned int addr;
char val;
Status = scif_cmd_WriteI(serverindex, addr, val);
```

3.18. scif_cmd_WriteR, scif_cmd_WriteTMRV, scif_cmd_WriteTMRS, scif_cmd_WriteCNTV, scif_cmd_WriteCNTS

这些函式皆是用来作为单笔资料的写入，根据资料存放类型的不同，分为不同的函式去写值，但所需输入参数的意义皆相同。以函式 scif_cmd_WriteR 为例来说明。

语法

int scif_cmd_WriteR(char ServerIdx, unsigned int addr, unsigned int val);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

要写入的地址。

val：

要写入的值。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status;  char
serverindex;
unsigned int addr;
unsigned int val;
Status = scif_cmd_WriteI(serverindex, addr, val);
```

3.19. scif_cmd_WriteF

此函式是用来作为单笔资料的写入。

语法

int scif_cmd_WriteF(char ServerIdx, unsigned int addr, double val);

参数

ServerIdx:

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr:

要写入的地址。

val:

要写入的值。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status;
char serverindex;
unsigned int addr;
double val;
Status = scif_cmd_WriteI(serverindex, addr, val);
```

3.20. scif_cmd_WriteMultiI, scif_cmd_WriteMultiO, scif_cmd_WriteMultiC, scif_cmd_WriteMultiS, scif_cmd_WriteMultiA, scif_cmd_WriteMultiTMR, scif_cmd_WriteMultiCNT

这些函式皆是用来作为连续资料的写入，根据资料存放类型的不同，分为不同的函式去写值，但所需输入参数的意义皆同。以函式 scif_cmd_WriteMultiI 为例来说明。

语法

```
int scif_cmd_WriteMultiI(char ServerIdx, unsigned int addr, unsigned int num,
char *data);
```

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

要写入的地址。

num：

要写入的数量，最大值为 MAX_BIT_NUM(scif_define.h 档中定义)。

data：

要写入值的数组指针。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status; char serverindex, data[32];
unsigned int addr, num;
Status = scif_cmd_WriteMultiI(serverindex, addr, num, data);
```

3.21. scif_cmd_WriteMultiR, scif_cmd_WriteMultiTMRV, scif_cmd_WriteMultiCNTV, scif_cmd_WriteMultiCNTS, scif_cmd_WriteMultiTMRS

这些函式皆是用来作为连续资料的写入，根据资料存放类型的不同，分为不同的函式去写值，但所需输入参数的意义皆同。以函 scif_cmd_WriteMultiR 为例来说明。

语法

int scif_cmd_WriteMultiR(char ServerIdx, unsigned int addr, unsigned int num, unsigned int *data);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

要写入的地址。

num：

要写入的数量，最大值为 MAX_INT_NUM (scif_define.h 档中定义)。

data：

要写入值的数组指针。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

int Status;

char serverindex;

unsigned int addr, num, data[32]; Status = scif_cmd_WriteMultiI(serverindex, addr, num, data);

3.22. scif_cmd_WriteMultiF

此函式是用来作为连续资料的写入。

语法

int scif_cmd_WriteMultiF(char ServerIdx, unsigned int addr, unsigned int num, double *data);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

要写入的地址。

num：

要写入的数量，最大值为 MAX_INT_NUM (scif_define.h 档中定义)。

data：

要写入值的数组指针。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status;
char serverindex;
unsigned int addr, num;
double data[32];
Status = scif_cmd_WriteMultiF(serverindex, addr, num, data);
```

3.23. scif_cb_WriteI, scif_cb_WriteO, scif_cb_WriteC, scif_cb_WriteS,scif_cb_WriteA, scif_cb_WriteTMR, scif_cb_WriteCNT

这些函式皆是用来作为离散地址数据的写入，根据资料存放类型的不同，分为不同的函式去写值，但所需输入参数的意义皆同。以函式 scif_cb_WriteI 为例来说明。

语法

int scif_cb_WriteI(char ServerIdx, unsigned int num, unsigned int *addr, char *data);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

num：

要写入的数量，最大值为 MAX_CB_BIT_NUM (scif_define.h 档中定义)。

addr：

要写入的地址数组指针。

data：

要写入值的数组指针。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

int Status;

char serverindex, data[32];

unsigned int addr[32], num; Status = scif_cb_WriteI(serverindex, num, addr, data);

3.24.scif_cb_WriteR,scif_cb_WriteTMRV,scif_cb_WriteT MRS, scif_cb_WriteCNTV, scif_cb_WriteCNTS

这些函式皆是用来作为离散地址数据的写入，根据资料存放类型的不同，分为不同的函式去写值，但所需输入参数的意义皆同。以函式 scif_cb_WriteR 为例来说明。

语法

```
int scif_cb_WriteR(char ServerIdx, unsigned int num, unsigned int *addr,
unsigned int *data);
```

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

num：

要写入的数量，最大值为 MAX_CB_INT_NUM(scif_define.h 档中定义)。

addr：

要写入的地址数组指针。

data：

要写入值的数组指针。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

int Status;

char serverindex;

unsigned int addr[32], num, data[32]; Status = scif_cb_WriteR(serverindex, num, addr, data);

3.25. scif_cb_WriteF

此函式是用来作为离散地址数据的写入。

语法

int scif_cb_WriteF(char ServerIdx, unsigned int num, unsigned int *addr, double
*data);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

num：

要写入的数量，最大值为 MAX_CB_FIX_NUM(scif_define.h 档中定义)。

addr：

要写入的地址数组指针。

data：

要写入值的数组指针。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status;
char serverindex;
unsigned int addr[32], num;
double data[32];
Status = scif_cb_WriteF(serverindex, num, addr, data);
```

3.26. scif_cmd_WriteRBit

此函式是用来作为 R 值单个 bit 地址数据的写入。

语法

int scif_cmd_WriteRBit(char ServerIdx, unsigned int addr, unsigned char BitIdx, unsigned char BitValue);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

addr：

要写入 R 值的地址。

BitIdx：

要写入 R 值的位地址。

BitValue：

设定值，0 或 1。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status;
char serverindex;
unsigned int addr;
unsigned char bitindex, bitvalue;
Status = scif_cmd_WriteRBit(serverindex, addr, bitindex, bitvalue);
```

3.27. scif_cb_WriteRBit

此函式是用来作为多个 R 值 bit 地址数据的写入。

语法

int scif_cb_WriteRBit(char ServerIdx, unsigned int num, unsigned int *addr, unsigned char *BitIdx, unsigned char *BitValue);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。-1 代表所有联机都要套用此设定。

num：

要写入 R 值的数量。

addr：

存放 R 的地址的数组指针。

BitIdx：

存放位地址的数组指针。

BitValue：

存放设定值的数组指针，设定值为 0 或 1。

回传值

通讯封包的指针，若回传值为 0，代表命令设定失败。

范例

```
int Status;
char serverindex;
unsigned int num, addr[32];
unsigned char bitindex[32], bitvalue[32];
Status = scif_cb_WriteRBit(serverindex, num, addr, bitindex, bitvalue);
```

3.28. scif_StartCombineSet

此函式是用来作为同步数据的设定之用，呼叫此函式来设定自动组合旗标，呼叫此函式后，需再呼叫 scif_FinishCombineSet 函式，才能完成封包组合的设定。3.30 节有对封包组合作进一步的补充说明。

语法

void scif_StartCombineSet(char ServerIdx);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

回传值

无。

范例

```
char serverindex;
scif_StartCombineSet(serverindex);
```

3.29. scif_FinishCombineSet

此函式为完成自动组合设定并开始产生组合封包，呼叫此函式前必须先呼叫 scif_StartCombineSet。2.30 章节有对封包组合作进一步的补充说明。

语法

void scif_FinishCombineSet(char ServerIdx);

参数

ServerIdx:

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时，struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

回传值

无。

范例

```
char serverindex;
scif_FinishCombineSet(serverindex);
```

3.30. 封包组合设定补充说明

呼叫 scif_StartComboin 之后，未呼叫 scif_FinishComboin 之前，若 scif_cmd_Readxxx 时传入的 num 太小，该命令的内容将会被放到一暂存的 Buf，等到呼叫 scif_FinishComboin 之时，才会将 Buf 的内容自动重新整理以减少封包数量。每个组合封包的大小为 1440 个 byte，若为设定连续数据的函式，如 scif_cmd_ReadI、scif_cmd_ReadR、scif_cmd_ReadF 等，这些函式所读取的数据量可经由其所设定的读取地址数量和数据型态算出，如 ReadI 的数据型态回传为 char 则只占了 1 个 byte，ReadR 的数据回传型态为 unsigned int 则占了 4 个 byte，ReadF 的数据回传型态为 double 则占了 8 个 bytem。若为设定离散数据函式，如 scif_cb_ReadI、 scif_cb_ReadR、scif_cb_ReadF 等，因为其离散数据必须在组合封包中加上其地址对应回传值，而一个地址占了 4 个 byte，故 ReadI 的离散数据型态回传共占了(4+1)个 byte，ReadR 的离散数据型态回传占了(4+4)个 byte，ReadF 的数据型态回传占了(4+8)个 byte。

在范例中，所回传的数据量一共是 100*1+ 50*4 + 10*8 + 10*(4+1) + 20*(4+4) + 30*(4+8) = 950 个 byte，故可将这些回传数据放在一个组合封包内，以减少封包数量。

范例

```
char serverindex;
unsigned int addr[32];
scif_StartCombineSet(serverindex);
scif_cmd_ReadI(SC_POLLING_CMD, serverindex, 0, 100);
scif_cmd_ReadR(SC_POLLING_CMD, serverindex, 10, 50);
scif_cmd_ReadF(SC_POLLING_CMD, serverindex, 20, 10);
scif_cb_ReadI(SC_POLLING_CMD, serverindex, 10, addr);
scif_cb_ReadR(SC_POLLING_CMD, serverindex, 20, addr);
scif_cb_ReadF(SC_POLLING_CMD, serverindex, 30, addr);
scif_FinishCombineSet(serverindex);
```

被重新组合的标准：

命令型态为 SC_POLLING_CMD。

I,O,C,S,A,TMR,CNT→num 小于等于 MAX_CB_BIT_NUM/2。

R,TMRV,TMRS,CNTV,CNTS→num 小于等于 MAX_CB_INT_NUM/2。

F,P→num 小于等于 MAX_CB_FIX_NUM/2。

MAX_CB_BIT_NUM、MAX_CB_INT_NUM、MAX_CB_FIX_NUM

scif_define.h 档中定义。

3.31. scif_SetMaxGroupPkg

此函式为设定合并封包中包含命令封包的最大数量。当所要同步的命令封包数量很多时，将所有命令封包全部轮询一遍的时间也会相对的较长，PC 端的数据更新也会变慢。此项设定能让数据更新速度变快，但是若网络质量不佳，可能较容易发生数据遗失而需重送的情况。除非数据量真的快大，反应速度无法接受，或是一对一联机，才建议进行设定，以免占据过多网络带宽。

语法

int scif_SetMaxGroupPkg(char ServerIdx, unsigned char Count);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时， struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

Count：

最大组合封包数量。0、1 代表不执行合并动作，2 以上代表合并封包中包含命令封包的最数量，目前若设入大于 10 的数字，将会自动被记录为 10。

回传值

0：设定失败。

1：设定成功。

范例

```
char serverindex, count;
int Status;
Status = scif_SetMaxGroupPkg(serverindex, count);
```

4. 文件传输

此章节说明如何透过我们的函式与控制器间作档案的传输，包括上传档案、下载文件，在控制器上建立文件夹等动作。

4.1. scif_FtpSetTalk

此函式用来设定后续文件传输指令所对应的联机。

语法

int scif_FtpSetTalk(char ServerIdx);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时， struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

回传值

0：设定失败。

1：设定成功。

范例

```
char serverindex;
int Status;
Status = scif_FtpSetTalk(serverindex);
```

4.2. scif_FtpUploadFile

此函式为上传单一档案到控制器的文件夹。呼叫此函式后必须呼叫 scif_FtpCheckDone 函式来确认执行状态，并且等到回传值为 1，告知执行动作已完成后，才能再进行下一个 ftp 文件传输的动作。

语法

int scif_FtpUploadFile(char Folder, char *Filename, char *LocalFilename);

参数

Folder：

控制器所要上传档案的目标文件夹，可在 scif_define.h 档中找到定义，

例：#define FTP_FOLDER_NCFILES 10

Filename：

文件名。

LocalFilename：

PC 端的档案完整路径。

回传值

0：设定失败。

1：设定成功。

范例

```
char folder, filename[32], localfilename[32];
int Status;
Status = scif_FtpUploadFile(folder, filename, localFilename);
```

4.3. scif_FtpDownloadFile

此函式为从控制器下载单一档案到本地的文件夹。呼叫此函式后必须呼叫 scif_FtpCheckDone 函式来确认执行状态，并且等到回传值为 1，告知执行动作已完成后，才能再进行下一个 ftp 文件传输的动作。

语法

int scif_FtpDownloadFile(char Folder, char *Filename, char *LocalFilename);

参数

Folder：

控制器所要下载文件的目标文件夹，可在 scif_define.h 档中找到定义，

例：#define FTP_FOLDER_NCFILES 10

Filename：

文件名。

LocalFilename：

PC 端的档案完整路径。

回传值

0：设定失败。

1：设定成功。

范例

```
char folder, filename[32], localfilename[32];
int Status;
Status = scif_FtpDownloadFile(folder, filename, localFilename);
```

4.4. scif_FtpDeleteFile

此函式为删除控制器上的档案。呼叫此函式后必须呼叫 scif_FtpCheckDone 函式来确认执行状态，并且等到回传值为 1，告知执行动作已完成后，才能再进行下一个 ftp 文件传输的动作。

语法

int scif_FtpDeleteFile(char Folder, char *Filename);

参数

Folder：

控制器所要删除档案的目标文件夹，可在 scif_define.h 档中找到定义，

例：#define FTP_FOLDER_NCFILES 10

Filename：

文件名。

回传值

0：设定失败。

1：设定成功。

范例

```
char folder, filename[32];
int Status;
Status = scif_FtpDeleteFile(folder, filename);
```

4.5. scif_FtpUploadFiles

此函式为上传多个档案到控制器的文件夹。呼叫此函式后必须呼叫 scif_FtpCheckDone 函式来确认执行状态，并且等到回传值为 1，告知执行动作已完成后，才能再进行下一个 ftp 文件传输的动作。

语法

```
int scif_FtpUploadFiles(unsigned char Count, FTP_TRANFER_FILE
*TransferFiles);
```

参数

Count：

所要上传的档案数量。

TransferFiles：

文件传输的设定数据结构指针，在 scif_define.h 档中定义。

```
typedef struct tag_FTP_TRANFER_FILE
{
        char Folder;
        char Filename[FILENAME_LENGTH];  //文件名的最大字符数
        char LocalFilename[256];
}FTP_TRANFER_FILE;
```

回传值

0：设定失败。

1：设定成功。

范例

```
unsigned char count;
FTP_TRANFER_FILE TransferFiles[8];
int Status;
Status = scif_FtpUploadFiles(count, TransferFiles);
```

4.6. scif_FtpDownloadFiles

此函式为从控制器的文件夹下载多个档案到本地的文件夹。呼叫此函式后必须呼叫 scif_FtpCheckDone 函式来确认执行状态，并且等到回传值为 1，告知执行动作已完成后，才能再进行下一个 ftp 文件传输的动作。

语法

```
int scif_FtpDownloadFiles(unsigned char Count, FTP_TRANFER_FILE
*TransferFiles);
```

参数

Count：

所要下载的档案数量。

TransferFiles：

文件传输的设定数据结构指针，在 scif_define.h 档中定义。

```
typedef struct tag_FTP_TRANFER_FILE
{
    char Folder;
    char Filename[FILENAME_LENGTH];  //文件名的最大字符数
    char LocalFilename[256];
}FTP_TRANFER_FILE;
```

回传值

0：设定失败。

1：设定成功。

范例

```
unsigned char count;
FTP_TRANFER_FILE TransferFiles[8];
int Status;
Status = scif_FtpDownloadFiles(count, TransferFiles);
```

4.7. scif_FtpDeleteFiles

此函式为删除控制器文件夹内的多个档案。呼叫此函式后必须呼叫 scif_FtpCheckDone 函式来确认执行状态，并且等到回传值为 1，告知执行动作已完成后，才能再进行下一个 ftp 文件传输的动作。

语法

```
int scif_FtpDeleteFiles(unsigned char Count, FTP_TRANFER_FILE
*TransferFiles);
```

参数

Count：

所要删除的档案数量。

TransferFiles：

文件传输的设定数据结构指针，在 scif_define.h 档中定义。

```
typedef struct tag_FTP_TRANFER_FILE
{
    charFolder;
    char Filename[FILENAME_LENGTH];  //文件名的最大字符数
    char LocalFilename[256];
}FTP_TRANFER_FILE;
```

回传值

0：设定失败。

1：设定成功。

范例

```
unsigned char count;
FTP_TRANFER_FILE TransferFiles[8];
int Status;
Status = scif_FtpDeleteFiles(count, TransferFiles);
```

4.8. scif_FtpCheckDone

此函式为取得 ftp 文件传输执行结果，每当呼叫完关于上传、下载、删除案、建立目录等函式后，皆需要呼叫此函式，当此函式回传值为 1 后，代表动作完成，FTP 的状态将回复成闲置状态，如此才能再进行下一个有关于 FTP 传输的动作。

语法

int scif_FtpCheckDone(unsigned char *State, unsigned char *Result);

参数

State：

用以回传文件传输最后状态。在 scif_define.h 档中定义。

Result：

用以回传文件传输结果，在 scif_define.h 档中定义。

回传值

1：要求执行的动作已完成。

0：未完成。

范例

```
unsigned char state, result;
int Status;
Status = scif_FtpCheckDone(&state, &result);
```

4.9. scif_FtpMakeDir

在控制器上建立文件夹，呼叫此函式后必须呼叫 scif_FtpCheckDone 函式来确认执行状态，并且等到回传值为 1，告知执行动作已完成后，才能再进行下一个 ftp 文件传输的动作。

语法

int scif_FtpMakeDir(char Folder, char *DirName);

参数

Folder：

控制器所的目标文件夹，可在 scif_define.h 档中找到定义，

例：#define FTP_FOLDER_NCFILES 10

DirName：

文件夹名称指标。

回传值

0：命令设定失败。

1：命令设定成功。

范例

```
char folder, dirname[32];
int Status;
Status = scif_FtpMakeDir(folder, dirname);
```

4.10. scif_FtpGetFileList

取得控制器文件夹内的档案列表，并建立档案索引，呼叫此函式后必须呼叫 scif_FtpCheckDone 函式来确认执行状态，并且等到回传值为 1，告知执行动作已完成后，才能再进行下一个 ftp 文件传输的动作。

语法

int scif_FtpGetFileList(char Folder, char *HeadFilter, char *TailFilter);

参数

Folder：

控制器所的目标文件夹，可在 scif_define.h 档中找到定义，

例：#define FTP_FOLDER_NCFILES 10

HeadFilter：

文件名前导字符过滤字符串。

TailFilter：

文件名终止符过滤字符串。

回传值

0：命令设定失败。

1：命令设定成功。

范例

```
char folder, headfilter [8], tailfilter[8];
int Status;
Status = scif_FtpGetFileList(folder, headfilter, tailfilter);
```

4.11. scif_FtpReadFileCount

读取 FTP 档案清单中的档案个数，呼叫此函式前，必须过执行 scif_FtpGetFileList 函式。

语法

int scif_FtpReadFileCount();

参数

无。

回传值

档案清单中的档案个数。

范例

```
int Status;
Status = scif_FtpReadFileCount();
```

4.12. scif_FtpReadFile

读取 FTP 文件名，呼叫此函式前，必须执行过 scif_FtpGetFileList 函式。

语法

int scif_FtpReadFile(unsigned short Index, FTP_FILE *File);

参数

Index：

要读取的档案索引。

File：

用来接收文件属性数据的结构，在 scif_define.h 档中定义。

```
typedef struct tag_FTP_FILE
{
        char      filename[FILENAME_LENGTH];//文件名的最大字符数
        unsigned int      filesize;
        unsigned short    year;
        unsigned char     month;
        unsigned char     day;
        unsigned char     hour;
        unsigned char     minute;
        unsigned char     second;
        unsigned char     Reserve;
}FTP_FILE;
```

回传值

0：命令设定失败。

1：命令设定成功。

范例

```
unsigned short index;
FTP_FILE file
int Status;
Status = scif_FtpReadFile(index, &file);
```

4.13. scif_FileGetFileList

取得本地端档案清单，并建立起档案索引。

语法

int scif_FileGetFileList(char *Path, char *HeadFilter, char *TailFilter);

参数

Path：

本地端文件夹路径。

HeadFilter：

文件名前导字符过滤字符串。

TailFilter：

文件名终止符过滤字符串。

回传值

0：命令设定失败。

1：命令设定成功。

范例

```
char path[32], headfilter [8], tailfilter[8];
int Status;
Status = scif_FileGetFileList(path, headfilter, tailfilter);
```

4.14. scif_FileReadFileCount

读取本地端档案清单中的档案个数，呼叫此函式前，必须过执行 scif_FileGetFileList 函式。

语法

int scif_FileReadFileCount();

参数

无。

回传值

档案清单中的档案个数。

范例

```
int Status;
Status = scif_FileReadFileCount();
```

4.15. scif_FileReadFile

读取本地端文件名，呼叫此函式前，必须执行过 scif_FileGetFileList 函式。

语法

int scif_FileReadFile(unsigned short Index, FTP_FILE *File);

参数

Index：

要读取的档案索引。

File：

用来接收文件属性数据的结构，在 scif_define.h 档中定义。

```
typedef struct tag_FTP_FILE
{
        char      filename[FILENAME_LENGTH];//文件名的最大字符数
        unsigned int     filesize;
        unsigned short   year;
        unsigned char    month;
        unsigned char    day;
        unsigned char    hour;
        unsigned char    minute;
        unsigned char    second;
        unsigned char    Reserve;
}FTP_FILE;
```

回传值

0：命令设定失败。

1：命令设定成功。

范例

```
unsigned short index;
FTP_FILE file
int Status;
Status = scif_FileReadFile(index, &file);
```

4.16. scif_FileDeleteFile

此函式为删除本地端档案。呼叫此函式前，必须执行过 scif_FileGetFileList 函式。

语法

int scif_FileDeleteFile(unsigned short Index);

参数

Index：

要删除的档案在本地档案列表中的索引。

回传值

0：设定失败。

1：设定成功。

范例

```
unsigned short index;
int Status;
Status = scif_FileDeleteFile(index);
```

5. 函式库内部信息

此章节说明如何取得函式库内部信息，信息内容分为一般数据、联机信息及错误信息取得。

5.1. scif_GetCommonMsg

此函式为从函式库内部信息取得一般的信息。

语法

unsigned int scif_GetCommonMsg(char id);

参数

id：可以为以下值，在 scif_define.h 档中定义。

```
#define SCIF_PROC_COUNTER              1    //porcess counter
#define SCIF_MEDIA_STEP                5    //与媒合主机通讯的处理步骤
#define SCIF_MEDIA_STATE               6    //与媒合主机通讯的结果
#define SCIF_FTP_STATE                 11   //FTP 状态
#define SCIF_FTP_RESULT                12   //FTP 处理结果
#define SCIF_FTP_STEP                  13   //FTP 处理步骤
#define SCIF_FTP_TOTAL_PACKAGE         21   //FTP 传送总封包数
#define SCIF_FTP_CURRENT_PACKAGE       22   //FTP 已处理的封包数
#define SCIF_FTP_TOTAL_FILE            31   //FTP 传输档案
#define SCIF_FTP_CURRENT_FILE          32   //FTP 已处理的档案数
```

回传值

内部的资料内容，与 id 有关。

范例

```
unsigned int Status;
char id;
Status = scif_GetCommonMsg(id);
```

5.2. scif_GetTalkMsg

此函式为从函式库内部信息取得联机信息。

语法

unsigned int scif_GetTalkMsg(char ServerIdx, char id);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时， struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

id：可以为以下值，在 scif_define.h 档中定义。

```
#define SCIF_CONNECT_STATE          2   //联机状态
#define SCIF_REMOTE_IPLONG          3   //目前的联机对象
#define SCIF_CONNECT_STEP           4   //联机步骤
#define SCIF_CONNECT_RESPONSE       5   //联机回应状态
#define SCIF_TALK_STATE             6   //数据通讯状态
#define SCIF_RESPONSE_TIME          11  //目前封包的反应时间
#define SCIF_OK_COUNT               12  //正确封包次数
#define SCIF_CRC_ERR_CNT            13  //CRC 错误次数
#define SCIF_LOOP_QUEUE_PKG_COUNT   21  //LOOP QUEUE 中的封包笔数
#define SCIF_DIRECT_QUEUE_PKG_COUNT 22  //Direct Queue 中的封包笔数
#define SCIF_LOOP_COUNT             23  //LOOP QUEUE 的查询循环次数
```

回传值

内部的资料内容，与 id 有关。

范例

```
unsigned int Status;
char serverindex, id;
Status = scif_GetTalkMsg(serverindex, id);
```

5.3. scif_GetTalkError

此函式为从函式库内部信息取得错误讯息，而错误数据被读取后即会立即被清除。

语法

void scif_GetTalkError(char ServerIdx, ERROR_MSG *Msg);

参数

ServerIdx：

使用的联机索引，用户可自定义，但此值必须小于 scif_Init 函式初始化时， struct DLL_USE_SETTING 中 TalkInfoNum 所设定的联机数目。

Msg：

错误数据的结构指针，用来回传错误内容，在 scif_define.h 档中定义。

```
typedef struct tag_ERROR_MSG
{
  unsigned char Type;
  unsigned char Cmd;
  unsigned int  addr;
  unsigned int  num;
  unsigned char Error;
}ERROR_MSG;
```

回传值

无。

范例

scif_GetTalkError()

附录 A scif_define.h 档内容

```
//功能限定的定义

#define RECON_CONFIGURE     1
#define RECON_DEBUGGER      2
#define RECON_FTP           3
#define RECON_HMI           4
#define RECON_SHOPFLOOR     5

//------------------------------
#define MAX_SYNC_COUNT      10   //Mapper 中所用的最大 Sync 数
#define MAX_CONTROLLER_NUM_PER_MAKER   300    //向 Media 读
取控制器清单时，最大的允许数量

//------------------------------
#define BIT_CB_SIZE         4096     //Combin 封包中包含的最大地址数
                                       for Bit(I,O,C,S,A)
#define INT_CB_SIZE         4096     //Combin 封包中包含的最大地址数
                                       for Int(R)
#define FIX_CB_SIZE         4096      //Combin 封包中包含的最大地址数
                                       for Fix(double)
#define MAX_DEFAULT_SIZE    32       //Loop Queue 可容纳的通讯笔数
#define MAX_POLLING_SIZE    64       //Loop Queue 可容纳的通讯笔数
#define MAX_DIRECT_SIZE     64       //Direct Queue 可容纳的通讯笔数
#define DIRECT_ADDR_MASK  0x3F   //Direct 地址的 mask，要与
MAX_DIRECT_SIZE 搭配

//------------------------------
#define MAX_DATA_BYTES   1440
#define MAX_BIT_NUM         1440    // MAX_DATA_BYTES / 1
#define MAX_WORD_NUM        720    // MAX_DATA_BYTES / 2
#define MAX_INT_NUM         360    // MAX_DATA_BYTES / 4
```

```
#define MAX_FIX_NUM             180     // MAX_DATA_BYTES / 8
#define MAX_PTR_NUM             180     // MAX_DATA_BYTES / 8
#define MAX_CB_NUM              288     // MAX_DATA_BYTES /(4+1)
                                        // 地址 4bytes, data 为 byte 时(1byte)
#define MAX_CB_BIT_NUM          288     // MAX_DATA_BYTES /(4+1)
#define MAX_CB_WORD_NUM         240     // MAX_DATA_BYTES /(4+2)
#define MAX_CB_INT_NUM          180     // MAX_DATA_BYTES /(4+4)
#define MAX_CB_FIX_NUM          120     // MAX_DATA_BYTES /(4+8)
#define MAX_CB_PTR_NUM          120     // MAX_DATA_BYTES /(4+8)

#define I_OFFSET            0
#define O_OFFSET            (5120*1)
#define C_OFFSET            (5120*2)
#define S_OFFSET            (5120*3)
#define A_OFFSET            (5120*4)
#define TT_OFFSET           (5120*5)
#define CT_OFFSET           (5120*6)
#define RBIT_OFFSET         100000

#define R_OFFSET            0
#define TV_OFFSET           10000000
#define TS_OFFSET           10500000
#define CV_OFFSET           11000000
#define CS_OFFSET           11500000
#define F_OFFSET            10000000

#define I_NUM               4096
#define O_NUM               4096
#define C_NUM               4096
#define S_NUM               4096
#define A_NUM               4096
#define TT_NUM              256
```

```
#define CT_NUM              256
#define R_NUM               6000000
#define TV_NUM              256
#define TS_NUM              256
#define CV_NUM              256
#define CS_NUM              256
#define F_NUM               100000

//---------scif_GetCommonMsg 的自变量
#define SCIF_PROC_COUNTER            1   //porcess counter
#define SCIF_MEDIA_STEP              5   //与媒合主机通讯的处理步骤
#define SCIF_MEDIA_STATE             6   //与媒合主机通讯的结果
#define SCIF_FTP_STATE               11  //FTP 状态
#define SCIF_FTP_RESULT              12  //FTP 处理结果
#define SCIF_FTP_STEP                13  //FTP 处理步骤
#define SCIF_FTP_TOTAL_PACKAGE       21  //FTP 传送总封包数
#define SCIF_FTP_CURRENT_PACKAGE     22  //FTP 已处理的封包数
#define SCIF_FTP_TOTAL_FILE          31  //FTP 传输档案
#define SCIF_FTP_CURRENT_FILE        32  //FTP 已处理的档案数

//-----scif_GetTalkMsg 的自变量
#define SCIF_CONNECT_STATE           2   //联机状态
#define SCIF_REMOTE_IPLONG           3   //目前的联机对象
#define SCIF_CONNECT_STEP            4   //联机步骤
#define SCIF_CONNECT_RESPONSE        5   //联机回应状态
#define SCIF_TALK_STATE              6   //数据通讯状态
#define SCIF_RESPONSE_TIME           11  //目前封包的反应时间
#define SCIF_OK_COUNT                12  //正确封包次数
#define SCIF_CRC_ERR_CNT             13  //CRC 错误次数
```

```
#define SCIF_LOOP_QUEUE_PKG_COUNT  21  //LOOP QUEUE 中的封包笔数
#define SCIF_DIRECT_QUEUE_PKG_COUNT 22  //Direct Queue 中的封包笔数
#define SCIF_LOOP_COUNT                 23    //LOOP QUEUE 的查询循环次数

//联机状态由  scif_GetTalkMsg(SCIF_CONNECT_STATE)  取得
#define SC_CONN_STATE_DISCONNECT   0     //联机关闭
#define SC_CONN_STATE_CONNECTING   1     //联机中
#define SC_CONN_STATE_FAIL              2     //联机失败
#define SC_CONN_STATE_OK                3     //联机正常
#define SC_CONN_STATE_NORESPONSE   4     //联机无回应

//------------联机回应状态
#define CONNECT_RESULT_NORESPONSE          0
#define CONNECT_RESULT_OLD_SOFTWARE_CLEAR 1 //原本占用的软
                                                体已经清除
#define CONNECT_RESULT_INVALID_SOFTWARE       11//无效的软件代号
#define CONNECT_RESULT_DISABLE_SOFTWARE       12//软件功能停用
#define CONNECT_RESULT_DISABLE_INTERNET       13 //停用自外网来
                                                的联机
#define CONNECT_RESULT_CLOSED_SOFTWARE       14 //软件功能关闭
#define CONNECT_RESULT_CLOSED_INTERNET       15 //关闭自外网来
                                                的联机
#define CONNECT_RESULT_INVALID_MAKERID          16 //不相符的
                                                MakerID
#define CONNECT_RESULT_WAIT_CONFIRM            21 //等待人机确认中
#define CONNECT_RESULT_SOFTWARE_CONNECTED 31//软件已联机
#define CONNECT_RESULT_SOFTWARE_REJECTED   32//停用自外网来
                                                的联机
```

```
#define CONNECT_RESULT_SOFTWARE_INREQ      41//其他使用者占
                                              用此软件
#define CONNECT_RESULT_PENDING              50

//-----FTP 目标文件夹
#define FTP_FOLDER_NCFILES                  10
#define FTP_FOLDER_MACRO_MAKER              20
#define FTP_FOLDER_MACRO                    21
#define FTP_FOLDER_MACHINE                  30
#define FTP_FOLDER_SETUP                    40
#define FTP_FOLDER_SETUP_MACHINE            41
#define FTP_FOLDER_BAK                      50
#define FTP_FOLDER_DATA                     51
#define FTP_FOLDER_LANGUAGE                 52
#define FTP_FOLDER_LANGUAGE_DEF             53
#define FTP_FOLDER_LOG                      54
#define FTP_FOLDER_RECORD                   55

//------FTP  状态
#define FTP_STATE_IDLE                      0   //闲置
#define FTP_STATE_UPLOAD                    1    //上传
#define FTP_STATE_DOWNLOAD                  2   //下载
#define FTP_STATE_DELETE                    3   //删除
#define FTP_STATE_LIST                      11   //取得目录
#define FTP_STATE_UPLOAD_MANY               21   //上传多个
#define FTP_STATE_DOWNLOAD_MANY             22   //下载多个
#define FTP_STATE_DELETE_MANY               23   //删除多个
#define FTP_STATE_MAKE_DIR                  30  //建立目录
#define FTP_STATE_PENDING                   99  //命令设定中
```

```
//------FTP  处理结果
#define FTP_RESULT_IDLE                  0     //无
#define FTP_RESULT_PROCESSING            1     //处理中
#define FTP_RESULT_SUCCESS               2     //成功

#define FTP_RESULT_FAIL_TO_READ_LOCAL_FILE     11  //读取本地档案
                                                   失败
#define FTP_RESULT_FAIL_TO_WRITE_LOCAL_FILE    12  //写入本地档
                                                   案失败
#define FTP_RESULT_FAIL_TO_READ_REMOTE_FILE    13  //读取远程档
                                                   案失败
#define FTP_RESULT_FAIL_TO_WRITE_REMOTE_FILE 14  //写入远程档
                                                   案失败
#define FTP_RESULT_FAIL_TO_SET_COMMAND     15   //命令传送失败
#define FTP_RESULT_FAIL_TO_COMMUNICATION   16   //通讯错误
#define FTP_RESULT_FILE_MISMATCH           17   //档案比对不正确

//------MEDIA  处理结果
#define MEDIA_RESULT_IDLE                0
#define MEDIA_RESULT_PENDING             1
#define MEDIA_RESULT_PROCESSING          2
#define MEDIA_RESULT_SUCCESS             3
#define MEDIA_RESULT_FAIL                4

//------------数据通讯状态
#define TALK_STATE_NORMAL                0
#define TALK_STATE_ERROR                 1
#define TALK_STATE_OVER_RETRY            2
```

```
//------单笔通讯数据的状态
#define SC_TRANSACTION_PENDING          0       //等待处理中
#define SC_TRANSACTION_PORCESSING       1       //处理中
#define SC_TRANSACTION_FINISH           2       //完成
#define SC_TRANSACTION_INVALID          3       //无效的索引

//------通讯封包错误编号
//错误码为 0                                    //没有发生错误
#define SCIF_ERROR_INVALID_PACKET_SET  255    //Local 检查到此封包设
                                                定无效
//其他编号的错误码由主机传回的错误---直接记录代码即可

//---一些定义
#define FILENAME_LENGTH                 32    //文件名的最大字符数
#define MAX_FILE_LIST_NUM               240   //最大的档案清单大小
#define MAX_TRANSFER_FILE_COUNT   128   //一次传输的最大档案量
#define MAX_SOFTWARE_COUNT          5   //最大的软件种类数

//错误讯息来源
#define ERROR_TYPE_NONE         0
#define ERROR_TYPE_POLLING      1
#define ERROR_TYPE_DIRECT       2

//命令种类
#define SC_DEFAULT_CMD  0  // Default Command--当要在画面上同时显
                                示多个控制器的信息时，应使用此种封包
#define SC_POLLING_CMD   1  // Polling Command
#define SC_DIRECT_CMD    2  // Direct read & setting

typedef struct tag_ERROR_MSG
{
```

```
    unsigned char   Type;
    unsigned char   Cmd;
    unsigned int    addr;
    unsigned int    num;
    unsigned char   Error;
}ERROR_MSG;

//指针数据的结构
typedef struct tag_SC_POINTER
{
    unsigned int PtrA;
    unsigned int PtrV;
}SC_POINTER;
//单笔通讯的数据结构
typedef union tag_SC_DATA
{
    unsigned char     Bytes[MAX_BIT_NUM];       //bit 或 byte 资料
    unsigned short    Words[MAX_WORD_NUM];  //word 资料
    unsigned int      Ints[MAX_INT_NUM];        //整数
    double            Fixs[MAX_FIX_NUM];        //double
    SC_POINTER        Ptrs[MAX_PTR_NUM];        //指标的值
}SC_DATA;

//自动侦测主机的响应封包
typedef struct tag_LOCAL_CONTROLLER_INFO1
{
    unsigned int  IPLong;
    char          IP[16];
    char          Name[16];
}LOCAL_CONTROLLER_INFO;
```

```
//由媒介主机取回的控制器信息
typedef struct tag_MEDIA_CONTROLLER_INFO1
{       int         Idx;
        unsigned int  CtrID;
        unsigned int  Port;
        unsigned int  IPLong;
        char        IP[16];
        char        Name[16];
}MEDIA_CONTROLLER_INFO;

//FTP  或本地列举档案列表传回的档案信息
typedef struct tag_FTP_FILE
{
        char            filename[FILENAME_LENGTH];
        unsigned int      filesize;
        unsigned short    year;
        unsigned char     month;
        unsigned char     day;
        unsigned char     hour;
        unsigned char     minute;
        unsigned char     second;
        unsigned char     Reserve;
}FTP_FILE;

//FTP 文件传输的设定数据
typedef struct tag_FTP_TRANFER_FILE
{
    char Folder;
    char Filename[FILENAME_LENGTH];
    char LocalFilename[256];
}FTP_TRANFER_FILE;
```

```
//功能设定的结构
typedef struct tag_FUNCTION_SETTING
{
    unsigned int    MakerID;
    unsigned int    MakerPwd;
    unsigned int    MediaIPLong1;
    unsigned int    MediaIPLong2;
    unsigned char  HaveMediaFunction;      //有使用媒合主机功能
}FUNCTION_SETTING;

typedef struct tag_DLL_USE_SETTING
{
    unsigned short    TalkInfoNum;    //联机数目
    unsigned char     SoftwareType;   //软件种类
    unsigned int      MemSizeI;
    unsigned int      MemSizeO;
    unsigned int      MemSizeC;
    unsigned int      MemSizeS;
    unsigned int      MemSizeA;
    unsigned int      MemSizeTT;
    unsigned int      MemSizeCT;
    unsigned int      MemSizeR;
    unsigned int      MemSizeTV;
    unsigned int      MemSizeTS;
    unsigned int      MemSizeCV;
    unsigned int      MemSizeCS;
    unsigned int      MemSizeF;
}DLL_USE_SETTING;
```

```
//-----控制器的功能设定
typedef struct tag_FUNC_SETTING
{
    unsigned int    MediaServerIP1;        //媒介主机  IP 1
    unsigned int    MediaServerIP2;        //媒介主机  IP 2
    unsigned char  LocalDetectEnable;    //是否响应区网的侦测
    unsigned char  MediaPrivateEnable;  //向媒介主机取得 Private ID 功能
    unsigned char  MediaReportEnable;  //向媒介主机通报上线
    unsigned char  Reserve1;
    unsigned char  SoftwareEnable[MAX_SOFTWARE_COUNT];
                                        //软件功能//bit0(启用/停用)
                                        //bit1(是否允许外网连入)
                                        //bit2(是否无条件接受联机)
unsigned char  Reserve2[3];
}FUNC_SETTING;
```

史卡拉机器手臂控制器连线函式库使用观念与说明

出版日期：2018年07月第一版印行

1. HIWIN为上銀科技的注册商标，请勿购买来路不明之仿冒品以维护您的权益。
2. 本型录所载规格、照片有时会与实际产品有所差异，包括因为改良而导致外观或规格等发生变化的情况。
3. 凡受”贸易法”等法规限制之相关技术与产品，HIWIN将不会违规擅自出售。若要出口HIWIN受法律规范限制出口的产品，应根据相关法律向主管机关申请出口许可，并不得供作生产或发展核子、生化、飞弹等军事武器之用。
4. HIWIN产品专利清单查询网址：http://www.hiwin.tw/Products/Products_patents.aspx

Copyright © HIWIN Technologies Corp.

HIWIN®
Motion Control and System Technology

海外厂 / 研发中心

上銀科技（中国）有限公司
HIWIN TECHNOLOGIES (CHINA) CORP.
江苏省苏州市苏州工业园区夏庄路2号
Tel : (0512) 8068-5599
Fax: (0512) 8068-9858
www.hiwin.cn

德国 欧芬堡
HIWIN GmbH
OFFENBURG, GERMANY
www.hiwin.de
www.hiwin.eu

意大利 米兰
HIWIN Srl
BRUGHERIO, ITALY
www.hiwin.it

新加坡
HIWIN SINGAPORE
SINGAPORE
www.hiwin.sg

日本 神户・东京・名古屋・长野・东北・静冈・北陆・广岛・福冈・熊本
HIWIN JAPAN
KOBE・TOKYO・NAGOYA・NAGANO・TOHOKU・SHIZUOKA・HOKURIKU・HIROSHIMA・FUKUOKA・KUMAMOTO, JAPAN
www.hiwin.co.jp

瑞士 优纳
HIWIN Schweiz GmbH
JONA, SWITZERLAND
www.hiwin.ch

韩国 水原・马山
HIWIN KOREA
SUWON・MASAN, KOREA
www.hiwin.kr

美国 芝加哥・矽谷
HIWIN USA
CHICAGO・SILICON VALLEY, U.S.A.
www.hiwin.com

捷克 布尔诺
HIWIN s.r.o.
BRNO, CZECH REPUBLIC
www.hiwin.cz

以色列 海法
Mega-Fabs Motion Systems, Ltd.
HAIFA, ISRAEL
www.mega-fabs.com

全球营运总部

上銀科技股份有限公司
HIWIN TECHNOLOGIES CORP.
台湾40852台中市精密机械园区精科路7号
Tel: +886-4-23594510
Fax: +886-4-23594420
www.hiwin.tw
business@hiwin.tw

本型录的内容规格若有变更，恕不另行通知。

Copyright © HIWIN Technologies Corp.

©2018 FORM 2018-07 (PRINTED IN TAIWAN)