

ISaGRAF

Version 3.4

사용자 가이드

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.

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.

항목보기

A. 사용자 가이드	1
A.1 시작	1
A.1.1 ISaGRAF 설치	1
A.1.2 온라인 도움말의 사용법	4
A.1.3 샘플 어플리케이션	4
A.2 프로젝트 관리	11
A.2.1 프로젝트 생성 / 작업	11
A.2.2 복수 프로젝트 그룹 관리	13
A.2.3 옵션	14
A.2.4 도구	15
A.3 프로그램 관리	16
A.3.1 프로젝트 구성요소	16
A.3.2 프로그램 관리	19
A.3.3 TIC 코드 [만들기]	23
A.3.4 기타 ISaGRAF 도구	24
A.3.5 [도구] 메뉴에 사용자 가이드 추가	25
A.3.6 시뮬레이션 - 디버깅	25
A.4 SFC 편집기 사용법	28
A.4.1 SFC 언어 메인 토픽	28
A.4.2 SFC chart 입력	31
A.4.3 SFC chart 의 작업	33
A.4.4 레벨 2 프로그램 입력	34
A.4.5 SFC 갤러리 사용법	38
A.5 Flow Chart 편집기 사용법	39

A.5.1	FC 언어 기본	39
A.5.2	Flow Chart 들어가기	41
A.5.3	chart 에서 작업 하기	44
A.5.4	레벨 2 프로그램	45
A.5.5	프로그램 & QuickLD	46
A.5.6	보기 옵션	47
A.6	QuickLD 편집기 사용법	48
A.6.1	LD 언어 기본 사항	48
A.6.2	LD diagramd 입력	51
A.6.3	작업	54
A.6.4	옵션	55
A.7	FBD/LD 편집기 사용법	57
A.7.1	FBD/LD 언어 기본사항	57
A.7.2	입력 FBD	60
A.7.3	FBD 편집	62
A.7.4	보기 옵션	63
A.7.5	스타일과 편집 트래킹	64
A.8	텍스트 편집기 사용법	66
A.8.1	편집 명령어	66
A.8.2	옵션	67
A.9	프로그램 편집기 상세 정보	68
A.9.1	다른 ISaGRAF 도구 호출	68
A.9.2	파라미터	69
A.9.3	기타 공통적인 「파일」 메뉴	70
A.9.4	일기장갱신	70
A.9.5	변수목록에서 변수선택	71
A.10	변수목록 편집기 사용법	73
A.10.1	변수목록 메인 창	75
A.10.2	변수 관리	76

A.10.3	오브젝트 요약	77
A.10.4	빨리 만들기	79
A.10.5	Modbus SCADA 주소	80
A.10.6	다른 프로그램과 정보 교환	81
A.11	I/O 연결 편집기 사용법	86
A.11.1	I/O 보드 정의	87
A.11.2	보드 파라미터 설정	88
A.11.3	직접 표현 변수	89
A.11.4	숫자 메김	90
A.11.5	채널마다 보호설정	90
A.12	변환 테이블 만들기	92
A.12.1	메인 명령어	92
A.12.2	테이블 포인트 입력	93
A.12.3	규칙과 제한사항	93
A.13	코드 생성기 사용법	95
A.13.1	명령어	95
A.13.2	컴파일러 선택	96
A.13.3	C 소스 코드 생성	99
A.13.4	정보 보기	100
A.13.5	리소스 정의	100
☞	리소스 정의 파일	100
A.14	크로스 래퍼런스	107
A.15	그래픽 디버거 사용법	109
A.15.1	디버거 창	109
A.15.2	어플리케이션 제어	110
A.15.3	옵션	113
A.15.4	"적어넣기(Write)" 명령어	113
A.15.5	On line 수정	115
A.15.6	DDE 통신	120

A.16	변수추적(Spying)	121
A.17	ST / IL 프로그램 디버깅	123
A.18	스포트라이트 사용법	125
A.18.1	그래픽 윤곽구축	125
A.18.2	리스트 윤곽	128
A.18.3	스타일 설정	128
A.18.4	「파일」 메뉴 명령어	129
A.18.5	버전 3.2 사용자로서의 주의 사항	130
A.19	어플리케이션 갱신	131
A.19.1	프로젝트 갱신	131
A.19.2	통신 설정	132
A.19.3	갱신 준비	132
A.19.4	타겟에서 압축 소스 보관	133
A.19.5	타겟에 필요한 메모리 용량	133
A.19.6	갱신 프로젝트에 관해서	134
A.19.7	버전 호환성	134
A.20	디버그 (진단) 도구의 사용법	135
A.21	시뮬레이션 사용법	136
A.21.1	디버거와의 링크	136
A.21.2	I/O 시뮬레이션	136
A.21.3	라이브러리 구성요소	137
A.21.4	옵션	140
A.21.5	입력신호 상태의 보존 / 읽어내기	140
A.21.6	주기개요	141
A.21.7	시뮬레이션 메모	142
A.22	라이브러리 편집기 사용법	150
A.22.1	라이브러리 편집기 관리 : 「파일」 메뉴 명령어	151
A.22.2	I/O 구성	154

A.22.3	I/O 장치 기기	155
A.22.4	I/O 보드	155
A.22.5	IE C언어로 작성된 Function, FunctionBLOCKS	158
A.22.6	"C" Functions / function Blocks	160
A.22.7	변환 함수	161
A.23	보관함 사용법	162
A.23.1	보관함 관리자 호출	162
A.23.2	옵션	163
A.23.3	백업 / 복구	163
A.23.4	보고한함 파일 확장자	164
A.24	인쇄	165
A.24.1	문서 목차 변환	165
A.24.2	옵션	166
A.25	암호 보호	169
A.26	다른 테크닉..	172
A.26.1	다른 ISaGRAF 도구	172
A.26.2	잠금 I/O 와 가상 I/O	172
A.26.3	PC-PLC link validation	176
A.26.4	ISaGRAF 디렉토리 구성	177
A.26.5	어플리케이션 심벌 파일 (APPL.TST)	179
A.26.6	ISaGRAF workbench I/O 무제한판 (W D L) 의 제한	184
B.	언어	1
B.1	프로젝트 구성 (Architecture)	2
B.1.1	프로그램	2
B.1.2	Cyclic 및 sequential 동작	2
B.1.3	Child SFC / FC 프로그램	3
B.1.4	Functions / 서브-프로그램	4
B.1.5	Function blocks	5

B.1.6	6-언어	6
B.1.7	실행 규칙	7
B.2	공통 오브젝트	9
B.2.1	기본 타입	9
B.2.2	상수 표현	9
B.2.3	변수	12
B.2.4	주석	16
B.2.5	예약어	16
B.3	SFC 언어	18
B.3.1	SFC chart 메인	18
B.3.2	SFC 기본 구성	18
B.3.3	분기/ 결합	21
B.3.4	매크로 스텝	23
B.3.5	스텝 내의 액션	24
B.3.6	transitions 조건	30
B.3.7	SFC 다이내믹 규칙	32
B.3.8	SFC 프로그램 계층	33
B.4	Flow Chart 언어	35
B.4.1	FC 기본요소	35
B.4.2	FC 복잡 구조	38
B.4.3	FC 문법검사	39
B.5	FBD 언어	41
B.5.1	FBD 형식	41
B.5.2	RETURN	42
B.5.3	점프 / 라벨	42
B.5.4	이진 반전	43
B.5.5	FBD 에서 function / function block 호출	44
B.6	LD 언어	45
B.6.1	모선 / 접속선	45

B.6.2	다중 접속	46
B.6.3	기본 접점과 코일	47
B.6.4	RETURN 기술법	53
B.6.6	점프 / 라벨	54
B.6.7	LD 내의 Blocks	54
B.7	ST 언어	56
B.7.1	ST 문법	56
B.7.2	표현법과 괄호	57
B.7.3	Function / function block 호출	57
B.7.4	ST 특수 이진 명령어	59
B.7.5	ST 기본	61
B.7.6	ST 확장	67
B.8	IL 언어	73
B.8.1	IL 문법	73
B.8.2	IL 명령	74
B.9	표준명령, FunctionBlock 과 Function	83
B.9.1	표준명령	83
B.9.2	표준 function blocks	104
B.9.3	표준 functions	123
C.	TARGET 사용자 가이드	1
C.1	소개	2
C.2	설치	3
C.3	ISaGRAF DOS target	4
C.3.1	ISaGRAF 실행 : ISA.EXE	4
C.3.2	DOS-Target 특징	5
C.4	ISaGRAF OS9 Target	10
C.4.1	ISaGRAF Single Task 실행: isa 옵션	10

C.4.2	ISaGRAF multitasks: isaker, isatst, isanet	11
C.4.3	OS-9 Target 특징	16
C.5	ISaGRAF VxWorks target	21
C.5.1	시스템 리소스 관리: isassr.o	21
C.5.2	모듈 : isa.o, isakerse.o, isakeret.o	21
C.5.3	ISaGRAF single task 실행 : isa.o	22
C.5.4	ISaGRAF multitasks 실행: isakerse.o and isakeret.o	24
C.5.5	VxWorks Target 의 특징	30
C.6	ISaGRAF NT target	34
C.6.1	ISaGRAF 실행	34
C.6.2	일반 정보 옵션	34
C.1.3	NT-Target 의 특징	41
C.6.3	사용자 interface	46
C.7	"C" 프로그래밍	54
C.7.1	개요	54
C.7.2	"C" 변환 functions	56
C.7.3	"C" Functions	61
C.7.4	"C" FUNCTION BLOCKS	69
C.7.5	컴파일 / 연결 기술	86
C.8	Modbus 연결	93
C.8.1	MODBUS 네트워크 / protocol	93
C.8.2	ISaGRAF implementation	94
C.9	전원 이상시의 관리	101
C.9.1	기본	101
C.9.2	Application 변수의 백업	102
C.9.3	프로그램 백업	106
C.10	부록: 에러 리스트와 설명	108
D.	용어설명	2

E. 색인	1
-------	---

A. 사용자 가이드

A.1 시작

ISaGRAF 는 각종제어기기의 제어프로그램을 개발할 Workbench 를 실행할 타겟이 되는 소프트웨어입니다.

Workbench(개발환경)는 제어 프로그램에 IEC1131-3 을 채용해서 더욱이 개발에 필요한 여러가지 도구를 구비하고 있기 때문에 효율적으로 제어 프로그램의 개발을 행할 수 있습니다.

타겟(실행환경) 은 Workbench 로 개발된 제어프로그램이 PC 라든지 보드 위에 real time 으로 실행되는 소프트웨어로 CPU OS 에 의존하지않는 멀티플랫폼의 실행이 가능합니다.

A.1.1 ISaGRAF 설치

본장에서는 ISaGRAF Workbench 설치에 관한 설명을 합니다.

또한,ISaGRAF 의 간단한 어플리케이션을 집어넣어 ISaGRAF 를 바로 사용할 수 있도록 해설을 합니다.

☞ **하드웨어/ 소프트웨어 요구사항**

- PC 는 D O S / V 또는 N E C 9 8 2 1 시리즈
- C P U는 8 0 4 8 6 이상
- 메모리는 8 M B 이상
- 플로피디스크드라이브는 3.5 인치 1.44M B 드라이브
- C D - R O M드라이브
- 하드디스크 공간은 2 0 M B 이상
- 비디오드라이브는 V G A , S V G A
- 마우스
- 프린트용 parallel 보드 L P T 1 (프로텍션키에 필요)

ISaGRAF Workbench 설치 전에 다음의 소프트가 설치 되어 있을 필요가 있습니다.

- Windows3.1
- Windows95
- WindowsNT version 3.51 또는 4.00 이후



설치레이션 프로그램 사용법

CD - ROM에서 설치 할 수 있습니다. workbench를 설치 할 경우에는 ISaGRAF Workbench **[Korean]**을 선택해 주세요. 선택 후 Install product 폰트를 선택하면 설치가 시작 됩니다. 또한 Make Disks 폰트를 선택하면 설치용 디스크작성을 개시합니다. 화면표시에 따라서 설치를 진행합니다.

ISaGRAF 설치레이션 디렉토리는 과거의 버전이 하드디스크에 존재하는 경우는 신규의 디렉토리를 지정해 주세요.

타겟을 설치할 경우는 목적 타겟을 선택해 주세요.

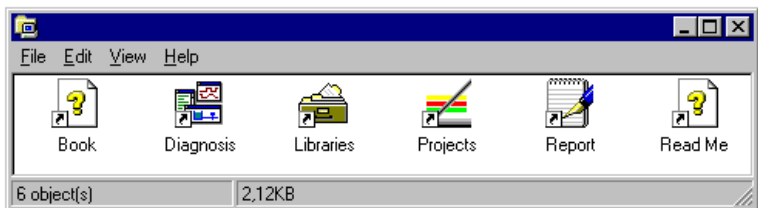
- ISaGRAF 실행 프로그램
- 온라인도큐먼트 및 도움말파일
- ISaGRAF 표준 라이브러리
- ISaGRAF 샘플프로그램

ISaGRAF 설치 될 디렉토리는 기본이 **[ISAWIN]**으로 되어 있습니다. 모든 파일이 복사된 후는 프로그램 관리자에게는 다음의 그룹이 추가 됩니다. 하드디스크에 복사된 압축 라이브러리 파일은 복사된 후에 풀립니다.

Windows 디렉토리에 ISA.ini 파일이 있을 경우 이것을 삭제해 주세요. (데모프로그램이 이전에 설치 되면 ISA.ini 파일이 Windows 디렉토리에 복사되어 있을 가능성이 있습니다.)

Windows 프로그램관리자에 [ISaGRAF 3.2]그룹 (또는 필터) 에 등록된 설치전의 \EXE 서브디렉토리는 ISA.ini 파일이 작성됩니다.

(ISaGRAF 설치용 서브디렉토리 이외의 파일은 복사되지 않습니다.)



프로젝트 :프로젝트관리

진단:.....사용자용 진단 도구
라이브러리:라이브러리 관리
도큐먼트:.....ISaGRAF 에 사용된 언어의 온라인도큐먼트
Read Me:.....ISaGRAF 의 새버전에 관한 정보
레포트:.....제출용 버그레포트 양식

만약 버그등의 문제에 직면한 경우는 버그레포트보고의 양식에 필요사항을 기입해서 송부 부탁드립니다.

≡ 시스템 파일 갱신

ISaGRAF 는 DOS 의 환경변수는 사용하지 않지만 다음의 내용이 들어 있지 않으면 CONFIG.SYS 의 항목에 들어있지 않으면 추가할 필요가 있습니다.

만약 이미 존재해 있는 경우는 숫자가 20 이상으로 있는 것을 확인해서 필요하다면 변경시켜 주세요.

ISaGRAF 실행디렉토리는 path 에 포함시킬 필요는 없습니다.

```
files=20
buffers=20
```

ISaGRAF Workbench 는 ISaGRAF 타겟통신에 기본으로 시리얼통신 (COM1) 을 사용합니다. 만약 COM1 이 이미 마우스 등으로 사용중인 경우는 만약 가능 하다면 마우스를 COM2 로 변경 바랍니다.

⇒ Windows NT 사용자에 대한 주의 사항:

만약 ISaGRAF Workbench 을 WindowsNT 3.51, 4.00 에 설치 할 경우는 지정된 파일에 다음의 라인 (NT=1) 을 메뉴얼로 추가할 필요가 있습니다. 추가장소는 파일명 ISA.ini 섹션[WS001]로 기본 디렉토리는 \ISAWIN\EXE 입니다.

```
[WS001]
NT=1
Isa=C:\ISAWIN
IsaExe=C:\ISAWIN\EXE
IsaApl=C:\ISAWIN\APL1
IsaTmp=C:\ISAWIN\TMP
```

역시 이 설정은 COM1 등을 사용한 Serial 통신 직전에 필요하게 됩니다.

≡ 프로텍션키

PC 의 프린트보드에 접속된 하드웨어프로텍션키는 불법적인 소프트 복사를 방지하기 위하여 사용되어 집니다.

그러나 본키가 접속되어 있지않는 경우에도 대부분의 ISaGRAF Workbench 기능을 사용할 수 있습니다. 프로텍션키에 의해 ISaGRAF Workbench 의 옵션을 지정합니다. 예를 들면 개발어플리케이션의 최대서비스와 접속 I/O 최대수 등입니다. 프로텍션키가 바르게 접속되어 있는가의 확인은 ISaGRAF 어느 창에서도[도움말]메뉴의[버전정보]로서 할 수 있습니다. 여기에는 ISaGRAF Workbench 옵션이 표시됩니다.

⇒ Windows NT 사용자에게 대한 주의사항:

만약 ISaGRAF Workbench 를 WindowsNT 3.51, 4.00 에 설치 할 곳은 워크벤치의 설치와는 별개로 [Sentinel/Rainbow™] 의 드라이버를 설치 해야합니다.

A.1.2 온라인 도움말의 사용법

온라인도큐멘테이션은 아래의 토픽에 관해서 ISaGRAF 워크벤치에 설치 되어 있습니다.

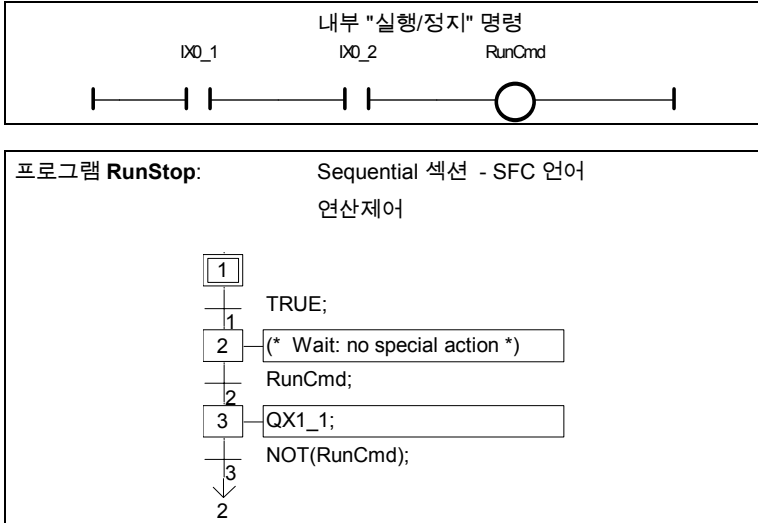
- ISaGRAF 언어 레퍼런스 메뉴얼
- 사용자 가이드 (ISaGRAF 전반의 정보)
- 표준 라이브러리내의 요소에 관한 기술메모

A.1.3 샘플 어플리케이션

여기에서는 컴팩트한 어플리케이션을 설계,제작,테스트하기 위해 모든 기본적인 조작에 관해 순서에 따라 설명합니다.

아래 도면은 L D와 S F C의 표현으로 적혀진 어플리케이션을 나타냅니다.상세한 설명은 뒷장에 설명이 있기 때문에 여기에서는 프로그램작성에서 시뮬레이션까지의 흐름을 파악 할 수 있습니다.

이진 변수:	
IX0_1, IX0_2:	연산 명령용 입력 변수
RunCmd:	내부 "실행/정지" 명령어
QX1_1:	출력 변수: 프로세스 상태
프로그램 명령어:	
사이클 시작 섹션 - LD 언어	



Start ISaGRAF workbench 동작

ISaGRAF Workbench 를 작동하기 위해서는 ISaGRAF 그룹 (폴더) 내의 [프로젝트]아이콘을 더블클릭합니다. 그러면 프로젝트관리 창이 열립니다.



프로젝트의 작성

「파일」 메뉴의 「새파일」 명령어로[RunStop]라는 이름으로 프로젝트를 작성합니다.

다이아로그박스에서는 아래와 같이 입력합니다.

프로젝트명: "RunStop"

I/O 구성의 선택: "Sim_Boo"

"확인" 버튼

이것으로 프로젝트가 완성됩니다.

(I/O 구성의 선택을 행하는 것으로 I/O 변수의 변수목록등록은 자동적으로 됩니다.)



프로그램관리창 열기

프로젝트 안에서 프로그램은 프로젝트관리창으로 「파일」 메뉴의 「편집」 명령어를 선택하든지 프로젝트를 더블클릭하는 것으로

프로그램관리 창을 열 수 있습니다. (이 시점에서는 프로그램은 아직 작성되어 있지 않습니다.)



프로그램작성

여기에서 「파일」 메뉴의 「새파일」 명령어를 선택해서 우선 최초의 프로그램을 작성합니다. 다이아로그박스가 열려있습니다. 아래와 같이 입력, 선택합니다.

프로그램명 입력 : "Command".

언어의 선택 : "QuickLD"

프로그램섹션의 선택 : "Begin" 섹션

"확인" 버튼

같은 형식으로 2 번째 프로그램을 작성합니다.

「파일」 메뉴의 「새파일」 명령어를 선택해서 우선 최초의 프로그램을 작성합니다. 다이아로그박스가 열리도록 아래와 같이 입력 선택합니다.

프로그램명의 입력 : "RunStop".

언어의 선택 : "SFC"

프로그램섹션의 선택 : "Sequential" 섹션

"확인" 버튼

이것으로 두개의 프로그램이 작성되었습니다. 이것들은 프로그램관리 창에 표시됩니다.



변수의 선언 (변수목록편집기 열기)

프로그램 내용을 작성하기 전에 프로그램으로 사용된 내부변수를 변수목록에 선언해 둘 필요가 있습니다. 때문에 「파일」 - 「변수목록」 메뉴명령어로 변수목록편집기를 엽니다. 역시 I/O 변수에 관해서 프로젝트를 신규작성했을 때 I/O 구성을 선택하기 위해 이미 자동적으로 선언 되고 있습니다. (프로젝트를 작성할 때[sim_boo]을 선택하기 위해 시뮬레이션용의 I/O 가 자동생성 되어 있습니다.)



전역변수의 등록

변수목록편집기로 「파일」 - 「기타」 명령어에서[전역변수][풀형]을 선택합니다. 이 조작으로 전역변수의 풀형변수용 편집기가 열립니다. (같은 형식으로 조작하기 위해 도구바 아이콘의 선택도 가능합니다.)



「편집」 - 「새파일」 명령어에 따라 신규 풀형변수를 작성 (등록) 할 수가 있습니다. (같은 형식으로 조작하기 위해 도구바 아이콘에서[변수의 삽입]을 선택하는 것도 가능합니다.)

다이아로그박스가 열린다면 아래와 같이 입력을 행합니다.

이름 : RunCmd

명령어 Run/Stop 명령어

속성 : 속성의 선택

"저장" 버튼의 선택으로 변수가 작성됩니다.

"취소" 버튼으로 다이아로그박스에서의 설정내용을 취소합니다.

마지막으로 수정내용을 보존한 후에 변수목록편집기를 종료합니다. 「파일」 - 「종료」 명령어를 선택해서 [수정내용을 보존합니까 ?]에 대해서[예]를 선택합니다.



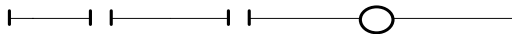
QuickLD 프로그램의 편집

L D 프로그램을 편집해 봅시다. 프로그램관리 창에서 프로그램명을 더블클릭 합니다 (또는 「파일」 - 「편집」 명령어를 선택합니다.)



QuickLD 편집기가 열립니다. 창사이즈를 필요에 따라서 최대화 등을 선택해서 조정합니다.

F2 F3 Function 키 F2, F3 을 입력 (또는 마우스로 도구바 버튼을 클릭) 합니다.
(")



L D표의 심벌에 대응하는 변수를 나눠부칩니다. 키보드의 커서와 마우스로 심벌을 선택해서 ENTER 키를 입력합니다. 변수의 선택용 다이아로그박스가 열립니다. (선택할 변수의 스코프가[전역]이 선택되고 변수의 형이[boolean 형]이 되어 있는 것을 확인해 주세요.)

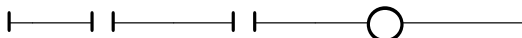
최초의 a 접점에는 IX0_1 을 선택해서 [확인]버튼을 입력

그다음 a 접점에는 IX0_2 을 선택해서 [확인]버튼을 입력

코일에는 RunCmd 을 선택해서 [확인]버튼을 입력

이것으로 L D 프로그램 입력을 할 수 있었습니다. 결과는 아래와 같이 됩니다.

IX0_1 IX0_2 RunCmd



프로그램의 작성이 완료되었기 때문에 프로그램편집기를 종료합니다. 종료시에는 수정내용을 보존해 주세요.



SFC 프로그램의 편집

"RunStop"

SFC 프로그램을 편집하기 위해서 프로그램관리 창에서 프로그램명을 더블클릭합니다.



S F C 프로그램편집기가 열립니다. 필요에 따라서 창사이즈를 변경 또는 최대화합니다.

적당히 S F C편집기의 옵션을 선택합니다. 예를 들면

「옵션」 메뉴에서 「새로고치기」 레벨편집기를 선택.

「옵션」 메뉴로 「윤곽」 명령어를 선택해서 모든 옵션을 선택 이것으로 커서의 좌표가 S F C편집기의 상태바에 표시됩니다.

S F C의 초기스텝은 이미 존재하고 있습니다. 최초의 Transition 의 입력을 위해 도구바 아이콘에서 대응할 아이콘을 선택합니다.

셀좌표(0,1) 을 선택해서 액션키

F4 또는 도구바 버튼을 눌러서 Transition 을 배치시킵니다.

이후 나머지 스텝과 Transition 을 프로그램 합니다.셀의 선택은 자동적으로 아래 셀에 이동하기 때문에 F 4 , F 3 키를 교대로 누르는 것으로 스텝과 Transition 은 교대로 입력가능합니다.

셀좌표(0,2).을 지정합니다. 스텝을 배치합니다.

셀좌표(0,3).을 지정합니다.Transition 을 배치합니다.

셀좌표(0,4).을 지정합니다. 스텝을 배치합니다.

셀좌표(0,5).을 지정합니다. Transition 을 배치합니다.

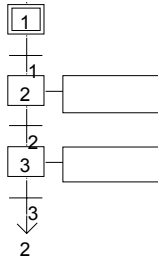


점프심벌을 세트합니다. 셀좌표(0,6).을 지정하고 도구바에서 왼쪽 아이콘을 선택하든지 액션키의 F5 을 선택합니다.

점프전에 스텝을 선택합니다.

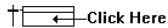
"GS2]을 선택해서[확인]버튼을 선택합니다.

아래와 같이 S F C 차트를 완성합니다.



다음은[레벨 2]프로그램의 입력을 스텝과 Transition 에 대해서
행합니다. 「편집」 - 「레벨 2 편집」 명령어를 선택합니다.

Transition[2]의 레벨 2 프로그램을 입력하기 위해서 Transition 심벌의 아래 그림
부분을 더블클릭 하든지 「편집」 - 「레벨 2 편집」 명령어를 선택합니다. :



(0,3) 셀좌표(0,3)의 더블클릭
창이 분할되어 우측에 텍스트편집기가 열립니다. 여기서
S T 프로그램 (또는 L D 프로그램) 을 입력할 수가 있습니다. Transition[2]의
조건으로서 다음의 입력을 행합니다.

RunCmd;

같은 형식의 조작을 스텝[3]에 대해서 행합니다.

(0,4) 셀좌표 (0,4)을 더블클릭
레벨 2 편집기창에 스텝[3]의 조건으로서 다음의 입력을 행합니다.

QX1_1;

(0,5) 다음으로 트랜지션[3]의 입력을 행합니다. 창을 스크롤해서 표시
시킵니다.

셀좌표 (0,5)을 더블클릭
텍스트편집기가 열립니다. 트랜지션[3]의 조건으로서 다음의 입력을 행합니다.

Not (RunCmd);

이상으로 S F C 프로그램을 종료했습니다.
수정내용도 보존해서 S F C 편집기를 종료합니다.

프로젝트에 필요한 두개의 프로그램의 작성이 완료되었기 때문에 다음은 프로그램을 실행할 수 있는 코드를 생성합니다.



어플리케이션의 생성

프로그램관리창의 「만들기」 - 「컴파일러선택」 어플리케이션 명령어를 선택해서 중간 코드를 생성합니다.

(만약 입력에러가 있다면 코드생성 중의 컴파일러가 에러를 찾아냅니다. 이 경우에는 에러메세지 : 적색표시를 더블클릭 해서 대상이 되고 있는 프로그램 수정을 행한 후에 만들기를 행합니다.)



시뮬레이션

프로그램관리 창에서 「디버그」 - 「시뮬레이션」 명령어를 선택해서 ISaGRAF Kernel 시뮬레이터를 작동합니다. 시뮬레이터 창 (가상 I/O 패널) 위에서부터 어플리케이션을 테스트 할 수 있습니다. 그 예로서는 입력 1, 2 (녹색) 를 누르면 출력 (적색) 이 켜집니다.

프로그램관리 창에서 프로그램을 더블클릭하는 것으로 프로그램이 모니터 모드로 열립니다.

시뮬레이션 디버그를 종료할때는 디버그창의 「파일」 - 「종료」 명령어를 선택합니다..

A.2 프로젝트 관리

ISaGRAF 프로젝트 관리도구의 입장에서는 ISaGRAF 그룹 **[프로젝트]** 아이콘을 더블클릭합니다.

프로젝트 관리 창이 열립니다. 프로젝트창의 윗부분은 현재의 프로젝트 리스트를 보이며, 아랫창은

선택한 프로젝트에 대한 간단한 설명이 보입니다



창 크기 조절

상하 창 사이의 구분선을 클릭해서 창의 사이즈를 상하 변경할 수 있습니다. 다만, 하부의 프로젝트 기술창은 최소한 1 줄의 텍스트가 표시됩니다.



구분선 삽입

프로젝트 사이에 구분선을 삽입 할 수가 있습니다. 이구분선에 의해서 프로젝트의 그룹을 구분할 수 있습니다. **「편집」** - **「구분선」** 명령어에 따라 선택 프로젝트 직전에 구분선이 삽입 또는 소거 됩니다.



목록에서 프로젝트 이동

리스트 중에 프로젝트의 상하 이동을 할 수 있습니다. 프로젝트를 이동시키기 위해서는 프로젝트를 선택 해서 프로젝트를 이동 끝까지 드래그 시킵니다. 이 사이에 왼쪽구석에 화살표 아이콘이 표시되어 이동 끝을 나타냅니다. **「편집」** - **「이동」** 명령어에 따라 이동도 할 수 있습니다. 구분선이 프로젝트 전에 있을 경우에는 프로젝트와 함께 구분선이 이동합니다.

A.2.1 프로젝트 생성 / 작업

프로젝트관리 창의 메뉴를 사용해서 프로젝트의 신규작성, 편집, 관리를 할 수 있습니다.



새 프로젝트

파일 - **새파일** 명령어에 따라 신규의 프로젝트를 작성 할 수가 있습니다. 공간에 새로운 프로젝트가 열립니다. 라이브러리에 등록 마친 I/O config 의 선택을 할 수

있습니다. 만약 I/O config 의 선택을 행하면 모음편집기와 I/O 접속편집기로 입출력변수가 자동적으로 설치됩니다. 프로젝트명은 다음의 룰에 따라서 입력합니다.

- 프로젝트명은 최대 8 문자
- 최초의 문자는 영문자 (a - z)
- 이후의 문자는 영문자, 숫자, _ 중에서
- 대/소문자 구분은 없음 (전각기호 입력은 할 수 없습니다.)

프로젝트가 작성되면 **편집 - 주석문** 명령어로 프로젝트 주석을 기입합니다. 이 명령어는 프로젝트의 우측에 표시됩니다.



프로젝트 메모장 편집

「프로젝트」 - 「메모장」 선택하면 해석으로 프로젝트를 설명하기 위해 텍스트편집기가 열립니다. 프로젝트를 관리차원에서 프로젝트 간의 차이의 표현상에 있어서도 프로젝트의 라이브러리에 걸쳐 활용됩니다



프로젝트 편집

파일 - 편집 명령어에 따라 선택된 프로젝트의 편집이 가능 하게 됩니다. 즉 프로그램관리 창이 열립니다. 프로그램관리 창에서 프로젝트의 모든 내용 (프로그램, 파라미터, I/O 접속, 그래픽, 컴파일러 선택....)을 관리할 수 있습니다.

「편집」 명령어 대신으로 프로젝트를 마우스로 더블클릭 하는 것으로 프로그램관리 창을 열 수 있습니다.



수정목록 (Modifications History)

「수정목록」 명령어 또는 아이콘에 따라 프로젝트에 관한 수정이력 파일의 내용을 표시 및 인쇄 할수 있습니다. 수정이력 내용의 일부 또는 전체 삭제도 할 수 있습니다. 프로젝트 수정목록파일에는 프로젝트 내의 프로그램의 어떠한 수정내용 (코드생성, 어플리케이션 다운로드 . .) 이 날짜,시간,타이틀을 포함한 형식으로 보존됩니다. 수정목록파일에는 최대 500 건의 내용이 보존됩니다.

수정목록 DialogBox 에서는 아래의 조작이 가능합니다.

- 확인**DialogBox 을 닫다.
- 인쇄** 수정이력 내용의 인쇄
- 도움말** 수정이력에 관한 도움말 화면 삭제
- 선택** 선택된 리스트를 삭제

전부 모든 리스트의 삭제
검색 입력된 문자열의 검색

문자열 검색시에는 **검색** 버튼에서 입력 파일에 검색하고 싶은 문자열을 입력합니다.
 입력 문자열은 대/소문자의 구별이 있습니다. 검색이 리스트의 마지막에 와도 계속
 끌어당겨 리스트의 맨 위에서 검색을 계속합니다.

주의: 프로젝트의 수정이력은 각각의 프로그램에 속해 있는 수정이력을 모아 놓고
 있습니다.



인쇄

「프로젝트」 - 「인쇄」 명령어에 따라 선택 가운데 프로젝트에 관한 목차부착의
 완전한 문서를 인쇄할 수 있습니다. 이 도큐먼트는 많은 섹션 (프로그램 소스코드,
 크로스 래퍼런스, 수정이력 .) 부터 성립되어 있습니다. 부분적인 문서 작성시는
 목차 가운데서 필요한 섹션 읽기를 선택해서 인쇄하는 것이 가능합니다.



암호

「프로젝트」 - 「암호설정」 명령어에 따라 선택중의 프로젝트에 대해서
 최대 15 종류의 암호를 설정할 수 있습니다. 암호에 따라 개별의 기능마다 보호를 걸
 수가 있습니다. 여기에 설정된 암호는 다른 프로젝트와 라이브러리 프로젝트에는
 모두 무효가 됩니다.

A.2.2 복수 프로젝트 그룹 관리

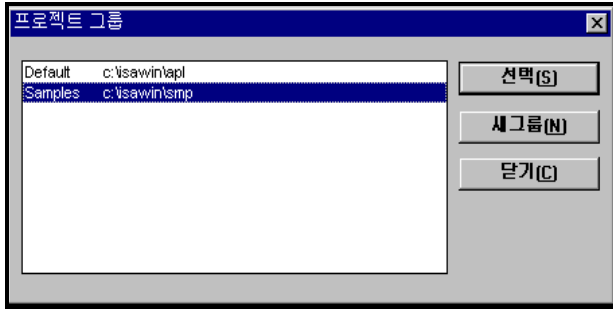
ISaGRAF 프로젝트는 어떤 하나의 디렉토리에 모든 프로젝트 (각 프로젝트는
 하나의 서브디렉토리)가 보존되어 있습니다. 프로젝트그룹은 그 이름에 따라서
 식별됩니다. 기본으로 아래 두개의 프로젝트그룹이 만들어집니다.

"Default" "ISAWINAPL" : 기본 작업용 서브디렉토리
 "Samples" "ISAWIN\SMP" : ISaGRAF Workbench 에 포함된
 샘플 프로젝트용 서브디렉토리

선택 되어 있는 프로젝트 그룹명은 도구 바상에 표시되어 있습니다.
 이 버튼의 선택은 다른 프로젝트그룹으로 바꿀 수 있습니다.



역시 **파일 - 프로젝트 그룹설정** 명령어에 따라 다음의 DialogBox 가 열려 기존의 프로젝트그룹의 선택과 신규 프로젝트그룹의 작성을 할 수 있습니다.



프로젝트 관리리스트의 내용을 다른 프로젝트그룹으로 대처하기 위해서는 목적프로젝트그룹리스트를 선택해서 **선택** 버튼을 선택합니다. 프로젝트그룹을 선택해서 더블클릭을 함으로서도 선택 할 수 있습니다. **[새그룹]** 버튼에 따라 신규의 프로젝트그룹을 정의 할 수 있습니다. 이때 프로젝트그룹명을 존재하고 있는 서브디렉토리명으로 할당하기도 하고 신규 서브디렉토리의 작성을 할 수 있습니다.

주의 : 어떤 프로젝트의 창 (예를 들면 프로그램편집기 모음편집기) 이 열려 있을 때는 다른 프로젝트그룹으로의 대체와 신규 프로젝트그룹의 작성은 할 수 없습니다..

A.2.3 옵션

옵션 메뉴의 **[프로젝트관리자열어두기]** 가 체크되어 있을 경우 프로젝트를 열어도 프로젝트관리 창은 열려 있는 체이지만 체크를 빠뜨린 경우에 프로젝트를 열면 프로젝트관리 창은 자동적으로 닫힙니다.

옵션 메뉴에서 **[도구바보기]**가 체크 되어 있는 경우는 도구바 버튼이 표시됩니다.

옵션 메뉴의 **[글꼴]** 명령어에 따라 프로젝트기술과 ISaGRAF 텍스트편집기의 글꼴을 선택할 수 있습니다.

A.2.4 도구

도구 메뉴 에서 다른 ISaGRAF 유틸리티를 작동 시킬 수 있습니다.

도구 - 보관함 - 프로젝트 에 따라 프로젝트를 백업/복구를 할 수 있습니다.

도구 - 보관함 - 일반데이터 에 따라 모든 프로젝트에 공통적인 데이터 (예를 들면 common 정의워드)의 백업 / 복구를 할 수 있습니다.

도구 - 라이브러리 에 따라 라이브러리 관리 창이 열립니다.

「 도구 」 - 「 IL 프로그램 가져오기 」 명령어에 따라 PLC Open 으로 규정된 파일형식의 IL 프로그램의 텍스트파일이 하나의 프로젝트로서 가져올 수 있습니다..

A.3 프로그램 관리

프로그램관리 창은 프로젝트(어플리케이션)으로 사용되는 **모든 프로그램(modular, 프로그램유닛)**을 표시합니다. 여기에서는 프로젝트 구성의 디자인, 프로그램의 편집기 작동, 컴파일러와 디버그의 작동 등이 행해집니다. 이 창은 어플리케이션을 개발할 경우에 Workbench의 중심적인 창입니다. 프로젝트관리 창의 편집명령어 (또는 프로젝트명을 더블클릭) 로 열 수 있습니다.

A.3.1 프로젝트 구성요소

프로젝트를 구성하고 있는 요소는 프로그램입니다. 프로그램은 제어의 내용을 논리적으로 기술한 것입니다.

전역변수 (예를들면 I/O 변수) 는 어플리케이션 전체의 어떠한 프로그램에서 사용할 수 있습니다. 지역변수는 현재의 프로그램에서만 액세스 됩니다. 프로그램관리 창은 프로젝트내의 프로그램의 계층적인 구성, 연관성을 나타내고 있습니다. 프로젝트를 구성하는 구성요소 는 프로그램이라 불립니다. 프로그램은 논리적인 처리를 합니다. 프로그램은 계층적으로 구성된 3 종류의 섹션 (Begin, Sequence, End) 으로 나뉘져 있습니다. 어떤 섹션에서도 계층의 좌측에 오는 프로그램은 탭레벨 프로그램이라 불립니다.

☐ Top 레벨 프로그램

3 개의 섹션에 있었어도 탭레벨 프로그램은 계층투어의 좌단에 위치합니다. Sequence 는 항상 활성상태(액티브)가 됩니다. 다음의 순으로 실행됩니다.

- (입력 집어넣기)
- **Begin** 섹션의 탭레벨 프로그램 실행
- **Sequence** 섹션의 탭레벨 프로그램 실행
- **End** 섹션의 탭레벨 프로그램 실행
- (출력 갱신)

Begin 섹션은 항상 사이클의 최초에 실행됩니다. 같은 모양으로 End 섹션은 항상 사이클의 마지막에 실행됩니다.

Begin, End 섹션의 탭레벨 프로그램은 **SFC** 또는 **FC 언어**로 기술할 수 없습니다. 역으로 **Sequence** 섹션의 탭레벨 프로그램은 반드시 **SFC** 또는 **FC 언어**로

기술 되어야 합니다. 모든 섹션으로 어느 프로그램도 하나 다음의 하위프로그램을 가질 수 있습니다.

☐ **Functions / function Blocks**

Function 섹션과 **FunctionBLOCKS 섹션**은 어떤 섹션의 어떤 프로그램에서도 호출 할 수 있는 서버프로그램이 됩니다. Function-Function BLOCKS 섹션의 프로그램은 SFC/FC 언어에서는 기술할 수 없습니다.

Function 은 복수의 입력과 하나의 출력을 가집니다

Function 중에서는 변수의 내용은 매회 없어져 버립니다. 즉 **FunctionBLOCKS** 과는 다릅니다.

FunctionBLOCKS 은 입력과 내부 (hidden) 의 static data 로부터 출력합니다. 이 내부의 static data 는 **FunctionBLOCKS** 이 부르면 매번 복사 됩니다. (즉 사용할 **FunctionBLOCKS** 에 매번 설치가 됩니다.)

☐ **서브-프로그램**

하위프로그램은 하나의 Father 프로그램 에서 호출될 것인가 **Function, Function BLOCKS 섹션**으로 적힌 프로그램으로서 관리되고 있는가 두가지의 경우가 있습니다. **Function, Function BLOCKS** 섹션 내에 하위프로그램은 프로젝트 중의 어떠한 프로그램에서도 호출 할수 있지만 Function 섹션에 없는 하위프로그램은 그 Father 프로그램 (하나로 한정) 으로 밖에 호출되지 않습니다. Father 프로그램의 실행은 하위프로그램의 처리가 완료할 때까지 정지해 있습니다.

어떤 섹션도 하나 이상의 하위프로그램을 가질 수 있습니다. 하위프로그램 오너는 하나의 Father 프로그램 이 됩니다. Function 섹션의 프로그램은 더구나 하위프로그램을 호출 할 수 있습니다. 하위프로그램은 **지역변수, 지역정의 워드**를 가질 수 있습니다.

☐ **Child SFC / FC 프로그램**

child SFC 프로그램은 Father 프로그램에서 작동, 정지, 동결이 관련된 병렬처리 프로그램입니다.

Father 프로그램과 child 프로그램도 함께 **SFC 언어**로 기술되어야 합니다.

- Father SFC 프로그램이 child 프로그램이 SFC 프로그램을 작동하면 **SFC token**을 child 프로그램의 초기 스텝에 설정합니다. 이 명령어는 **GSTART** 로 합니다.
- Father SFC 프로그램이 child SFC 프로그램을 정지 시키면 child 프로그램에 존재하고 있던 SFCtoken 이 모두 클리어 됩니다. 이 명령어는 **GKILL** 로 합니다.

- Father SFC 프로그램이 child SFC 프로그램을 **동결**시키면 child 프로그램에 존재하고 있던 SFC token 전부를 제거하지만 이때 token 의 장소를 모두 기억해 놓습니다. 이 명령어는 **GFREEZE** 로 합니다.
- Father SFC 프로그램이 동결된 child SFC 프로그램을 **재작동**시키면 child 프로그램이 동결 되었을 때 제거된 SFCtoken 을 전부 원래의 장소로 되돌립니다. 이 명령어는 **GRST** 로 합니다.

Sequence 섹션의 FC 언어프로그램은 다른 FC 하위프로그램을 컨트롤할 수 있습니다. FC 하위프로그램이 실행되고 있을 때는 FatherFC 프로그램의 처리는 정지해 있습니다. 따라서 FatherFC 프로그램과 거기서 읽어낼 FC 하위프로그램은 동시 실행은 되지 않습니다.

☞ 프로그램과 서브-프로그램 연결:

하위프로그램과 child 프로그램은 Father 프로그램과는 계층 구조로 연결되어 있습니다. Father SFC 프로그램과 child SFC 프로그램과의 링은 화살표부착 라인으로 접속됩니다. 이 표시는 프로그램이 **병렬**처리된 것을 의미합니다.

☞ 프로그램 언어

각 프로그램은 하나의 언어로 기술됩니다. 프로그램 신규 작성시에는 선택한 언어는 변경할 수 없습니다. 단지, **FBD** 는 **LD** 를 포함 시킬 수 있습니다. 또한 **LD** 프로그램은 Function BLOCKS 호출을 포함 시킬 수 있습니다. 그래픽언어로서는 SFC(Sequential Function Chart), FC (Flow Chart), FBD (Functional BLOCKS Diagram), LD (Ladder Diagram)가 있습니다. 텍스트언어로서는 ST (Structured Text), IL (Instruction List)가 있습니다. SFC,FC 언어는 Sequence 섹션에서의 Father 프로그램, child 프로그램

(FC에서는 하위프로그램)으로서 사용 됩니다. 프로그램언어는 프로그램관리 창에 프로그램명과 더불어 다음의 아이콘에서 표시됩니다. 다음에서는 프로그램언어를 나타낼 아이콘을 표시 합니다.

.....		SFC Sequential Function Chart
.....		FC Flow Chart
.....		FBD Functional BLOCKS Diagram
.....		LD Ladder Diagram (entered with QuickLD editor)
.....		ST Structured Text
.....		IL Instruction List

A.3.2 프로그램 관리

파일, **도구** 메뉴에는 프로그램을 작성, 갱신, 변경할 다음의 명령어를 포함시키고 있습니다



새 프로그램

파일 - **새파일** 명령어에 따라 프로그램을 지정한 섹션에 신규 작성할 수 있습니다.

우선 프로그램명을 다음의 룰에 따라 입력합니다.

- 프로그램명은 최대 8 문자
- 최초의 문자는 영문자(a - z)
- 이후의 문자는 영문자, 숫자,중에 아무거나
- 대문자, 소문자의 구별 없음 (전각입력은 할 수 없습니다)

다음 사용할 프로그램언어를 선택합니다.

- **SFC**
- **FC**
- **FBD**
- **LD**
- **ST**
- **IL**

마지막에 프로그램 스타일 / 섹션을 선택합니다.

- **Begin** Begin 섹션의 탑레벨
- **Sequential** Sequential 의 탑레벨
- **End** End 섹션의 탑레벨
- **Fnc** Function 섹션
- **Fnc Blk** Function BLOCKS 섹션
- **...의 하위** 기존의 프로그램 하위프로그램, Child-SFC 프로그램

이상의 모두 5 개의 섹션 중에서 하나를 선택하는 것으로 **Begin End Sequential Function Function BLOCKS** 의 어떠한 섹션의 탑레벨 에도 프로그램 위치를 정할 수 있습니다. 이것 이외엔 작성할 프로그램이 **child SFC 프로그램**, **FC 하위프로그램**, 또는 하위프로그램 일때 선택합니다. 이경우는 **Father 프로그램명**을 선택 입력 할 필요가 있습니다. **sequence 프로그램의 탑레벨은 SFC / FC 언어일 필요가** 있습니다.

SFC / FC 언어 프로그램은 Begin End 섹션의 탭레벨, 또는 그 하위프로그램에는 사용할 수 없습니다.



주석삽입

ISaGRAF에서는 프로젝트 중에 각 프로그램에 대해서 명령어를 입력할 수 있습니다.

이명령어는 프로그램 옆의 작은 폰트로 표시됩니다.명령어의 입력에는 **파일 - 주석문**

명령어를 선택합니다. 여기서 새로운 명령어의 입력, 또는 수정을 합니다.



프로그램 편집

「파일」 메뉴의 「편집」 메뉴에 따라 프로그램의 내용을 수정하기 위해 편집기 창을 열 수가 있습니다. ISaGRAF는 프로그램언어를 받아들여 편집기를 엽니다. 프로그램의 편집은 개별창에 표시됩니다. 복수 편집기창을 동시에 여는 것도 가능합니다. 프로그램이 선택되어져 있는 상태에서 **ENTER** 키 입력, 또는 프로그램명의 마우스에 의한 **더블클릭**에 따라 프로그램 편집창을 열수 있습니다.



수정 이력 파일 편집

[수정이력] 명령어에 따라 각 프로그램에 부속할 수정이력파일을 편집할 수 있습니다. 편집중의 프로그램을 종료 할 때 수정이력메모를 기입하기 위해, DialogBox가 열리기 때문에 필요한 메모 기입할 수 있습니다. 또한 코드작성, 검증을 행한 결과도 수정이력파일에 자동적으로 기입됩니다.

역시 각 편집기에서의 「**옵션**」 - 「**입기장 갱신**」 명령어가 set 되지 않는 경우는 편집중의 프로그램 종료 시에 DialogBox는 열리지 않습니다.



변수목록

[변수목록] 메뉴에 따라 프로젝트로 사용된 변수와 정의워드의 모음파일 편집을 할 수 있습니다. 변수와 정의워드의 범위에 따라 전역변수, 지역변수가 있고 지역변수는 선택중의 프로그램에서만 액세스 됩니다. 전역변수는 프로젝트 전체에서 액세스할 수 있습니다. 정의워드는 프로그램에서 사용된 문자열 (숫자) 의 바뀌 놓음을 의미합니다. 예를 들면 정수, 키워드, 논리치 상태를 다른 언어 (워드) 로 바뀌 놓을 때 사용합니다



파라미터

「파일」 - 「파라미터」에 따라 선택된 하위프로그램 또는 Function, Function BLOCKS 섹션의 프로그램의 입력/출력 파라미터를 정의할 수 있습니다. 선택되어 있는 프로그램이 Begin, End 섹션의 탭레벨 Father 프로그램 또는 SFC 프로그램의 경우에는 「파라미터」 명령어는 무효가 됩니다. Function, 하위프로그램, F B는 최대 3 2의 입출력파라미터를 가질 수 있습니다. Function과 하위프로그램은 하나의 출력파라미터 밖에 가질 수 없습니다. 그래서 그파라미터명은 그 Function과 하위프로그램의 이름과 같아야 합니다. Function BLOCKS에서는 출력 파라미터를 복수로 가질 수 있습니다. 파라미터 Dialog Box의 상부의 창에서는 파라미터 입력, 출력파라미터가 표시되고 있습니다. 하부의 창에서는 현재 선택되어 있는 파라미터에 관한 상세정보가 표시되어 있습니다. 파라미터의 데이터 타입으로서 ISaGRAF가 지원하는 어떠한 데이터 타입으로도 사용할 수 있습니다.

출력파라미터는 파라미터 리스트의 마지막에 있을 필요가 있습니다. 파라미터명에서는 다음의 룰이 있습니다.

- 파라미터명 최대길이는 반각 16 문자
- 최초의 문자는 영문자(a-z)
- 이후의 문자는 영문자, 숫자 중에 아무것이나
- 대문자, 소문자 구별 없음

하나의 Function과 Function BLOCKS 내에서 동일한 파라미터명은 허용하지 않습니다. 동일한 파라미터명은 다른 Function에서는 취급할 수 없습니다. 출력파라미터명과 Function명, Function BLOCKS 명과는 동일할 필요가 있습니다.

[삽입]버튼은 새로운 파라미터를 선택중의 파라미터앞에 삽입합니다. **[삭제]**버튼은 선택중의 파라미터를 삭제합니다. **[정렬]**버튼은 자동으로 파라미터의 순번을 바꾸어 나열해서 출력 파라미터가 마지막에 오도록 합니다.



이동

[파일] - [이동/이름 바꾸기] 명령어에 따라 프로그램명의 변경과 프로그램을 다른 섹션에 이동할 수가 있습니다. 다만 프로그램언어는 변경할 수 없습니다. 이 명령어를 실행하면 프로그램의 신규작성시와 같은 DialogBox가 열립니다. 선택된 프로그램의 속성이 선택 표시됩니다. 여기서 프로그램명의 변경, 또는 이동시키고 싶은 섹션, Father 프로그램의 선택을 할 수 있습니다.

[파일] - [프로그램정렬] 에 따라 동일레벨의 프로그램 사이에서는 순번 바꿔넣기를 할 수 있습니다. 예를 들면 탑레벨의 프로그램 이 선택되어 있을 때는 탑레벨 사이에서 프로그램 의 순번을 바꿔넣기 할 수 있습니다. 하위프로그램의 레벨에서는 동일 Father 프로그램을 가진 프로그램 사이에서 순번 바꿔넣기가 가능합니다.

프로그램의 순서바꿔넣기 DialogBox 에서 **위로이동** 또는

[아래로이동]버튼을 선택하는 것으로 프로그램순을 바꿔 넣습니다.



복사

[파일] 메뉴의 **[복사]** 명령어에 따라 선택되어져 있는 프로그램을 다른 이름의 프로그램으로 복사할 수 있습니다.

복사 명령어를 실행하면 프로그램 신규 작성시 DialogBox 가 열립니다. 선택된 프로그램의 속성에서 선택 표시되어 있습니다. 여기서 복사 전에 프로그램명의 입력, 또는 복사 전에 Father 프로그램의 선택을 할 수 있습니다. 복사 전의 프로그램이 존재 하지 않을 경우는 지정된 장소에서 프로그램이 복사되어, 이미 프로그램이 존재하고 있는 경우는 겹쓰기가 됩니다. 프로그램 복사에 따라 프로그램에 첨가된 모음파일 내용도 정리되어 복사됩니다. **[확인]**버튼에 따라 프로그램이 복사됩니다. **[취소]** 버튼에 따라 프로그램은 복사 되지 않고 Dialog Box 는 닫힙니다.

「파일」 메뉴의 「**다른프로젝트에복사**」에 따라 선택 되어 있는 프로그램을 동일명인 채로 다른 프로젝트에 복사할 수 있습니다. child 프로그램, 하위프로그램도 정리되어 복사됩니다. 복사 전의 프로젝트에서 동일명의 프로그램 및 하위프로그램 이 존재하고 있지 않는 것을 확인해 둘 필요가 있습니다. 이 명령어에서는 프로그램 은 덮어쓰기 되지 않습니다. 이 명령어에 따라 프로그램 에 첨가시켜 변수목록파일 (지역변수, 지역정의의워드만) 도 포함시켜 복사됩니다.



삭제

「파일」 - 「**삭제**」 명령어에 따라 선택 되어 있는 프로그램을 삭제할 수 있습니다. Child 또는 하위프로그램을 가진 Father 프로그램을 선택해서 삭제할 수 없습니다. 이 경우에는 최초의 하위프로그램을 삭제 한 후 나중에 Father 프로그램을 삭제할 필요가 있습니다. 프로그램을 삭제 할 때는 모음파일로 정의 되어 있는 지역한 정의내용도 포함해서 삭제됩니다.



라이브러리 에서 function / function Blocks 가져오기

「도구」 - 「라이브러리 에서 가져오기」에 따라 I E C로 적힌 Function 또는 F B 라이브러리와 현재의 프로젝트사이에서 프로그램정보 교환이 가능합니다. 라이브러리 내에서 선언되어 있는 지역변수, 정의 도 함께 가져오기 됩니다. 라이브러리 내에서 이미 검증, 컴파일러선택의 경우는 Obj-code 도 포함해서 copy 됩니다. 이 경우에는 열려 있는 프로젝트 내에서의 컴파일러는 생략됩니다. 이 명령어에 따라 I E C Language 로 적혀진 라이브러리에서 **Function, F B 섹션**의 탭레벨에 복사할 수 있습니다. 가져오기된 함수는 프로젝트의 계층구조 가운데 소정의 섹션에 「**이름변경 / 이동**」명령어에 따라 옮길 수 있습니다. 이미 존재하는 Function 과 F B에서의 겹쓰기를 기피하기 때문에 가져오기된 라이브러리는 이름변경이 필요한 경우가 있습니다. 이런 경우에는 Function 의 복귀치의 변수명도 잊지 말고 Function 명에 맞춰 둘 필요가 있습니다.



라이브러리로 function / function Blocks 보내기

「도구」 - 「라이브러리로 보내기」명령어에 따라 오픈 중의 프로젝트의 **Function** 또는 **F B 섹션** 내의 Function 에 대응했던 라이브러리로 옮길 수 있습니다. Function 내에서 선언된 지역변수, 정의워드는 정리되어 옮겨집니다. 보내기된 Function, F B는 라이브러리 관리 창에서 Compile 할 필요가 있습니다. 보내기된 Function, F B는 전역변수를 사용할 수 없습니다.

A.3.3 TIC 코드 [만들기]

「만들기」 메뉴에는 프로젝트 실행코드를 생성하기위한 명령어와 그 옵션이 포함되어 있습니다.

상세한 것은 「코드생성 사용법」의 장을 참조 바랍니다.



만들기

「만들기」 - 「만들기」에 따라 설정종료의 컴파일러옵션에 의해 프로젝트코드가 생성됩니다. 프로젝트 코드 생성시에는 또한, 문법체크가 되어 있지 않은 코드에 관해서는 검증도 됩니다. Increment 컴파일러를 위해 이미 컴파일러 된 코드에 관해서의 다시컴파일 을 실행하지 않습니다.



검증

「만들기」 - 「검증」에 따라 프로그램관리 창에서 선택되어 있는 프로그램 문법 검사, 검증을 할 수 있습니다. 만약 에러가 검출 되지 않았던 경우는 코드 생성시에 내용과 모음에 변화가 없는 한 재차 프로그램 검증 되는 일은 없습니다..

**시뮬레이션**

「만들기」 - 「Touch」에 따라 각 프로그램에 수정이 첨가된 것을 시뮬레이션 다음회의 코드 생성시에는 모든 프로그램이 검증, 컴파일됩니다

**run-time 옵션**

「만들기」 - 「프로그램실행시간선택」에 따라 중간코드 생성옵션 및 run-time 우선순위의 설정을 할 수 있습니다. 다음에서 각각의 Priority 의 설명을 합니다. 상세한 것은 「만들기사용법」을 참조 바랍니다.

**컴파일러 옵션**

「만들기」 - 「컴파일러 선택」명령어에 따라 ISaGRAF 의 코드 생성기가 타겟 실행코드의 종류 선택 및 최적화를 할 수 있습니다. 상세한 것은 「코드 생성 사용법」을 참조 바랍니다

**리소스 정의**

「만들기」 - 「리소스」명령어에 따라 Target 실행코드와 함께 사용할수 있는 데이터 (혹은 참조할 파일) 을 리소스파일로 정의해 다운로드할 수 있게 됩니다. 상세한 것은 리소스 정의파일의 설정 포맷 등 「코드생성 사용법」을 참조 바랍니다

A.3.4 기타 ISaGRAF 도구

「프로젝트」 메뉴에는 ISaGRAF 도구를 실행하기 위한 명령어가 포함되어 있습니다. 상세한 것은 각 도구의 사용법을 참조 바랍니다.

**I/O 변수 연결**

「도구」 - 「I/O 연결」명령어에 따라 I/O 연결편집기 를 열 수 있습니다. 여기서는 선언되어 있는 I/O 변수를 대응하는 I/O 보드와 mapping 을 합니다. 논리적인 변수를 물리적으로 할당하게 됩니다.

**크로스 래퍼런스 편집기**

「프로젝트」 - 「크로스 래퍼런스」명령어에 따라 프로젝트의 크로스 래퍼런스 편집기가 작동합니다. 크로스 Reference 로 프로그램의 소스코드 중의 모든 변수가 사용되어지고 있는 장소를 볼 수 있습니다. 특정 변수가 사용되어지고 있는 장소에 유효하게 사용할 수 있습니다.

**프로젝트 설명 추가**

「프로젝트」 - 「메모장」에 따라 프로젝트의 기술용 텍스트의 편집을 할 수 있습니다. 또한 이 명령어는 프로젝트관리 창에서도 작동할 수 있습니다.

☐ 문서 인쇄

「프로젝트」 - 「인쇄」에 따라 열려 있는 프로젝트에 관한 도큐먼트를 인쇄할 수 있습니다. 이 도큐먼트에는 프로그램, 변수, 파라미터 등이 포함됩니다. 사용자는 필요한 인쇄항목을 선택해서 도큐먼트를 인쇄할 수 있습니다. 또한 「프로젝트관리」에도 같은 명령어가 있습니다.

☐ 수정 이력

이 명령어는 DialogBox를 열어 프로젝트의 수정 이력을 보여 줍니다. 자세한 것은 [프로젝트 관리]를 참고 하시기 바랍니다.

A.3.5 [도구] 메뉴에 사용자 가이드 추가

도구 메뉴에 사용자 가이드를 추가할 수 있습니다. \ISAWIN\COM\ISA.MNU 텍스트파일을 편집하는 것에 따라 사용자 가이드를 추가할 수 있습니다. 최대 10 개의 명령어를 추가할 수 있습니다. 명령어에 대한 명령어는 ; 으로 시작될 행에 기입할 수 있습니다. 각 명령어는 아래에 표시할 2 행의 텍스트라인으로 구성됩니다. 다음의 문법에 따릅니다.

M=메뉴에 표시될 문자열

C=명령어

명령어라인에는 DOS 와 Windows 의 실행모듈을 인수를 합쳐서 기술 가능합니다. 다를 인수로서 "%A" 는 열려 있는 프로젝트명, "%P"는 선택종의 프로그램명 입니다. 다음의 예에서는 메모장에서 선택된 프로그램을 편집할 경우 명령어라인이 됩니다.

M=메모장에서 편집

C=Notepad.exe \isawin\apl\%A\%P.lsf

A.3.6 시뮬레이션 - 디버깅

「디버거」 메뉴에는 ISaGRAF 어플리케이션 시뮬레이션, 온라인 디버깅에 필요한 명령어가 포함되어 있습니다



시뮬레이션

디버그 - 시뮬레이션 에 따라 디버그 창이 시뮬레이션 모드로 작동됩니다. 이것과 동시에 Kernel 시뮬레이터라고 불리는 창이 작동됩니다. Kernel 시뮬레이터 창과는 어플리케이션으로 입출력 I/O 변수가 사용되고 있고, 이것들이 접속되어 있는 경우가 상 I/O 패널을 가리킵니다. 시뮬레이션을 행하기 전에는 시뮬레이션용의 프로젝트 코드가 생성되어 있을 필요가 있습니다 (코드 생성컴파일러옵션으로 시뮬레이션용 코드 선택이 필요)

디버그 창이 열려질 때 프로그램관리 창은 일단 닫힙니다. 시뮬레이션 창이 열린 후에 프로그램관리 창은 다시 디버그 창으로 열립니다. 또한 오픈 중의 프로젝트내의 프로그램편집기 코드생성 창 I/O 접속 창 등이 열려 있을 때는 디버그창은 열리지 않습니다. 디버그 전에는 이것들을 전부 닫을 필요가 있습니다. 이 명령어는 프로그램 관리 창에도 포함되어 있습니다.



온라인 디버그

디버그 - 디버그 명령어에 따라 디버그 메인 창이 작동합니다. 온라인 디버그 전에는 Target 용의 어플리케이션 코드가 생성 되어 있을 필요가 있습니다. 디버그 창이 열려질 때에 프로그램관리 창은 일단 닫힙니다. Workbench 의 디버그가 Target 어플리케이션과 통신이 되었을 때 다시 디버그모드로 열립니다. 또한 오픈 중의 프로젝트 내의 프로그램편집기 코드 생성기창, I/O 접속창 등이 열려 있을때는 디버그 창은 열리지 않습니다. 디버그 전에는 이것들을 전부 닫을 필요가 있습니다. 프로그램관리 창도 포함됩니다.

시뮬레이션과의 차이는 Kernel 시뮬레이션창이 열리지 않는 점입니다. 이것 이외에는 같은 시뮬레이션기능을 가집니다.



Link 설정

「디버그」 - 「타겟연결」에 따라 ISaGRAF Workbench 와 Target 사이의 통신파라미터 설정을 합니다. 타겟연결 DialogBox 로 다음의 통신파라미터를 설정합니다. 이들 설정내용은 접속될 ISaGRAF Target 쪽의 설정내용과 일치할 필요가 있습니다.

Target slaveNO 는 동시에 실행중의 ISaGRAFTarget 의 ID 값번호가 됩니다. 1 ~ 255 의 수치를 입력할 수 있습니다. Target slave No 의 설정에 관해서는 Target 매뉴얼을 참조 바랍니다.

통신보드는 Workbench 디버그와 ISaGRAF Target 과의 통신 포트를 설정합니다. 표준으로서 COM1~4, Ethernet 가 있습니다.

Ethernet 는 winsock ver 1.1 을 사용한 TCP/IP 통신이 됩니다. Ethernet 통신은 SunPC-NFS Ver5.0 으로 동작확인 종료입니다.

Time Out 은 디버그와 Target 간의 통신타임아웃 조건을 설정합니다. **Time Out** 시간은 디버그 질문에 대응할 응답까지의 간격을 가리킵니다. 단위는 초입니다.

Retry 는 디버그가 Target 간에 통신을 할 때 통신에러를 검출하기 전의 통신트라이 횟수.



Serial link 설정

통신보드에 COM 1 - 4 가 선택된 때에[설정]버튼을 선택하면 시리얼링 통신파라미터 설정 Dialog 가 열립니다. 여기에서는 다음의 파라미터를 설정할 수가 있습니다.

Baudrate 디버그와 Target 간의 통신속도(bit/sec) 를 나타냅니다. 최대 19200Baud 가 설정 가능 합니다.

Parity..... 통신 페리티로 짝수, 홀수, 없음 으로 선택 가능합니다.

Format 데이터 set 수와 스폿트 set 수를 지정합니다. 데이터 set 수는 7,8 중에서, stopset 수는 1,2 중에서 선택 가능 합니다.

Flow Control 제어의 선택을 합니다. ISaGRAF Workbench 는 하드웨어 handshake 를 가능케 하기위해 CTS, DSR 신호를 제어하는 것도 가능합니다.



Ethernet link 설정

통신포트에 Ethernet 가 선택된 때에는[설정]버튼에 따라 TCP/IP 통신을 위해 Ethernet 주소 및 Ethernet 보드 선택의 DialogBox 가 열립니다. 이 필드들은 소켓 인터페이스에 따라 정의종료 표준 형식 입니다. Workbench 는 TCP/IP 통신에 winsock.dll(ver1.1)을 사용합니다. Workbench 환경에는 이 소켓이 바르게 설치되어 있을 필요가 있습니다. 기본 포트번호는 1100 입니다.

A.4 SFC 편집기 사용법

SFC 언어는 Sequential 한 프로세스 동작을 기술 할 때 사용합니다. 프로세스스텝을 표현할 부분과 스텝사이의 transition 을 표현할 부분의 그래픽에서 성립되고 있습니다. SFC 언어는 IEC 1 1 3 1 - 3 표준이 되는 부분입니다. 기타 언어는 스텝 내에 액션을 기술하기 위해 또는, transition 의 논리적인 조건을 기술하기 위해 사용되는 경우가 있습니다. SFC 언어편집기는 차트를 작성할 그래픽편집기 부분과 ST 언어에 의한 텍스트 프로그래밍할 부분을 모두 처리 할 수 있습니다. 그래서 액션부분 transition 조건부분의 프로그램도 연속적으로 할 수 있습니다.

A.4.1 SFC 언어 메인 토픽

S F C는 Sequential 한 프로세스를 표현하기에는 최적의 언어입니다. S F C 언어는 프로세스의 사이클을 연속한 **스텝**을 **transition** 에 따라 구분되어 구성되어 있습니다. 상세한 것은 ISaGRAF Technical Reference 메뉴얼 S F C 언어를 참조해 주세요
S F C Component 사이는 **방향을 가진 라인**으로 접속됩니다. Default 라인의 방향은 **위에서 아래로 (↓)** 입니다

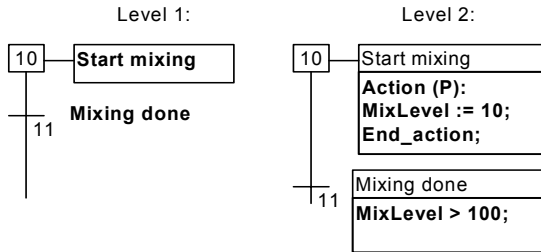
	시작 스텝
	스텝
	Transition
	점프
	매크로 스텝
	매크로 시작 스텝
	매크로 종료 스텝

SFC 언어에 따라 프로그램은 2 개의 레벨로 나뉘져 있습니다.

레벨 1에서는 그래픽차트를 표시합니다. step 과 transition 에는 Reference no 와 명령어가 attach 되어 있습니다.

레벨 2에서는 **S T 언어(transition 조건기술에는 L D언어도 가능)또는 I L 언어**에 따라 프로그램으로 스텝 내의 액션 또는 transition 에 attach 되어 있는 조건 기술을 합니다.액션과 transition 은 기타 언어(LD, FBD, ST, IL)로 적혀진 **서브프로그램**을 참조할수 있습니다.

레벨 1, 레벨 2 의 프로그램의 예를 아래에서 가리킵니다.

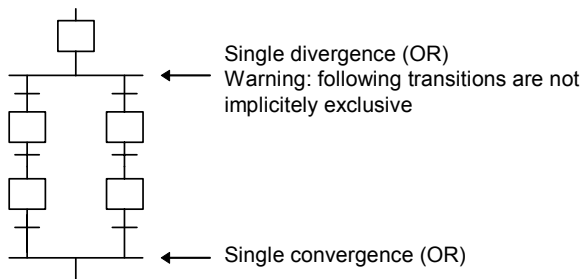


레벨 2 의 프로그램에서는 스텝 내의 액션프로그램에는 S T , I L 언어를 사용합니다. Transition 조건기에는 S T , I L 이외에 L D 언어 (QuickLD 편집기를 사용) 를 사용할 수 있습니다.

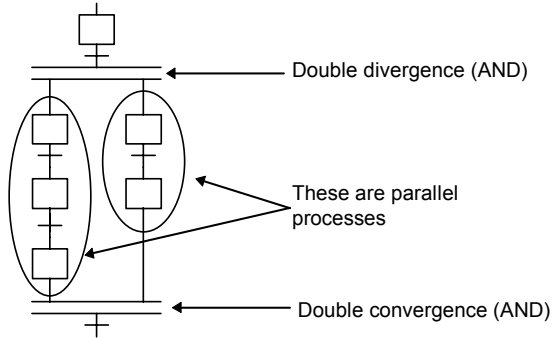
분기와 결합

분기와 결합은 복수스텝과 transition 을 연결하는 다중접속 입니다.

선택 (OR) 분기에서는 몇 개의 패스 중에서 어떤 하나의 선택이 행해져 실행됩니다.



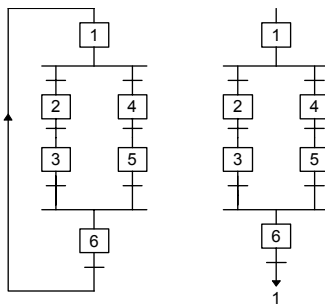
병렬 (AND) 분기는 parallel (병렬) 처리를 의미합니다.



점프 스텝

SFC 편집기에서는 컴퍼넌트 간의 링은 위에서 아래 방향이 됩니다. 단, 스텝으로 점프할 경우는 상향 링이 됩니다.

아래 그림은 같은 의미를 가집니다.

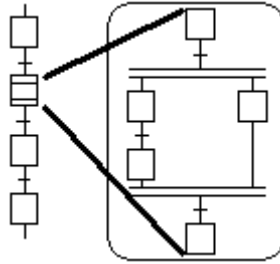


병렬 (AND) 결합 때를 제외하고 transition으로의 점프는 금지되어 있습니다.

매크로 스텝

매크로스텝은 독립해서 존재하고 있는 스텝과 transition 집단을 가리킵니다.

매크로스텝은 개시시스템으로 시작되고, 종료시스템으로 마칩니다.



매크로스테프는 동일 SFC 차트 내에 적혀 있을 필요가 있습니다. 매크로스테프 개시스텝번호와 매크로스테프 심벌은 같은 **참조번호** 이어야 할 필요가 있습니다. 호출된 매크로스테프는 더욱이 다른 매크로스테프 심벌을 지닐 수도 있습니다.

A.4.2 SFC chart 입력

SFC chart 를 그릴 때는 key component 만을 배치하는 것만으로 잘 접속라인으로서 수평-수직라인은 자동적으로 그려집니다.

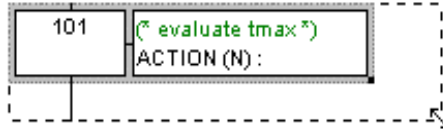
SFC component 를 차트상에 배치하기 위해서는 커서를 목적장소에 이동해서 편집기 도구 바에서 목적타입의 component 를 선택합니다. 심벌이 목적장소에 삽입됩니다. 아래에서 function key 입력을 사용하는 것도 가능합니다

- F2: 초기스텝 삽입
- F3: 단일스텝 삽입
- F4: transition 삽입
- F5: 스텝에 점프 삽입
- F6:  F7: 선택(OR) 분기 또는 결합 삽입/ branch 추가
- †F6:  †F7: 병렬(AND) 분기 또는 결합 삽입/ branch 추가
- F8: 매크로스테프 삽입
- F9:  †F9: 매크로스테프 내용 시작 또는 종료스텝 삽입

("†" 심벌은 + SHIFT 키를 의미합니다)

편집용 **grid** 는 배치된 심벌 장소를 가리킵니다. grid 표시가 있을 때 최초 SFC 차트와 서버퍼즈 레이아웃 구성이 용이합니다. **옵션 - 윤곽** 에 따라 grid 의 표시/비표시가 가능합니다.

SFC 편집기는 항상 현상에 선택되어 있는 장소표시가 좌표형으로 표시됩니다. 선택되어 있는 Cell 위치는 gray 표시됩니다. 선택되어 있는 Box 오른쪽 코너를 마우스로 드래그하는 것으로 셀 사이즈를 자유로이 변경할 수 있습니다. X/Y 비율을 변경하는 것도 가능합니다..



F6: *F6: 분기 또는 결합

분기와 결합 라인을 그을 경우는 **좌에서 우로** 라인을 component 배치해 갑니다. 우선 **좌의 branch** 장소를 아래의 어떠한 도구 바 버튼으로 선택한 후에 차트 내에서 장소를 선택합니다

F6: *F6: F7: *F7: 분기에 branch 추가

각 branch 의 **시작**와 **마지막**에는 다음의 도구 바 버튼을 선택해서 장소를 지정할 필요가 있습니다. 왼쪽코너 스타일 (선택, 병렬)을 맞춘 순번의 우측에 같은 스타일의 component 를 배치합니다. 왼쪽코너에서 component 가 선택되어 있지 않을 때는 우측에 component 를 배치 할 수 없습니다

F8: 매크로스텝 삽입

이 도구바 아이콘버튼에 따라 메인 차트에 매크로스텝을 삽입할 수 있습니다. 매크로스텝 은 메인 차트와 같은 S F C 차트 내에 기술해야 합니다.

F9: *F9: 매크로스텝 내용

매크로스텝은 Father 차트와 동일 S F C 차트 내에 적힐 필요가 있습니다. 매크로스텝은 **개시시스템**에서 시작되고 **종료시스템**에서 끝마칩니다. 매크로 개시시스템과 Father 매크로스텝은 **같은 참조번호**이어야 합니다. 매크로스텝을 적기위한 도구 바 버튼을 아래에서 나타냅니다.

A.4.3 SFC chart 의 작업

마우스 또는 커서키에 따라 차트 내의 셀을 선택합니다. 선택된 장소는 회색이 됩니다. 선택된 장소에 대해서는 「편집」 메뉴 명령어(자르기, 붙여넣기)를 사용할 수 있습니다

자르기/복사/삭제/붙여넣기

도구 바에서 화살표 버튼이 선택되어 있을 때 다음의 편집명령어가 선택된 심벌에서 사용할 수 있습니다.

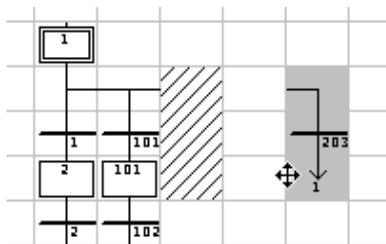
- **자르기**.....선택된 장방향 영역 클립보드로의 이동
- **인쇄**.....선택된 장방향 영역 클립보드로의 인쇄
- **붙이기**.....선택된 장방향 영역에서 클립보드 내용을 붙인다.
- **삭제**.....선택된 장방향 영역 삭제

붙이기 명령어는 클립보드 내용을 화면에서 인쇄하지만 이때 스텝 transition 과 함께 레벨 2 프로그램 내용도 인쇄됩니다.



이동

SFC chart 에서 선택되어 있는 심벌 / Element 는 마우스의 드래그 조작으로 다른 장소로 이동시킬 수 있습니다. 드래그 중에 최초의 장소는 사선표시 됩니다.



심벌의 이동 전은 공백이어야 합니다. SFC 심벌을 이동 중에 삽입은 할 수 없습니다

스텝과 transition 의 번호 매김

각 스텝, transition 에는 참조번호가 붙어 있습니다. 이 번호는 「편집」 - 「다시번호적기」에 따라 편집중의 SFC차트의 Reference 번호를 새로 자동으로 고칠 수 있습니다. 이때 스텝으로의 점프 참조번호와 매크로스텝으로의 점프 등의 번호도 자동으로 갱신됩니다.



스텝 또는 transition 에 직접 접근

S F C 차트 편집중에 「F5: Jump To Step」 명령어에 따라 편집중의 S F C 내에 지정된 스텝과 transition 으로 점프할 수 있습니다. 이때 점프 전의 스텝과 transition 이 보이도록 창 스크롤이 자동으로 행해집니다.



텍스트 검색과 치환

「편집」 - 「검색 / 치환」 명령어에 따라 편집중의 S F C 차트 내의 스텝, transition 내에 기술되어 있는 모든 프로그램을 대상으로 텍스트 검색과 변환을 할 수 있습니다. 검색 / 치환 DialogBox 에서 검색할 텍스트를 입력합니다. 만약 검색텍스트가 발견된 경우는 레벨 2 프로그램도 동시에 열립니다

A.4.4 레벨 2 프로그램 입력

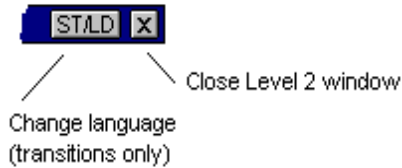
레벨 2 에서 프로그램을 입력할 때에는 SFC 스텝, transition 을 더블클릭 합니다. 레벨 2 프로그램창 창이 SFC 의 오른쪽 에서 열립니다. SFC 프로그램과 레벨 2 프로그램창 사이에는 세프레이션이 있고 자유롭게 비켜 놓을 수 있습니다.

레벨 2 프로그램을 최대 두개까지 동시에 열 수가 있습니다. 다음 키보드 입력명령어와 마우스 조작에 따릅니다.

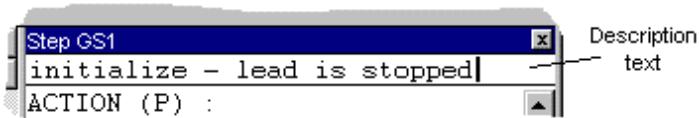
	키보드	마우스	[편집] 메뉴 명령어
제 1 창	Enter	더블클릭	레벨 2 의 편집
제 2 창	Ctrl+Enter	Ctrl + 더블클릭	분리창에서 레벨 2 편집

두개의 레벨 창이 열려 있을 때는 이들간의 구분 위치는 조정할 수 있습니다. 레벨 2 의 타이틀 바 오른쪽 위의 버튼으로 레벨 2 창을 닫을 수 있습니다.

레벨 2 프로그램에는 기본으로 S T 언어가 사용됩니다. 단 transition 레벨 2 프로그램에 관해서는 L D 언어 (QuickLD 편집기를 사용) 사용할 수 있습니다. 레벨 2 창 의 타이틀 바에 있는 「ST/LD」 바꾸기 버튼에 따라 액티브한 언어를 바꿀 수 있습니다. 이 바꾸기는 레벨 2 창이 공백 상태일 때만 유효하게 됩니다.



레벨 2 창의 상부에는 일행의 편집 가능한 텍스트 행이 있습니다. 이 텍스트가 스텝과 transition 의 명령어가 됩니다. 이 명령어는 「점프」 명령어와 SFC 문서 인쇄시의 명령어로서 사용됩니다.



「옵션」 - 「갱신」 메뉴 명령어에 따라 레벨 2 창이 열려 있을 시에 레벨 2에서 수정내용을 메인 SFC 차트에 반영 시킬 수 있습니다..



변수명 입력

텍스트 편집기로 프로그래밍을 행하고 있을 경우에 이 버튼을 선택하면 Project 에 선언되어 있는 변수목록이 열려 삽입하고 싶은 변수명을 선택할 수 있습니다. QuickLD 에서 프로그램을 행하고 있을 경우는 선택되어 있는 심벌에 대해서 변수명, Function BLOCKS 명, 파라미터 를 선택할 수 있습니다.



스텝에 Pulse (P) 액션 삽입

스텝의 레벨 2 프로그램을 행하고 있을 경우에 이 버튼을 선택하는 것으로 pulse 액션블록을 삽입할 수 있습니다. 다음에서는 형식을 나타냅니다.

```

Action (P) :
    ST statement;
    ...
End_Action;

```

pulse 액션은 스텝이 활성화상태가 되었을 때만 실행됩니다. 상세한 것은 언어 Reference



스텝에 normal(N) 삽입

스텝의 레벨 2 프로그램을 행하고 있을 경우에 이 버튼을 선택하는 것으로 normal 액션블록을 삽입할 수 있습니다. 다음에서 형식을 나타냅니다.

```

Action (N) :
    ST statement;
    ...
End_Action;
    
```

normal 액션은 스텝이 활성화 상태일 때 매번 사이클이 실행됩니다. 상세한 것은 언어 Reference 를 참조 바랍니다

P0 P1

P0 , P1

ISaGRAF 는 새로운 **P0** , **P1** 를 지지하고 있습니다. 스텝의 레벨 2 프로그램을 행하고 있을 경우에 이 버튼을 선택하는 것으로 **P0** , **P1** 액션블록을 삽입할 수 있습니다. 다음에서는 이들 형식을 나타냅니다.

<pre> Action (P0) : ST statement; ... End_Action; </pre>	<pre> Action (P1) : ST statement; ... End_Action; </pre>
--	--

P1 액션은 스텝이 활성화상태가 되었던 때만 (pulse 와 같은 모양) P0 액션은 스텝이 비활성화 상태이었을 때만 실행됩니다. 상세한 것은 언어 Reference 를 참조 바랍니다



Boolean 액션

Boolean 액션이란 현재의 **스텝상태**을 내부.외부변수에 할당하는 것입니다. 다음에서는 기본적인 Boolean 액션문법을 나타냅니다.(< > 는 입력하지 않습니다)

< TRUE 형 변수명 > (N) ;스텝의 Activity 상태를 변수에 할당
< TRUE 형 변수명 > ;(N 속성은 옵션)
< TRUE 형 변수명 > ;스텝의 Activity 상태의 반전을 변수에 할당

이들 이외에도 스텝상태가 액티브 되었을 때 변수치를 set 또는 reset 할 경우도 있습니다. 다음에서는 이들 문법을 나타냅니다.

< TRUE 형 변수명 > (S) ;스텝의 상태가 TRUE 가 되었을 때 변수치를 TRUE 로 한다.
< TRUE 형 변수명 > (R) ;스텝의 상태가 TRUE 가 되었을 때 변수치를 FALSE 로 한다.

☐ SFC 액션

S F C 액션에서는 child SFC Sequence 를 콘트롤 합니다. 즉 스텝의 상태에 대응해서 작동, 정지, Kill 을 합니다. S F C 액션에는 **N** (Non store) **S** (Set) **R** (Reset) 식별자가 붙습니다. 다음에서는 문법을 나타냅니다.:

<child_program> (N);	스텝이 액티브 된 때는 child Sequence 를 작동해 비액티브된 때에 child Sequence 를 정지시킵니다
<child_program>;	상동 (N 식별자는 옵션)
<child_program> (S);	스텝이 액티브된 때에 child sequence 를 작동해 Inactive 된 때에는 child Sequence 에 대해서 아무것도 하지 않습니다.
<child_program> (R);	스텝이 액티브된 때에 child Sequence 를 정지 (kills) 하고 Inactive 된 때에는 아무것도 하지 않습니다.

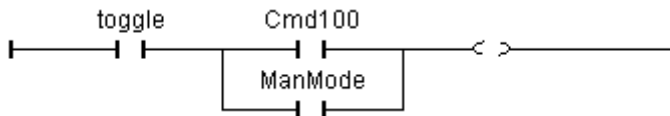
S F C 액션에서 지정되어 있는 프로그램은 현재 편집중의 **child S F C 프로그램**일 필요가 있다.

☐ ST 로 작성된 Transition

transition 의 레벨 2 프로그램은 TRUE 표현을 사용합니다. 이것은 TRUE 변수를 S T 문법에 따라서 입력하는 것입니다. 마지막에 ';' 를 부칩니다..

☐ QuickLD 로 작성된 Transition

transition 의 레벨 2 프로그램에는 QuickLD 를 사용할 수 있습니다. 이 경우는 1 행의 rung 에서만 표현해야 합니다. transition 상태를 나타내는 코일에는 변수명을 나눠서 부칠 필요는 없습니다. 다음에서 그 예를 나타냅니다.



QuickLD 편집기로 프로그램할 경우 키보드의 입력만으로 할 수 있습니다. 커서키로 커서를 이동시켜 심벌을 삽입할 때는 다음의 Function 키를 선택합니다.

F2:접점을 뒤에 삽입

F3: 접점을 앞에 삽입
 F4: 접점을 병렬로 삽입
 F6:Function BLOCKS 을 뒤에 삽입
 F7:Function BLOCKS 을 앞에 삽입
 F8:Function BLOCKS 을 병렬로 삽입
 (마우스를 사용해서 화면 위를 클릭하는 것도 가능합니다.)

접점과 Function BLOCKS, I/O 파라미터를 커서로 선택해서 ENTER 키를 선택하면, 변수를 선택할 DialogBox 가 열립니다. 목적 변수명, Function BLOCKS 명에 커서를 맞춰서 ENTER 키를 선택합니다. 더블클릭에 의해서도 선택할 수 있습니다.

커서키를 접점과 코일에 맞춰서 Ctrl 키+스페이스키를 선택하는 것으로 접점, 코일타입 (a 접점, b 접점) 을 변경할 수 있습니다. 상세한 것은 「QuickLD 의 사용법」을 참조 바랍니다

A.4.5 SFC 갤러리 사용법

SFC 편집기에서는 SFC 갤러리를 취급할 수 있습니다. SFC 갤러리라는 것은 각종 SFC 구조를 모은 것으로 SFC 차트 안에 삽입할 수 있습니다. SFC 갤러리에는 레벨 2 프로그램 내용을 포함시킬 수 있습니다. 다음에서 「도구」 메뉴명령어를 나타냅니다.

SFC 갤러리에 복사 선택된 element 를 SFC 갤러리에 복사

SFC 갤러리에 붙여넣기 SFC 갤러리 내의 element 를 SFC 차트 위에 붙여넣기

SFC 갤러리에 element 을 인쇄할 때 (즉 SFC 갤러리 element 의 신규작성 때) 레벨 2 프로그램을 포함시킬지 않을지 선택을 할 수 있습니다.

A.5 Flow Chart 편집기 사용법

ISaGRAF FlowChart 편집기는 액션, 테스트블록을 ST IL QuickLD 언어로 프로그램할 수 있습니다. FlowChart 는 판단을 기술하는 다이어그램이지만 sequence 조작도 기술할 수 있습니다. LD 와 IL 에서는 무한한 loop 이 될 경우에도 FlowChart 편집기에서는 Target 으로의 처리를 블록하지 않고 되돌리기 점프기술도 가능하게 됩니다

A.5.1 FC 언어 기본

Flow Chart (FC) 는 Sequence 한 처리를 기술하는 그래픽 언어 입니다. Flow Chart 프로그램은 **액션**과 **테스트**로 성립되어 있습니다. 액션과 텍스트를 데이터 흐름을 나타내는 **링**으로 결합합니다. 다음은 Flow Chart 언어의 기본요소를 나타냅니다.



FC 차트의 개시:

"begin" 심벌은 Flow Chart 프로그램 최초로 반드시 존재합니다. 이 심벌은 차트 내에 하나만 존재하고 삭제할 수 없습니다. 이 심벌은 Flow Chart 가 실행될 때 최초의 상태를 유지합니다.



FC 차트의 종료:

"end" 심벌은 Flow Chart 프로그램의 마지막에 반드시 존재합니다. 이 심벌은 차트 내에 하나만 존재하고 삭제할 수 없습니다.(End)심벌에 아무것도 접속되어 있지 않는 것과 같은 상태(loop 상태)가 가능합니다. 이 심벌이 실행된 때는 프로차트 마지막 상태로 됩니다.



FC flow 링(링):

flow 링은 차트 내에 두개의 포인트를 접속하는 라인입니다. 링에서는 흐름의 방향성을 가리키는 화살표가 붙어 있습니다. 두개의 링은 동일한 소스가 되는 포인트에서 그어지는 일은 없습니다.



FC 액션:

액션심벌은 실행될 액션을 기술합니다. 액션심벌은 번호와 이름에 의해 식별됩니다. 차트 내에 두개의 액션심벌은 동일번호, 또는 이름을 가질 수 없습니다. 액션은 ST,

LD, IL 의 어떤 언어로도 프로그램 됩니다. 액션은 링에 접속되고 하나의 링이 액션에 들어와 하나의 링이 액션에서 나갑니다.



FC 테스트:

테스트심벌은 TRUE 형의 조건식을 나타냅니다. 테스트심벌은 번호와 이름에 따라 식별됩니다. ST, LD, IL 언어로 기술된 표현의 평가결과에 따라 [YES], [NO] 중에서 buzz 로 프로가 이끌립니다. ST 언어로 프로그램 되어 있을 때는 표현 마지막에 세미콜론(;)찍는 일도 있습니다. LD 언어로 프로그램 되어 있을 때는 변수명이 정해져 있지 않는 하나의 코일이 평가결과를 유지합니다.



FC 서브프로그램:

Flow Chart 프로그램에서는 계층구조 프로그램을 할 수 있습니다. 하나의 Flow Chart 는 다른 Flow Chart 를 읽어낼 수 있습니다. 불러낼 프로그램은 FC 서브프로그램 [child FC 프로그램] 이라 불립니다. FC 서브프로그램을 읽어 낼 프로그램은 FatherFC 프로그램 이라고도 불립니다. Flow Chart 에 있어서는 FC 서브프로그램의 처리가 완료될 때까지는 FatherFC 프로그램 처리는 서스펜드 되어 있습니다. SFC 프로그램에서는 동시실행이 가능합니다.



FC I/O 조작블록:

I/O 조작블록심벌은 액션을 실행하기 위한 블록입니다. 기타 액션과 같은 모양으로 I/O 조작블록은 번호와 이름으로 식별됩니다. 다른 액션과 같은 의미를 가지지만, I/O 조작블록에서는 Target 의 하드창에 의존하고 있는 I/O 조작 등을 구별해서 표현하는 것에 따라, 프로그램을 알기 쉽도록 하기위해 만들어진 것입니다. 작동하는 것은 다른 액션과 같습니다.



FC 연결

연결은차트상에서의 두개의 포인트를 직접적으로 선을 긋는 것이 아니고 링 시키기 위해서 사용합니다. 연결은원으로 표현되어 접속소스에서 접속됩니다. 접속 전에 Flow Chart 상의 포인트를 (통상은 접속전의 이름과 번호)지정하는 것에 따라 연결설정이 끝납니다.



FC 명령어:

명령어는 Flow Chart 의 실행과는 관계 없지만 Flow Chart 상의 임의의 장소에 삽입할 수 있습니다. 명령어에 따라 프로그램을 읽기 쉽게 토큐먼트 할 수 있습니다.

A.5.2 Flow Chart 들어가기

Flow Chart 를 입력하기 위해서는 다음 구성요소 (액션, 테스트, 연결.. 등) 를 그래픽 편집기에 배치시켜 링으로 접속 할 수 있게 됩니다..



객체 삽입

차트에 오브젝트를 삽입하기 위해서는 도구 바에서 목적버튼을 선택해서 그래픽 편집기를 마우스클릭 합니다. 공백에 배치하는 것도 flow 링을 선택하는 것으로 링 사이에 삽입시키는 것도 가능합니다. 또한, 링 사이의 삽입은 수직 flow 링 위에서만 가능합니다.

다음의 기본요소를 삽입할 수 있습니다.

	 액션(ST,IL,QuickLD 중의 프로그램)
	IO 조작블록
	테스트(ST,IL,QuickLD 중의 프로그램)
	연결
	FC 서브프로그램 읽어내기
	명령어

ISaGRAF Flow Chart 에서는 다음의 전형적인 블록을 준비하고 있습니다. 이들은 이미 있는 flow 링에 삽입할 수 있습니다. 공백에서의 배치는 할 수 없습니다.

	If / Then / Else
	Repeat until
	While



객체 선택

그래픽오브젝트의 선택은 프로그램의 편집 작성에서 필요하게 됩니다. 차트 내의 하나 또는 복수 오브젝트를 선택하기 위해서는 도구 바상의 화살표 버튼  을 선택해서 차트 내의 오브젝트를 마우스클릭 합니다. 복수 오브젝트선택의 경우는 마우스를 드래그해서 선택하고 싶은 영역을 지정합니다. 선택된 오브젝트는 청색으로, 작은 사각으로 둘러싸인 것으로 표시됩니다. **Shift** 와 **Ctrl** 키 눌러면서 마우스 클릭하는 것에 따라 특정오브젝트를 포함시키기도 하고, 배제할 수도 있습니다.

새로운 오브젝트를 선택하기 위해서는, 전에 선택되어 있었던 오브젝트는 선택 에서 떼어 냅니다. 모든 선택을 떼어 내고 싶은 경우는 차트 내의 공백 영역을 마우스클릭 합니다.

단일 오브젝트선택의 경우는 키보드에서 화살표키에 따라 선택되어 있는 오브젝트를 바꿀 수 있습니다. flow 링을 선택할 수 있습니다.



주석 삽입

차트 내의 공백에리어에 명령어텍스트를 삽입할 수 있습니다. 명령어는 Flow Chart 처리에는 영향을 주지않습니다. 프로그램을 알기 쉽게 하기 위해 사용됩니다. 명령어를 삽입하기 위해서는 도구 바에서 명령어버튼을 선택해서 명령어를 배치하고 싶은 장소에 마우스클릭 합니다. 명령어 텍스트를 적어 넣을 때는 명령어 오브젝트를 더블클릭 합니다. "(" 와 ")" 등의 특정한 기호는 삽입할 필요는 없습니다. 선택된 명령어 블록은 코너부분을 드래그하면 자유롭게 사이즈 변경이 가능합니다



flow 링크 그리기

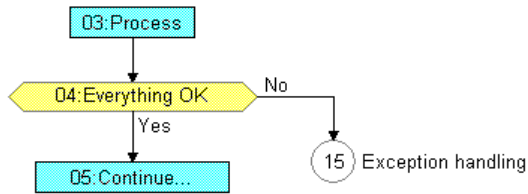
오브젝트 사이를 flow 에서 접속하기 위해서는 도구 바 위의 flow 삽입 버튼을 선택합니다. flow 의 방향성에 따라 마우스를 드래그해서 오브젝트 사이를 flow 로 접속합니다. 우선 접속되어 있지않는 오브젝트를 선택해서 목적버튼까지 마우스를 드래그합니다. 목적버튼은 미접속상태 오브젝트의 상부, 또는 존재하고 있는 링위의 장소가 됩니다. flow 위에서의 결합부분는 작은 그레이 원으로 표현됩니다. 이 결합버튼은 다이어그램을 정렬하기위해 이동시킬 수 있습니다.



연결

ISaGRAF Flow Chart 편집기에서는연결을 다룰 수 있습니다. 이것은 flow 링과 같이 두개의 오브젝트사이의 접속을 표현하는 것입니다. 연결이용하는 것으로 긴 flow 링을 사용하지 않고 끝내기 위해 차트가 알기 쉽게 되는 장점이 있습니다. 연결로 다른 Flow Chart 와의 접속은 할 수 없습니다.

연결은 Flow Chart 상에 하나의 오브젝트로 존재합니다. 도구 바상에서 연결버튼을 선택해서, 차트 위에 배치할 장소를 마우스클릭 합니다. 이때 접속 전에 리스트 Box 가 표시되며, 여기서 선택합니다. 연결은원으로 표현되어 접속 전의 오브젝트 번호와 이름을 유지하게 됩니다. flow 링에 따라 연결이 다른 오브젝트에서 접속됩니다.



이동

차트내의 오브젝트를 이동하기 위해서는 이동시킬 오브젝트를 선택버튼으로 미리 선택해 놓습니다. 하나 또는 복수오브젝트를 선택해 놓을 수 있습니다. 이들을 마우스로 드래그하면 이동시킬 수 있습니다. 오브젝트는 기타 오브젝트와는 서로 중복할 수 없습니다.

하나의 오브젝트 (액션, 테스트 등) 가 선택되어, 이동시키면 나머지 차트도 자동적으로 다음에 접속되어 있는 오브젝트도 이동합니다



크기조절

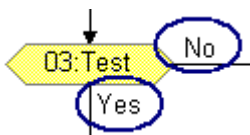
"Begin"과 "End"심벌 이외의 모든 그래픽오브젝트는 자유로이 사이즈변경이 가능합니다. 오브젝트 사이즈를 변경하기 위해서는 오브젝트를 선택해서, 오브젝트 경계선의 작은 사각부분을 마우스로 드래그합니다. 단 이때 오브젝트가 다른 오브젝트와 중복될 수 없기 때문에 (또는 창 경계와의 간섭) 중복될 것 같은 오브젝트를 미리 이동시켜 놓을 필요가 있을지도 모릅니다.

수직인 flow 링이 오브젝트와 접속되어 있을 경우는 오브젝트는 수평으로 변경하면 자동적으로 좌우로 나누어져 flow 링이 오브젝트의 중심이 됩니다.



출력 바꿔 넣기

테스트블록에서 출력된 YES/NO 의 위치를 바꿔 넣을 수 있습니다. 바꿔 넣기 위해서는 YES, NO 의 라인 주의 하나를 더블클릭 합니다.



A.5.3 chart 에서 작업 하기

Flow Chart 의 「**편집**」 메뉴 명령어에 따라 선택되어 있는 오브젝트에 대해서 편집이 가능합니다.

chart 수정

DEL 키는 선택한 오브젝트 삭제에 사용할 수 있습니다. 선택된 오브젝트 에서 접속되어 있는 flow 링도 동시에 삭제됩니다. 「**편집**」 - 「**되돌리기**」 메뉴 명령어에 따라 삭제된 오브젝트를 복원할 수 있습니다. DEL 키는 복수 선택된 오브젝트에 대해서도 조작이 가능합니다. 「**잘라내기**」, 「**인쇄**」, 「**붙이기**」 명령어에 따라 선택된 오브젝트를 이동, 인쇄할 수 있습니다.

찾기 / 바꾸기

「**편집**」 - 「**검색**」, 「**치환**」 메뉴 명령어에 따라 프로그램 전체 (액션, 테스트블록이 ST,IL,QuickLD 로 기술되어 있다) 에 대해서 텍스트의 검색, 치환이 가능합니다. 검색, 치환은 열려 있는 편집기창에서 텍스트가 검색되어 텍스트가 검색되면 레벨 2 프로그램이 열립니다.

요소에 직접 접근

「**편집**」 - 「**점프**」 메뉴 명령어에 따라 또는 도구 바상의  버튼 선택에 따라 열려 있는 Flow Chart 내에 지정된 그래픽오브젝트로 직접접근을 할 수 있습니다. 창 의 스크롤은 자동적으로 행해지며 오브젝트가 보이는 곳에 표시됩니다

번호 매기기

「**편집**」 - 「**번호 고치기**」 메뉴 명령어에 따라 Flow Chart 내에서 오브젝트에 할당되어 있는 번호 고치기를 합니다. Flow Chart 상에서의 오브젝트는 unique 한 참조번호를 취할 수 있습니다. 새로운 오브젝트가 삽입될 때마다 새로운 번호가 할당됩니다. 「**번호 고치기**」에 따라 차트상의 장소에 따라, 오브젝트번호가 고쳐집니다. 번호는 위에서 아래로, 좌에서 우로 번호가 증가합니다.

A.5.4 레벨 2 프로그램

액션과 테스트블록에 레벨 2 프로그램을 기술할 때는 액션과 테스트블록을 선택해서 더블클릭 합니다. 레벨 2 프로그램 창이 Flow Chart 창의 우측에 열립니다. Flow Chart 창과 레벨 2 프로그램 창 사이의 경계는 마우스로 자유롭게 이동할 수 있습니다. 최대로 두개까지 레벨 2 창을 동시에 열수 있습니다. 다음에서 레벨 2 창을 열기 위해 키보드, 마우스, 편집 메뉴조작을 나타냅니다.

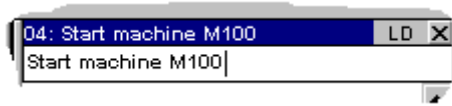
	키보드	마우스조작	편집-메뉴명령어
제 2 창	Enter	더블클릭	레벨 2 편집
제 2 창	Ctrl+Enter	Ctrl + 더블클릭	세퍼레이트 창에서 레벨 2 편집

두개의 레벨 2 창이 열려 있을 시에는 이들 경계는 마우스로 자유롭게 이동할 수 있습니다. 레벨 2 창의 오른 위 버튼으로 창을 닫을 수 있습니다.

레벨 2 프로그램언어의 Default 는 ST 언어입니다. 또한 프로그램언어는 IL, QuickLD 언어를 이용하는 것도 가능합니다. 선택되어 있는 프로그램 언어는 창의 타이틀바 우측에 표시됩니다. 프로그램 언어를 변경할 경우에는 이 부분을 마우스클릭해서 변경합니다. 「옵션」 - 「레벨 2 언어설정」 메뉴 명령어에 따라서도 언어를 바꿀 수 있습니다. 단, 프로그램언어 바꾸기는 레벨 2 프로그램이 공백 상태일 때만 유효합니다. 한번 프로그램을 행했던 경우는 이것을 완전하게 삭제하면 다시 선택이 가능하게 됩니다.



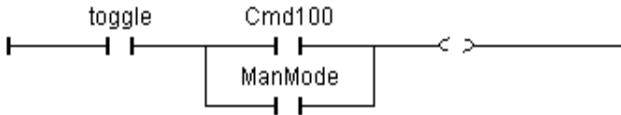
레벨 2 창 타이틀바 하부에 1행의 텍스트편집 Box 가 열립니다. 여기에 입력된 텍스트는 테스트와 액션블록 명령어가 되고, 「점프」 명령어 등으로 점프 앞을 선택할 경우에 유효하게 사용할 수 있습니다. 또한 Flow Chart 의 인쇄 시에도 표시됩니다.



「옵션」 - 「최신정보」 메뉴명령어에 따라 레벨 2 프로그램 내용을 메인 Flow Chart 프로그램에 반영시킬 수 있습니다.

A.5.5 프로그램 & QuickLD

레벨 2 프로그램에 QuickLD 편집기를 사용할 수 있습니다. 테스트블록에서의 LD 프로그램에는 괄줄의 rung 만 프로그램 됩니다. 코일에 변수할당은 필요 없습니다. 다음에서 그 예를 나타냅니다.



QuickLD 에 의한 프로그램으로 Ladder 프로그램용 심벌을 삽입하기 위한 쇼트컷 키를 다음에서 나타냅니다.

- F2: 접점을 뒤에 삽입_^rung 개시에 사용
- F3: 접점을 앞에 삽입
- F4: 접점을 병렬에 삽입
- F5: 코일을 병렬에 삽입(테스트블록에서는 사용불가)
- F6: 블록을 뒤에 삽입
- F7: 블록을 앞에 삽입
- F8: 블록을 병렬에 삽입
- F9: 선택된 코일과 병렬에서 점프삽입 (테스트블록에서는 사용불가)

점프하기 전은 rung 명이 됩니다. rung 명을 입력할때는 rung 의 맨앞(좌단)에 커서를 set 해서 ENTER 키를 선택합니다.이전에 입력마침 라벨명이 표시되고, 신규 rung 명을 입력 할 수 있습니다. 또, 점프 전에 입력할 [라벨점프] DialogBox 에서는 점프 전의 라벨명을 선택합니다. [삭제] 버튼에 따라 점프 전의 라벨명을 삭제할 수 없습니다. 단, 이 조작에 따라 라벨명은 삭제할 수 없습니다.

접점, 코일, 블록의 입출력 파라미터에 커서를 선택한 상태에서 ENTER 키를 선택해서 변수를 할당할 수 있습니다. 블록에 커서를 놓으면 Function

BLOCKS 타입을 선택할 수 있습니다. 더블클릭에 의해서도 같은 형식으로 가능합니다.

Ctrl + 스페이스 키에 따라 선택되어 있는 접점의 타입 (A 접점, B 접점) 을 변경할 수 있습니다. 상세한 것은 QuickLD 편집기의 사용법을 참조 바랍니다.

A.5.6 보기 옵션

옵션 - 윤곽 메뉴명령어에 따라 차트의 화면표시에 관계하는 파라미터가 정리된 DialogBox 가 열립니다.

[워크스페이스] 그룹 Box 에서는 도구 바와 상태바의 표시/비표시를 할 수 있습니다.

도큐먼트 그룹 Box 에서는 편집 grid 의 표시/비표시 와 칼라모드의 선택을 할 수 있습니다.



도구 바상의 **Zoom** 버튼에 의해 Zoom 비율을 변경할 수 있습니다. 이 버튼은 텍스트와 액션을 기술 하는 레벨 2 프로그램에서도 유효합니다. (선택되어 있는 창이 Zoom 의 대상이 됩니다.)



도구 바상의 [grid]버튼에 의해 편집 grid 의 표시/비표시를 할 수 있습니다. 이 버튼은 테스트와 액션을 기술하는 레벨 2 프로그램 에서도 유효합니다.(선택되어 있는 창이 Zoom 의 대상이 됩니다)

「옵션」 - 「글꼴」 메뉴명령어에 따라 차트내의 텍스트폰트타입을 선택할 수 있습니다. ST,IL 프로그램에서는 폰트사이즈도 유효하며 설정할 수 있습니다. FC, QuickLD 에서 폰트사이즈는 화면사이즈에 대응해서 자동적으로 변경되기 때문에 사이즈의 설정은 무효가 됩니다..

A.6 QuickLD 편집기 사용법

이 장에서는 QuickLD 편집기 에 관한 명령어특징을 설명합니다. 다른 편집기와의 공통점에 관해서는 [프로그램 편집기 공통항목에 관해서] 를 참조 바랍니다.

LD 언어는 TRUE 표현의 그래픽 표현입니다. TRUE 연산자 AND, OR, NOT 이 다이어그램으로 표현됩니다. TRUE 입력은 접점에 할당되고, TRUE 출력은 코일에 배정됩니다. QuickLD 편집기는 키보드와 마우스를 사용해서 간단하게 LD 다이어그램을 입력할 수 있도록 만들어진 편집기입니다. 이 편집기에서 각 심벌은 자동적으로 정렬되어 접속 됩니다. 사용자에게 의한 수동에서의 라인접속은 필요 없습니다. 또한, 심벌간의 불필요한 공백은 없습니다.

A.6.1 LD 언어 기본 사항

LD 다이어그램은 접점과 코일에서 표현되는 rung 리스트입니다. 다음에서는 LD 다이어그램에서의 기본컴퍼넌트를 나타냅니다:



좌모선 (left power rail)

각 rung 는 좌모선에서 시작됩니다. 이것은 항상 TRUE 상태입니다. QuickLD 편집기에서는 최초접점이 배치된 때에 자동으로 좌모선이 그어집니다. 각 rung 는 이름을 가질 수 있습니다. 이 이름은 점프인스트럭션 전에도 사용됩니다



접점

접점에 할당된 TRUE 형 변수상태(TRUE, FALSE)에 따라 TRUE 데이터의 흐름을 바꿉니다. 즉, 접점의 좌측상태를 우측에 어느 정도로 전달할 것인지 결정됩니다. 변수명은 접점심벌 위에 표시 됩니다. 다음의 접점타입이 QuickLD 편집기로 서포트되고 있습니다.



.....a 접점



.....b 접점



.....오르기 접점



.....내리기 접점



코일

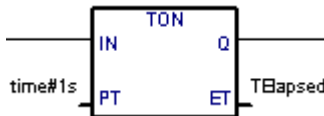
코일은 rung 상태를 받아들여 액션을 표현합니다. 즉 코일 좌측의 TRUE 상태가 코일에 배치되어 있는 TRUE 형 변수에 설정됩니다. 코일의 변수명은 코일상에 표시됩니다. 아래 타입의 코일이 QuickLD 편집기로 스포트되고 있습니다.

- { }코일
- {N}반전코일
- {S}set 코일
- {R}reset 코일
- {P}오르기 검출코일
- {N}내리기 검출코일



Function Blocks

LD 중의 블록은 Function, Function BLOCKS, 서브프로그램, 또는 operator (연산자) 중에 어떤 것이 됩니다. 최초입력과 출력은 항상 rung 에 접속되어 있습니다. 이것 이외의 입출력 파라미터는 블록 밖에 표시됩니다.



우모선 (right power rail)

rung 은 우모선 으로 끝마칩니다. QuickLD 편집기에서는 우모선은 코일이 배치된 때에 자동적으로 그어집니다



점프 심벌

점프심벌은 rung 라벨명으로 점프할 수 있습니다. 점프전의 rung 라벨은 동일 LD 다이어그램이어야만 합니다. 점프심벌은 rung 의 마지막에 배치됩니다. rung 상태가 TRUE 시에는 목적 rung 로 점프가 행해집니다. 역행점프는 처리가 loop 할 가능성이 있기 때문에 주의를 요합니다.



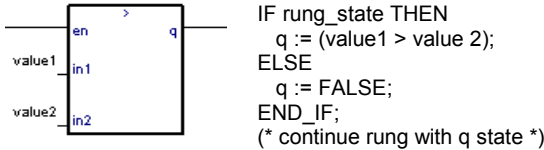
리턴 심벌

리턴심벌은 rung 의 마지막에 배치됩니다. 만약 이 rung 의 상태가 TRUE 일 때는 이 프로그램의 실행을 정지되고, 다음 프로그램으로 이동합니다



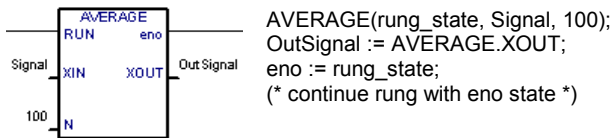
"EN" 입력

Function, Function BLOCKS, 연산자 중에서는 최초 입력파라미터가 TRUE 형이 아닌 것도 있습니다. 이 경우에도 최초의 파라미터는 반드시 랑그에 접속될 필요가 있기 때문에 이 같은 상황에서는 블록에 자동적으로 "EN"이라 불리는 새로운 입력파라미터가 생성됩니다. "EN"입력이 TRUE 상태에서만 블록이 실행됩니다. 아래에서는 "EN"입력을 포함한 블록의 예와 이것과 등가한 S T 프로그램 코드를 나타냅니다.:

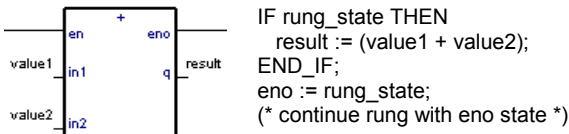


"ENO" 출력

Function, Function BLOCKS, 연산자 중에서는 최초에 출력파라미터형이 아닌 것이 있습니다. 이 경우에도 최초의 파라미터는 반드시 rung 에 접속할 필요가 있기 때문에 이 같은 상황에서는 블록에 자동적으로 "ENO"라 불리는 새로운 출력 파라미터가 생성됩니다. "ENO"출력은 항상 블록 최초의 입력 파라미터 상태와 같은 상태가 됩니다. 아래에서 "ENO"출력을 포함한 AVERAGE 블록의 예와 이것과 등가한 S T 프로그램코드를 나타냅니다



어떤 경우에는 EN 과 ENO 쌍방이 필요한 블록도 있습니다. 아래에서 산술연산 예를 나타내고, 등가한 S T 표현도 나타냅니다.



HH # QuickLD 편집기 제한사항

TOPIC110

QuickLD 편집기에서는 코일 우측에 심벌을 삽입할 수 없습니다. 만약, 복수 코일이 필요할 경우는 코일은 병렬에 배치됩니다. 하나의 rung 로 복수코일을 다른 조건으로 배치할 수 없습니다

A.6.2 LD diagramd 입력

QuickLD 편집기에서의 편집작성은 모든 키 보드에서만 할 수 있습니다.



눈금편집

LD 다이어그램의 각 심벌은 grid 에서 나뉜 셀 상에서 배치됩니다. 현재 커서위치를 이동하기 위해서는 커서 키에서 이동할지를 직접 마우스로 지정합니다. 선택중의 셀은 반전상태가 됩니다. 인쇄 / 붙이기 등의 조작을 위해 복수 셀을 선택할 경우는 마우스로 드래그할지를 SHIFT 키를 선택하면서 커서키를 움직입니다.



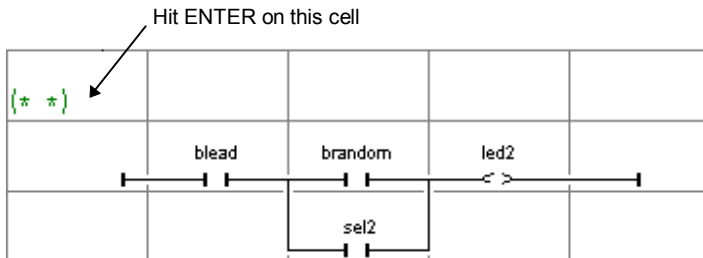
새로운 rung 시작

새로운 rung 을 작성할 경우에는 마지막 rung 다음 행에 선택 셀을 이동합니다. 거기에서 접점을 삽입하기 위해 **F 2** 키를 선택합니다. 접점 하나에 코일 하나의 새로운 rung 가 작성됩니다.



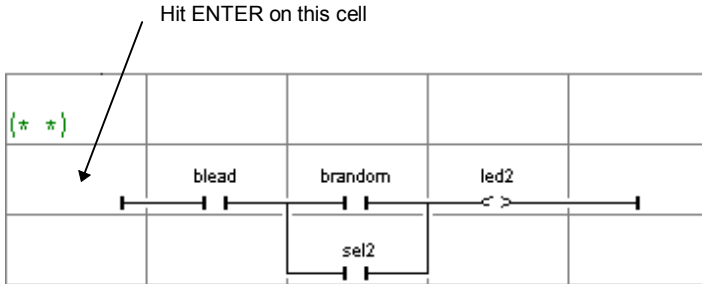
rung 주석삽입

각 rung 는 최대 2 행까지 텍스트명령어를 가질 수 있습니다. rung 명령어 입력을 위해서는 선택 셀을 rung 왼쪽 위의(* *) 로 해서 ENTER 키를 선택 합니다. 또는 더블클릭 합니다.



rung 라벨 입력

각 rung 은 이름을 가질 수 있습니다. 이 이름 (라벨) 은 점프 조작 때 점프 앞에 사용됩니다. rung 라벨을 입력할때는 선택 셀을 rung 의 좌단 으로 이동시켜 ENTER 키를 선택합니다. 또는 더블클릭 합니다.:



QuickLD 편집기는 입력종료 rung 라벨을 보존하고 있습니다. 아래의 DialogBox 로 점프전에 선택을 할 것일지를 새로운 rung 라벨 입력을 합니다.

새로운 rung 라벨을 입력하면 리스트에 추가됩니다. **[잘라내기]**버튼이 선택되면 선택 중의 라벨명은 삭제됩니다. 또한 선택중 rung 라벨명을 지우고 싶을 경우에는 간단히 DialogBox 입력필드 안을 비우고 **[확인]** 버튼을 선택합니다.

rung 에 심벌 삽입

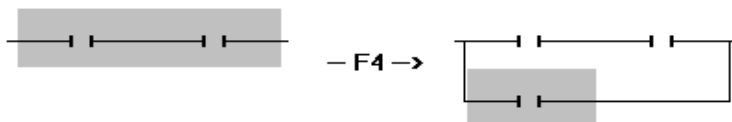
기존의 rung 에 심벌(접점, 코일, 블록)을 삽입할 경우는 우선 삽입대상 위치를 선택셀 (복수셀의 동시 선택도 가능) 에서 부터 아래의 Function 키 중에서 선택 합니다.

- F2좌측에 접점을 삽입
- F3우측에 접점을 삽입
- F4선택심벌과 병렬에 접점을 삽입
- F6좌측에 블록을 삽입
- F7우측에 블록을 삽입
- F8선택심벌과 병렬에 블록을 삽입

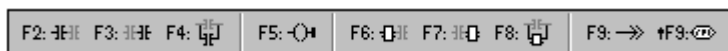
다음의 명령어는 선택셀이 코일 장소 에만 유효합니다.

- F5 병렬코일의 추가
- F9 병렬에 점프심벌 추가
- Shift + F9 병렬에 리더심벌 추가

F 4, F 8의 병렬삽입에 관해서는 복수접점이 모여 선택되어 있는 것과 같은 경우의 심벌삽입은 선택되어 있는 심벌에 대해 병렬로 행해집니다. 아래는 그 예를 나타냅니다.:



다이아그램심벌을 삽입할 경우는 「삽입」 메뉴명령어를 사용할 수 있습니다. 또한 마우스로 LD 도구 바위의 버튼을 클릭 할 수 있습니다.



심벌 입력

선택중의 접점코일, 블록에 변수명과 블록명을 할당하기 위해서는 ENTER 키를 선택합니다. 또는 더블클릭합니다. 변수선택 DialogBox 가 표시됩니다. 이 DialogBox 의 사용법에 관한 상세한 것은 [프로그램편집기 공통항목] 을 참조바랍니다. Function, Function BLOCKS, 연산자로의 할당의 경우는 Box 내부를 선택해서 ENTER 키를 선택합니다. 블록의 입출력 파라미터의 할당시에는 Box **바깥쪽**을 선택해서 ENTER 키를 선택합니다.

「옵션」 - 「키보드입력」 메뉴명령어가 선택되어 있을 시는 변수명과 Function BLOCKS 명의 입력을 위해 아래와 같은 1 행의 텍스트 Box 가 표시되어 지점 이름을 입력할 수 있습니다. 변수명과 Function 명을 입력한 후에 **ENTER 키**로 입력이 완료됩니다. **ESC 키**로 텍스트 Box 를 닫을 수 없습니다. 이 텍스트 Box 는 마우스 조작으로는 닫을 수 없습니다.



접점/코일 타입 변환

「편집」 - 「Coil/Contact 변경」에 따라 선택중의 접점과 코일타입을 변경할 수 있습니다. 이 조작은 스페이스 키, 또는 CTL + 스페이스 키를 선택함 으로서 할 수 있습니다.



rung 삽입

「**편집**」 - 「**rung 삽입**」 명령어에 따라 편집중의 rung 직전에 rung 을 삽입할 수 있습니다. 삽입된 rung 은 접점하나, 코일 하나의 기본 rung 입니다. 이것을 변경하게 됩니다.

A.6.3 작업

「**편집**」 메뉴로 다이아그램의 수정을 행하고, L D 다이아그램을 완성시킵니다. 많은 명령어는 선택셀에 대해서 액션이 됩니다.

≡ 수정

DEL 키가 선택된 심벌의 삭제에 사용됩니다. 단, 코일, 점프, 리턴심벌은 이것이 유일 출력일 경우는 이것을 삭제할 수 없습니다. 「**편집**」 - 「**재생**」 명령어에 따라 DEL 키 명령어 후에 삭제된 심벌을 되돌릴 수 있습니다. DEL 키를 rung 좌단에 사용하면 대상 rung 이 삭제되기 때문에 주의해 주십시오.

≡ 심벌 복사

편집 - 자르기,인쇄,붙이기 명령어에 따라 선택중의 심벌을 이동, 인쇄할 수 있습니다.
편집 - 그림복사 명령어에 따라 붙이기 할 때 선택할 수 있습니다.

- 좌측에 붙이기
- 우측에 붙이기
- 병렬에 붙이기

≡ 모든 rung 관리

선택셀이 rung 좌단의 경우는 모든 편집 명령어는 rung 전체에 대해서 행해집니다. 이것을 이용하는 것으로 다이아그램 내에서 rung 정렬이 커서를 좌단 옆에 이동하는 것만으로 쉽게 할 수 있습니다. 더욱이 복수 rung 을 정리해서 선택, 조작할 경우는 좌단의 옆에서 수직방향으로 복수 rung 를 선택합니다.

≡ 찾기/치환

편집 - 대체에 따라 다이아그램 중의 텍스트를 검색, 치환할 수 있습니다. 검색은 완전일치의 경우에만 스포트하고 있습니다. 검색은 접점명, 코일명, 블록명, 블록파라미터명, rung 라벨명에 대해서 행해집니다. rung 명령어 중의 텍스트

검색에서는 사용할 수 없습니다. 치환명령어에 따라 블록치환은 할 수 없습니다. 검색방향은 선택셀의 전후방에서 할 수 있습니다. 변수명 검색을 위해 아래에 short cut 준비되어 있습니다.

ALT + F2 선택변수명과 같은 변수명을 가까운 순으로 검색합니다. 이 검색은 Function BLOCKS 명과 rung 라벨명에 대해서도 할 수 있습니다.

ALT + F5 선택변수명의 코일을 가까운 순으로 검색합니다. 이 기능은 디버그 중에 수상한 접점이 발견된 경우의 코일 검색할 때에 유효합니다

A.6.4 옵션

「**옵션**」 메뉴에는 화면상에서의 LD 다이어그램의 표시방법 변경, 각종정보의 표시/비표시의 등의 명령어를 포함하고 있습니다

☐ **Rung 주석**

「**옵션**」 - 「**rung 주석**」 명령어에 따라 rung 명령어의 표시/비표시가 다이어그램 전체에 대해서 설정 가능합니다. rung 명령어를 비표시하는 것으로 보다 많은 행수의 rung 를 화면에 표시할 수 있습니다. 이 명령어에 따라 rung 명령어가 사라지는 일은 없습니다. 항상 표시/비표시의 바꾸기를 할 수 있습니다.

☐ **이름/별명**

접점 코일, 블록파라미터에 할당된 가 변수는 변수명 이외에 QuickLD 편집기에서는 새롭게 별명을 설정했습니다. 각 변수에 있어서 별명과는 변수명령어 텍스트의 ':' 으로 나뉜 부분의 텍스트, 또는 왼쪽에서 16 바이트 문자열이 됩니다. 아래에서 그 예를 나타냅니다.

<i>variable comment:</i>	<i>별명:</i>
short text	short text
long text with no 구분선	long text with n
short text: long de 시뮬레이션메모 ion	short text

한국어 주석도 사용할 수 있습니다. 한국어 명령어 별명은 LD 다이어그램에서는 주석과 같은 위치가 됩니다. 다이어그램에 표시할 변수명과

별명은 3 가지에 따른 표시방법이 있고, **옵션 - 코일 과 접점** 명령어에 따라 바꿀 수 있습니다.

- 이름(변수명)만
- 별명만
- 이름(변수명)과 별명

QuickLD 에서는 변수모음이 갱신된 때에 LD 프로그램의 변수별명명을 화면으로 자동갱신을 행하지 않습니다. 단, **옵션 - 코일 과 접점 - 별명갱신** 에 따라 LD 프로그램 중의 별명이 갱신됩니다.

또한 **「옵션」 - 「코일/접점」 - 「얼때마다갱신」** 명령어를 설정함에 따라 LD 프로그램이 열릴때에 변수별명의 자동갱신을 반드시 합니다. 단, 이 설정에 따라 프로그램이 열릴 때 요하는 시간이 길어질 수 있습니다.



그리기 옵션

「옵션」 - 「배치도」 명령어에 따라 편집기의 워크스페이스와 LD 표시방법에 관한 옵션파라미터를 선택할 수 있습니다.

워크스페이스 그룹 Box 내의 설정에는 도구 바, 상태바, LD 도구 바의 표시/비표시 선택이 있습니다. 도큐먼트 그룹 Box 내의 설정에는 grid, 칼라의 표시/비표시선택이 있습니다.



더욱이 Zoom 그룹 Box 내의 설정에서는 전체 Zoom 룰의 선택이 있습니다. Zoom 버튼은 도구 바 아이콘에도 포함되어 있습니다.



또한 X/Y 비율의 카스타마이즈에 따라 효율적으로 다이어그램내용을 창에 표시할 수 있습니다. X/Y 비율(셀폭)버튼은 도구 바 아이콘에도 포함되어 있습니다.

[폭 조정]버튼에 따라 자동적으로 상태의 QuickLD 편집기 창 사이즈에 맞도록 X/Y 비율을 설정할 수 있습니다.

「옵션」 - 「글꼴」 에 따라 폰트타입을 변경할 수 있지만, 폰트사이즈에 관해서는 그래픽 화면의 Zoom 설정룰에 따라 자동적으로 조정되기 때문에 여기서의 설정은 유효하지 않습니다

A.7 FBD/LD 편집기 사용법

ISaGRAF FBD/LD (그래픽)편집기는 LD 를 부분적으로 포함 하여서 사용자 에게 FBD 프로그램을 완벽하게 작성 할 수 있도록 도와 줍니다. 그래픽/텍스트 를 동시에 편집 가능하게 하여서 두 다이어그램과 해당 입/출력을 제어 가능 하게 합니다.이 편집기는 FBD 용으로 설계 되었으므로 완전한 LD 만을 사용할 경우 QuickLD 편집기를 이용 바랍니다.

A.7.1 FBD/LD 언어 기본사항

F B D 언어는 각종 타입의 장방형 (상자) 으로 나타낸 Function BLOCKS 이 접속된 그래픽표현 방법의 하나입니다. 상자 좌측에는 Function 의 입력이, 우측에는 Function 의 출력이 접속됩니다. 변수가 **논리적인 링**을 가진 상자에 접속됩니다. 출력은 다른 Function 의 입력에 접속되는 일도 있습니다.

LD 언어는 TURE 표현의 그래픽한 표현 방법입니다. TURE 연산의 **AND,OR,NOT** 이 그래픽에 표현됩니다. TURE 형 입력은 접점에서, TURE 형 출력은 **코일**에 할당됩니다.

LD 는 좌우 수직라인에 끼어 있습니다. 이것은 **좌 모션, 우 모션** 으로 불립니다.

접점과 코일은 좌우 모션에 **수평라인**으로 접속되어 있습니다. 각각 라인 상의 segment 는 **FALSE 또는 TRUE** 치를 가집니다. **좌 모션** 에 접속되어 있는 포인트는 모두 TRUE 가 됩니다.



좌모션

각 rung 은 **좌 모션**에 접속되어 있어야만 합니다.이것은 항상 TRUE 상태입니다. F B D편집기에서는 TURE 심벌을 접속할 수 있습니다



우모션

코일 우측은 **우모션**에 접속되어야만 합니다. 단, F B D편집기에서는 반드시 우 모션이 필요하다고 할 수는 없습니다.



LD 수직 "OR" 접점

L D 편집기로 수직접속 라인은 좌측 복수접속을 모아서 우측에 접속합니다. 이것은 좌측에 접속되어 있는 복수 라인의 논리화 (OR) 연산이 됩니다.



접점

접점에 나누어져 붙어있는 TURE 형 변수상태(TRUE, FALSE) 에 따라 TURE 데이터의 흐름을 바꿉니다. 즉, 접점의 좌측 상태를 우측에 어느 정도 전달할 것 인지가 정해집니다. 변수명은 접점심벌 상에 표시됩니다. 다음의 접점타입이 FBD/LD 편집기로 지원되고 있습니다.

- a 접점
- b 접점
- 올리기 접점
- 내리기 접점



코일

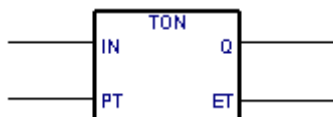
코일은 rung 상태를 받아들여 액션을 표현합니다. 즉, 코일 우측의 TURE 상태가 코일에 나누어져 붙어 있는 TURE 형 변수에 설정됩니다. 코일 의 변수명은 코일 심벌상에 표시됩니다. 다음의 타입 코일이 FBD/LD 편집기로 지원되고 있습니다.

- 코일
- 반전 코일
- set 코일
- reset 코일



Function Blocks

F B D 가운데 Function, Function BLOCKS, 서버프로그램, 또는 operator (연산자) 중에 어떤 것이 됩니다. 최초의 출력과 입력은 항상 rung 에 접속되어 있습니다. 이외의 입출력 파라미터는 블록 밖에 표시되어 있습니다.



라벨

라벨은 다이어그램 중의 어디에서도 배치할 수 있습니다. 라벨은 인스트럭션 실행순서를 변경하기 위해 점프명령의 점프 전에 사용됩니다. 라벨은 다른 심벌과는

접속되지 않습니다. 다이어그램을 읽기 쉽게 하기 위해 통상 라벨은 좌단에 배치됩니다



점프

점프명령은 다이어그램 중에 배치된 지정라벨에 실행을 옮깁니다. 이 심벌의 좌측은 TURE 변수 (상태) 에 접속될 필요가 있습니다. 왼쪽 접속라인이 TRUE 시에 점프가 실행됩니다. 역행점프는 처리가 loop 할 가능성이 있으므로 주의를 요합니다. 점프명령은 다이어그램 주에 배치된 지정라벨로 실행을 옮깁니다. 이 심벌의 좌측은 TURE 변수 (상태) 에 접속될 필요가 있습니다. 왼쪽 접속라인이 TRUE 시에 점프가 실행됩니다. 역행점프는 처리가 loop 할 가능성이 있으므로 주의를 요합니다.



반환 심벌

리더심벌은 rung 마지막에 배치됩니다. 만약 이 rung 상태가 TRUE 시에 이 프로그램의 실행을 정지하고, 다음 프로그램으로 옮겨집니다



변수

다이어그램 중의 변수는 이 심벌 중에서 사용됩니다. 좌우에서의 접속라인이 다른 심벌과 접속됩니다.



연결 링크

심벌간을 접속할 경우에 사용합니다. 접속라인 (링) 은 출력 버튼에서 입력버튼으로 접속됩니다.



논리반전 접속

논리접속을 할 때에 논리를 반전해서 논리를 반전해서 접속할 경우가 있습니다. 이 경우에 사용합니다.



사용자 정의 코너

접속라인을 그을때 사용자가 정의한 임의장소에 접속코너 포인트를 배치할 수 있습니다. 배치된 코너는 접속라인이 통하는 포인트가 됩니다. 이 코너를 배치함에 따라, 다이어그램의 접속라인을 정리해서 읽기 쉽도록 할 수 있습니다. 코너가 없는 경우는 F B D 편집기는 기본루트에서 접속라인이 그려집니다.

A.7.2 입력 FBD

다이아그램의 입력은 심벌 (블록, 변수, 접점, 코일 등) 을 배치해 이들을 접속라인으로 접속하게 됩니다.



심벌 배치

다이아그램에 심벌을 배치할 경우는 도구 바 아이콘에서 이 버튼을 선택해서, 심벌을 배치하고 싶은 장소에 클릭합니다.



심벌 선택

심벌을 선택하는 것은 편집작업에 대개의 경우 필요합니다. FBD/LD 편집기 에서는 하나 이상의 심벌 선택이 가능합니다. 심벌을 선택할 경우는 도구 바에서 **선택버튼** (화살표 버튼)을 체크해 놓습니다. 하나의 심벌을 선택할 경우는 그 심벌을 클릭합니다. 복수 심벌을 선택할 경우는 마우스로 장방형지역을 드래그하면서 필요로 하는 심벌을 모두 선택합니다.

선택된 지역내 (경계와 교차해서도확인) 의 심벌은 **선택된** 것을 표현합니다. 선택된 심벌은 작고 검게■ 둘러싸입니다



주석 삽입

명령어는 다이아그램의 임의의 장소에서 기입할 수 있습니다. 명령어는 다이아그램 실행에는 영향을 받지 않습니다. 프로그램을 읽기 쉽게하는 것입니다. 명령어를 삽입하기 위해서는 이 버튼을 선택합니다. 그 다음에 명령어를 삽입하고 싶은 장소에서 적당한 장방형을 드래그해서 지역을 정합니다. 명령어 입력에는(*) 등의 기호는 필요 없습니다. 명령어지역의 사이즈는 경계선을 드래그하는 것으로 변경 가능합니다.



이동

다이아그램 중의 심벌을 이동할 경우는 우선 이동하고 싶은 심벌을 선택 합니다. 다음에 선택되어 심벌의 영역을 마우스로 드래그해서 이동시킵니다.

심벌에 접속되어 있는 라인 등의 오브젝트는 심벌이동에 따라 자동적으로 재삽화 되고, 새 심벌장소에 맞추어서 라인은 접속됩니다.



접속라인 그리기

심벌간의 접속라인을 위해 이들 버튼을 도구 바 아이콘에서 선택합니다. 접속포인트에서 아무것도 없는 지역에 접속라인을 그으면, 자동적으로 사용자정의코너에 접속되어 계속 그어져 다른 심벌에 접속하기 쉽게 되어 있습니다.



접속라인 변경

「도구」 - 「라인 이동」 명령어에 따라 선택된 접속라인을 자동적으로 변경할 수 있습니다. 이 명령어는 user 정의코너를 경유하고 있는 경우는 사용할 수 없습니다. 접속라인이 세개의 라인 (종, 횡, 종) 에서 성립되어 있는 경우는 두번째 라인의 이동이 행해집니다. 다음에서 그 예를 나타냅니다.



접속라인 타입 변환

접속라인 끝단을 더블클릭하는 것으로 논리반전 (또는 그 역) 이 간단하게 변경 가능합니다.



LD rung 그리기

새로운 LD rung 을 작성할 경우는 아래와 같이 행합니다.

- 좌 모선을 삽입
- 코일의 삽입(이때 자동적으로 우 모선이 그어집니다)
- 기타 접점과 수직 OR 접속라인의 삽입(직접 rung 에 삽입 가능합니다.)
- rung 이외에 접점과 코일의 삽입(수평라인이 자동으로 좌/우모선에 그어집니다)
- 단, 좌/우모선 사이에 접점과 코일이 배치되지 않으면 수평라인은 그어지지 않습니다.
- rung 상에 삽입된 접점, 코일은 rung 위를 이동할 수 있습니다.
- 접점, 코일을 삽입한때 그어진 수평라인은 선택해서 삭제 가능합니다.

새로운 접점과 코일을 기존 rung 상에 삽입할 수 있습니다



다중 연결

멀티접속라인은 있는 **출력포인트**의 우측에서 만들어집니다. 즉, 다이어그램 가운데서 동일정보가 복수포인트로 **전반**되는 것을 말합니다.

- 우모선
 - 다중접속 수직 (OR) 접속 라인
- (이들 두개의 우측에서부터의 입력 수는 제한이 없습니다.)

A.7.3 FBD 편집

「편집」 메뉴명령어에는 다이어그램을 편집해서 완성시키기 위한 명령어를 포함하고 있습니다. 대개는 선택된 심벌에 대해서 조작이 가능합니다

조작: 다시 고치기

DEL 키가 선택된 심벌의 삭제에 사용됩니다. 「편집」 - 「되돌리기」 명령어에 따라 DEL 키 명령어 후에 삭제된 심벌을 되돌릴 수 있습니다. 그룹 선택된 복수 심벌에 대해서도 DEL 키 사용이 가능합니다.

「편집」 - 「잘라내기」, 「복사」, 「붙이기」 명령어에 따라 선택중의 심벌을 이동, 복사할 수 있습니다

찾기/대치

「편집」 - 「검색, 치환」 명령어에 따라 다이어그램중의 텍스트를 검색, 치환할 수 있습니다. 검색은 완전일치할 경우에만 지원하고 있습니다. 검색은 접점명, 코일명, 블록명, 블록파라미터명, rung 라벨명에 대해서 행해집니다. rung 명령어중의 텍스트 검색에는 사용할 수 없습니다. 치환명령어에 따라 바꿔 놓기는 불가능합니다. 검색방향은 선택 셀의 전후방 어느쪽으로도 행할 수 있습니다.

실행순서 표시

F B D에서는 loop 를 그리기도 하려면 바른 각 심벌의 실행순 (기본은 좌에서 우로, 위에서 아래로) 을 다이어그램에서 판단하기 어려운 경우가 있습니다. 혼란을 피하기 위해 「도구」 - 「실행순 표시」 명령어에 따라 심벌의 실행순을 표시 할 수 있습니다. 실행순은 심벌 가까이에 (n) 으로 표시됩니다.

심벌과 텍스트의 입력

텍스트입력과 변수명을 할당할 경우에는 대상이 되는 심벌을 더블클릭 하는 것으로 변수명 선택용의 다이아로그박스가 열립니다. 심벌이 접점과 코일의 경우는 타입변경(a 접점, b 접점, 시작 검색 등)도 가능합니다. **옵션 - 입력프로젝트** 명령어가

set 되어 있을 경우는 심벌이 배치된 직후에 변수명 등이 할당되도록 다이어로그박스가 열립니다.

「**옵션**」 - 「**매뉴얼 키보드 입력**」 메뉴명령어가 선택되어 있을 시는 변수명과 Function BLOCKS 명 입력을 위해 다음과 같이 1 행의 텍스트박스가 표시되고, 직접 이름을 입력할 수 있습니다. 변수명과 Function 명을 입력한 후에 **ENTER** 키로 입력이 완료됩니다. **ESC** 키로 텍스트박스를 닫을 수 있습니다.이 텍스트박스는 마우스 조작으로는 열수 없습니다.



function Blocks 타입 설정

블록타입을 변경할 경우는, 블록을 더블클릭해서 새로운 타입을 선택합니다. 이대 입력파라미터수의 변경이 가능합니다. (예를 들면, AND,OR,ADD, MUL 연산은 입력파라미터수가 기본에서는 2 이지만 이것을 변경할 수 있습니다.)



공간 삽입

FBD 다이어그램으로 마우스의 오른쪽 버튼을 클릭함에 따라, pop up 메뉴가 표시됩니다. 이 pop up 메뉴에는 커서가 있는 장소에 공간을 삽입 / 삭제하는 다음의 명령어가 포함되어 있습니다.

행의 삽입:.....이 명령어에 따라 커서가 있는 장소에 편집 grid 로 4 행분의 공간이 삽입됩니다.

행의 삭제:.....이 명령어에 따라 커서가 있는 장소의 공간 행이 삭제됩니다. 이 명령어로 FBD element 삭제되는 일은 없습니다.

오른쪽 마우스를 클릭한때에 pop up 메뉴가 열리면 동시에 수평라인이 표시됩니다. 이 장소에서 공백행이 삽입되기도 하고, 삭제되기도 합니다.

A.7.4 보기 옵션

「**옵션**」 메뉴명령어에 따라 F B D 다이어그램 표시방법에 관한 옵션 파라미터를 선택할 수 있습니다.



윤곽

TOPIC35

옵션 - 윤곽 명령어에 따라, FBD 다이어그램 표시방법에 관한 옵션파라미터를 선택할 수 있습니다.



더욱이 zoom 그룹박스내의 설정에는 전체 zoom 비율의 선택이 있습니다. 게다가 zoom 버튼은 도구 바 아이콘에도 포함되어 있습니다.

「**옵션**」 - 「**글꼴**」 메뉴명령어에 따라 폰트타입을 변경할 수 있지만, 폰트사이즈에 관해서는 그래픽화면의 zoom 설정 비율에 따라 자동적으로 조정되기 때문에, 여기에서의 설정은 무효합니다.

A.7.5 스타일과 편집 트래킹

FBD/LD 편집기에서는 FBD/LD 다이어그램 상에서 모든 element 에 대해 그래픽 스타일을 할당할 수 있습니다. 스타일명령어에 따라 element 칼라 할당이 가능합니다. 더욱이 프로그램 버전관리를 목적으로 한 편집 트래킹을 행하기 위해 스타일이 준비되어 있습니다.

또한, 시뮬레이션과 디버거 시에는 모두 칼라 표시된 element 가 보이지 않는 경우가 있습니다. 예를 들면, 빨강, 파랑은 변수의 TRUE/FALSE 를 표시하기 위해 사용 됩니다.

⇒ 미리 정의된 스타일

ISaGRAF 의 FBD/LD 편집기에 설정마침 스타일을 다음에 나타냅니다.

- | | |
|-------------|---|
| 준비 | 기본 스타일은 흑색으로 표시됩니다. 편집 트래킹의 목적에서는, 준비스타일은 오리지날 다이어그램을 나타냅니다. 프로그램 실행시는 이 부분은 스캔 됩니다. |
| 수정종료 | 수정종료 스타일은 핑크색으로 표시됩니다. 편집 트래킹의 목적에서는, 수정끝남 스타일은 오리지날 다이어그램에서 추가, 수정이 추가된 것을 가리킵니다. 프로그램 실행시는 이 부분은 스캔됩니다. |
| 삭제종료 | 삭제종료 스타일은 점선의 회색으로 표시됩니다. 편집 트래킹의 목적에서는, 삭제종료 스타일은 오리지날 다이어그램에서 삭제된 것을 가리킵니다. 프로그램 실행시는 이 부분은 스캔되지 않습니다. |

Custom 설정종료 스타일에 덧 붙여서 custom 스타일을 추가할 수 있습니다. 이 경우, 선택된 element 색을 칼라선택 다이아로그를 사용해서 임의로 설정할 수 있습니다. custom 스타일의 설정에 따른프로그램 실행시의 영향은 없습니다.

수동으로 element 스타일을 변경시킬 경우는, element 를 선택한 후에 「편집」 - 「스타일」 서버메뉴, 혹은 오른쪽 마우스클릭에 따라 pop up 메뉴의 「스타일」 서버메뉴를 사용합니다.

☐ 트래킹 수정

편집 - 스타일 - 편집장소에마크 메뉴명령어에 따라 편집 트래킹의 유효_^무효 바꾸기가 가능합니다. 기본에서 이 설정은 set 되어 있기 때문에 자동적으로 편집 트래킹 됩니다.

편집장소에마크 명령어가 set 되어 있을 시는 변경된 element 는 모두 수정종료 스타일이 됩니다. DEL 키와 cut 조작에서 삭제 된때는 다이어그램 상에서 완전히 삭제 되는 일없이, 삭제 마크로서 삭제종료 스타일(점선에서 회색표시)이 적용됩니다. 이 기능에 따라 프로그램 편집을 모두 남겨두는 것도 가능하게 됩니다.

편집 - 스타일 - 삭제종료아이템 clear 메뉴명령어에 따라, 삭제종료 스타일로 되어 있는 element 가 다이어그램 상에서 완전히 삭제됩니다. 이 명령어는 항상 다이어그램상의 모든 삭제종료 스타일에 적용됩니다.

삭제종료 스타일 element 를 표준스타일 등으로 복원하기 위해서는, 삭제종료 스타일 element 를 선택해서 「편집」 - 「스타일」 - 「표준」 등을 선택합니다. 이 조작에 따라, 오브젝트 접속상의 문법 에러 등이 생길 가능성 (예를 들면, 하나의 접속포인트에 두개 이상의 오브젝트가 접속되는 등) 이 있습니다. 이 에러 검출은 다음 프로그램의 검증이 됩니다.

A.8 텍스트 편집기 사용법

이 장에서는 ISaGRAF의 ST와 IL 언어의 소스코드에서 사용될 텍스트 편집기의 명령어를 설명합니다. 또한 텍스트 편집기는 프로젝트 스크립터, 일기장 파일, 메모장(온라인)으로 언제든지 사용자의 편의에 맞게 사용 가능합니다.

A.8.1 편집 명령어

"**편집**"은 텍스트 편집기에서 자주 사용되는 명령어입니다. 현재 커서위치에 대한 액션을 지정하는 명령어가 포함되어 있습니다.



자르기/ 붙여넣기

DEL 키로 삭제된 텍스트는 「편집」 - 「되돌리기」 명령어로 복원됩니다. 「편집」 - 「잘라내기」, 「복사」, 「붙이기」 명령어는 프로그램중의 텍스트를 이동시키기도 하고, 복사, 삽입하기도 합니다. 또한 다른 어플리케이션 사이에서도 클립보드를 통해서 텍스트 복사 등이 가능합니다.



찾기/ 치환

「편집」 - 「검색」, 「치환」 명령어에 따라 편집중의 텍스트창에서 지정텍스트를 검색 치환할 수 있습니다. 검색방향은 ↑ 방향과 ↓ 방향이 있습니다. 에러 발생시 등에 대상 변수명 등이 검색에 유효합니다.



Go to line

「편집」 - 「점프」 명령어에 따라 편집중 텍스트에 지정된 행수로 점프합니다. 이 기능을 사용하는 것으로, ISaGRAF 코드 생성중 발생한 에러표시가 ST, IL 프로그램 텍스트 행수를 포함하고 있기 때문에 에러장소 검출에 유효하게 됩니다.



변수목록에서 심벌 삽입

「편집」 - 「변수삽입」 명령어에 따라 현커서 장소에 프로젝트로 선언되어 있는 변수모음에서 변수 (심벌)를 선택해서 삽입 가능합니다. 이 변수 선택에는 전용 다이어로그박스가 표시됩니다. 상세한 것은 「프로그램 편집기 공통항목」을 참조 바랍니다..



파일 삽입

「**편집**」 - 「**파일 삽입**」 명령어에 따라 현커서 장소에 텍스트 파일의 삽입이 가능합니다. 단, 이 명령어에서는 텍스트 파일 이외의 파일은 다룰 수 없습니다.

A.8.2 옵션

옵션 메뉴명령어에는 도구바의 표시/비표시선택, 키워드의 표시/비표시, 폰트의 지정이 포함되어 있습니다. 각각으로 선택된 폰트는 work bench 상의 텍스트 편집기로 유효합니다.

ST / IL 프로그램 소스 코드 입력 때 「**옵션**」 - 「**키워드 표시**」 명령어에 따라 ST, IL 프로그램은 각각 공통의 키워드 도구 바 표시를 사용할 수 있습니다. 도구 바 상의 버튼을 클릭하면 현커서 위치에 키워드 삽입이 가능합니다.

A.9 프로그램 편집기 상세 정보

A.9.1 다른 ISaGRAF 도구 호출



검증

파일 - **검증** 명령어에 따라 ISaGRAF 코드 생성이 실행되고, 편집중의 프로그램문법을 체크합니다. 끝끝핍프로그램 에서는 레벨 1, 2 양쪽이 체크됩니다. 문법 체크가 완료되면 코드생성 창이 종료되고, 다음 조작으로 이동합니다. 만약, 프로젝트가 현재 편집중인 하나의 프로그램만에서 구성되어 있을 경우는 어려가 없으면 어플리케이션 코드도 생성됩니다.

옵션 - **컴파일러옵션** 명령어에 따라 컴파일러와 최적화 옵션을 설정하기 위해 사용됩니다. 또한, 이들에 관한 상세한 설명은 「코드생성의 사용법」을 참조 바랍니다.



시뮬레이션 & 디버깅

파일 - **시뮬레이션**, **디버깅** 명령어에 따라 ISaGRAF 그래픽 디버깅가 시뮬레이션 모드 또는 온라인 모드로 작동합니다. 이때, 각 프로그램 편집기는 모두 디버깅모드로 다시 오픈 됩니다. 이 디버깅모드에서는 프로그램의 직접 변경은 할 수 없습니다. (한번 디버깅모드에서 빠져 나와 프로그램 편집기를 통상 모드로 열 필요가 있습니다.) 상세한 것은 「그래픽디버깅의 사용법」을 참조 바랍니다.



변수목록 편집

파일 - **변수목록** 명령어에 따라 현재 편집 프로젝트 변수모음 편집기를 작동합니다. 또한 이 모음 편집기에서 사용자정의 워드편집도 가능합니다. 상세한 것은 「변수목록 편집기 사용법」을 참조 바랍니다

TOPICZ12

A.9.2 파라미터

편집중 프로그램이 Function, Function BLOCKS, 서버프로그램일때 **파일 - 파라미터** 명령어에 따라 프로그램 입출력 파라미터를 정의할 수 있습니다. 만약, 편집중의 프로그램이 SFC 프로그램과 Begin, End 섹션의 top 레벨 프로그램일때 이 명령어는 무효가 됩니다.

Function, Function BLOCKS, 서버프로그램은 최대 3 2 개의 입출력 파라미터 (합계) 를 가질 수 있습니다. Function 과 서버프로그램은 항상 한 개의 출력파라미터밖에 가질 수 없습니다. 이 출력파라미터명은 반드시 그 프로그램명과 일치해야 할 필요가 있습니다. 파라미터정의를 위해 다음의 다이아로그박스가 사용됩니다.

파라미터창 상부에는 서버프로그램 파라미터가 위에서부터 입력파라미터, 마지막에 출력파라미터순으로 나타냅니다. 하부에는 선택되어 있는 파라미터가 해설됩니다. 파라미터명을 붙이는 방법에는 아래의 룰이 있습니다.

- 파라미터명은 1 6 반각문자 이내일것.
- 최초문자는 영문자 (a - z) 일것.
- 이후의 문자는 영문자, 숫자일것.
- 파라미터명은 대-소문자 구별은 없음.

「**삽입**」 버튼에 따라 선택되어 있는 파라미터앞에 새로운 파라미터를 삽입합니다. 「**삽입**」 버튼에 따라 선택되어 있는 파라미터를 삭제합니다. 「**정렬**」 버튼에 따라 파라미터를 자동적으로 sorting 해서 리턴파라미터가 마지막에 오도록 바꿔나열할 수 있습니다. 「**확인**」 버튼에 따라



서버프로그램 인터페이스 정의 (즉, 파라미터구성) 를 보관해서 창을 닫습니다. 「취소」버튼에 따라 수정내용은 보관 되지 않고 창은 닫힙니다.

A.9.3 기타 공통적인 「파일」 메뉴

다음에 프로그램 편집기 「파일」 메뉴명령어로 기타 공통적인 항목을 ListUp 합니다



다른 프로그램 열기

「파일」 - 「열기」 명령어에 따라 현재 편집중인 프로그램을 닫고, 새로운 같은 언어의 다른 프로그램을 열수 있습니다. 다른 언어로 기술된 프로그램은 열수 없습니다.



Printing the program

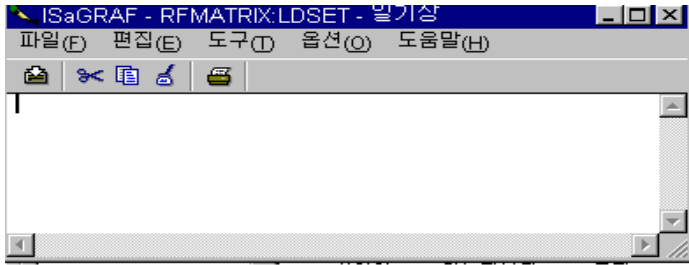
파일 - 인쇄 명령어에 따라 편집중인 프로그램은 인쇄가 불가능합니다. 단, 이 **인쇄** 명령어에서는 프로그램 리스트와 내부변수가 인쇄됩니다.

S F C F B D QuickLD 편집기에서는 「편집」 - 「메타파일로 복사」 명령어에 따라 편집중인 다이어그램을 W i n d o w s 메타파일로서 클립보드에 복사할 수 있습니다. 이 내용은 다른 어플리케이션에 첨부할 수 있습니다. S F C 프로그램에 있어서는 레벨 1 (차트, 레퍼런스번호, 명령어) 이 그 대상이 됩니다..

A.9.4 일기장갱신

파일 - 일기장 명령어에 따라 수정이력 파일의 편집이 가능합니다. 대개 이 파일에는 코드생성 등이 될 때, 문법체크 내용과 코드 생성된 일시 등이 자동적으로 추가되도록 되어 있습니다.

만약, 「**옵션**」 - 「**일기장 갱신**」 옵션이 set 되어 있으면, 변경내용을 디스크에 보관하기 위해, 다음의 다이어로그박스가 열립니다.



「확인」 버튼을 선택하면 입력된 텍스트는 수정이력 파일에 날짜/시간 stamp 붙임이 추가되어 보존됩니다. 이 기능은 프로그램의 유지상 유효하게 활용 가능 합니다.

A.9.5 변수목록에서 변수선택



S T / I L 프로그램 편집 중에는 「편집」 - 「변수 삽입」 명령어에 따라 편집중인 프로그램으로 현재 커서에 모음으로 등록되어 있는 변수명은 삽입이 가능합니다. 또한, QuickLD 에서는 「편집」 - 「텍스트 / 심벌설정」 명령어 (혹은 심벌과 지정된 영역을 더블클릭) 에 따라 다이어그램중의 접점, 코일, 블록 입출력 파라미터등에 변수명의 할당이 가능합니다. 또한, F B D 편집기에서는 심벌을 더블클릭함으로써 다이어그램중의 접점, 코일, 블록 입출력 파라미터등에 변수명의 할당이 가능합니다.

[스코프]용의 선택 변수의 범위를 글로벌, local 에서 선택 가능합니다. 오른쪽 위의 변수형 선택이 가능합니다. 옆의 4 개의 아이콘은 이 변수형을 선택하기 위해 shortcut 해서 사용할 수 있습니다.



.....ture 형



.....정/실수형

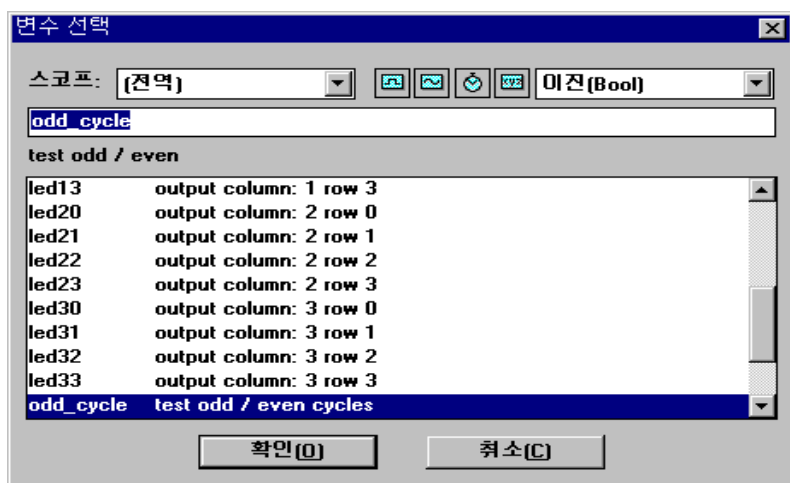


.....시변형



.....가변 문자열형

변수선택을 하기 위해, 리스트 가운데서 변수명을 선택해서 (클릭) 선택된 변수명과 그 케맨드가 선택 bo 확인 s (리스트의 상부) 에 표시된 것을 확인합니다. 다음에[확인]버튼을 선택하면, 선택된 변수명을 뽑을 수 있습니다. 또한, 변수명 더블클릭하든지, 선택버튼에 직접 변수명을 입력하면 같은 조작을 할 수 있습니다.



A.10 변수목록 편집기 사용법

변수, function Blocks, 예약어 는 소스 코드에서 사용 전에 반드시 변수목록에 선언 되어야 합니다. 변수 ,예약어 는 다음의 어느 언어 에서도 사용 가능 합니다: SFC, FBD, LD, ST , IL.
FBD 에서 사용할 Function Blocks 은 미리 선언 되지 않아도 됩니다. 왜냐면 ISaGRAF FBD , QuickLD 가 자동적으로 선언을 하기 때문 입니다..

변수

변수는 영역(range) 또는 타입에 따라 분류 됩니다. 동일한 영역과 타입의 변수만이 동일한 입력란에 설정 가능 합니다. 다음은 기본적인 영역(range)입니다.

-  **전역** 현재 프로젝트내의 어떤 프로그램에서도 호출 가능
-  **지역** 한 프로그램에서 만 사용가능

기본적인 타입의 소개 입니다

-  **BOOLEAN** TRUE/false 이진값
-  **ANALOG** 실/정수 값
-  **TIMER** 타임값(시변수)
-  **MESSAGE** 문자열

변수는 이름(변수명), 주석, 속성, 네트워크 주소...등으로 분류 합니다.
다음은 기본적인 변수의 속성 입니다.

- INTERNAL** 메모리 변수
- INPUT** 입력디바이스와 연결된 변수
- OUTPUT** 출력디바이스와 연결된 변수
- CONSTANT** 읽기전용 내부 변수

Note: **타임값(시변수)** 은 항상 **내부** 변수 입니다. **입력** 과 **출력** 은 항상 **전역** 영역 입니다.

예약어

예약어는 문자열을 대신하는 별명(별명)이다. 예약어는 영역에 따라 분류 되며 동일한 타입,영역에 의해서만 동일한 입력란에 설정 가능 합니다. 다음은 기본 영역입니다.

-  **공통**모든 프로젝트에서 가능
-  **전역** 현재 프로젝트내의 모든 프로그램에서 가능
-  **지역** 하나의 프로그램에서만 가능



Function Blocks

ST 와 IL 언어를 사용하는 function Blocks 의 예는 반듯이 변수목록에 선언 되어 있어야 합니다. 이유는 function Blocks 은 내부 "숨김" 데이터로 각각의 function Blocks 을 구분 해야 하기 때문이다. 아래의 예제는 라이브러리에 정의 되어 있는 function Blocks "R_TRIG" (rising edge detection)을 이용하는것을 보여준다. 각각의 Blocks 유일한 이름 으로 구분 된다. 라이브러리 관리자를 이용하여 각각의 이름을 구분하여 설정 해 놓는다.:

BLOCKS name: R_TRIG
파라미터: Input=CLK
Output=Q

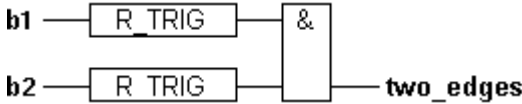
변수목록 편집기를 이용하여 이름 붙이기 예

Instance name:	TRIG_B1	BLOCKS	name:
	R_TRIG		
Instance name:	TRIG_B2	BLOCKS	name:
	R_TRIG		

ST 프로그램에서 사용된 선언 예..:

```
TRIG_B1 (b1);
edge_b1 := TRIG_B1.Q;    (* b1 variable edge detection *)
TRIG_B2 (b2);
edge_b2 := TRIG_B2.Q;    (* b2 variable edge detection *)
```

선언된 function Blocks 은 **전역** 또는 **지역** 이다. FBD 또는 LD 에서 사용된 Function Blocks 은 미리 정의 할 필요는 없는데, 이유는 ISaGRAF FBD 가 자동으로 사용된 BLOCKS 을 설정 하기 때문 이다.



(* function Blocks 은 항상 라이브러리에 정의된 이름을 사용한다. ISaGRAF FBD / QuickLD 편집기는 자동적으로 선언을 한다. *)

FBD 또는 LD 에서 자동적으로 정의된 Function Blocks 은 항상 **[지역]** 으로 선언된다.

네트워크 주소

네트워크 주소는 옵션의 설정입니다. 0 이외의 주소를 설정하면 외부시스템 (HMI 시스템 등) 에서 이 변수를 감시할 수 있습니다. 네트워크 주소는 각각의 변수에 설정되어야만 합니다. 「도구」 - 「주소 재설정」 명령어에 따라 네트워크 주소의 번호부치기를 자동으로 할 수 있습니다. 이 명령어 실행전에 번호를 부치고 싶은 변수를 리스트에서 변수목록에 선택해 둘 필요가 있습니다. 16 진수로 **pace** 주소 (그룹의 선두 주소) 를 입력 함으로서 선택된 변수가 **연속번호**로 나뉘어 부쳐집니다. pace address 에 0 을 입력하면 선택중인 변수의 네트워크 주소는 전부 0 으로 set 됩니다.

A.10.1 변수목록 메인 창

변수목록 편집기 의 메인 창에서는 동일타입으로 동일범위의 변수리스트가 표시됩니다. 변수의 타입과 범위는 항상 창 타이틀 바에 표시됩니다. 아래에서 창을 나타냅니다.

이 창에서는 변수에 관한 주된 필드를 나타내고 있습니다. 즉 이름, 속성, 네트워크 주소, 텍스트 명령어 입니다. 변수 각각의 완전한 기술은 창 하부의 상태바에 표시됩니다. 또한, 각 필드의 폭은 필드사이를 마우스로 드래그(마크가 표시)하면 폭을 변경할 수 있습니다.

다음의 도구 바 아이콘에 따라 변수범위를 선택할 수 있습니다.

-  **공통** 어떤 프로젝트에서도 다를 수 있습니다.
-  **전역** 프로젝트내의 프로그램에서도 다를 수 있습니다.
-  **지역** 하나의 프로그램에서만 다를 수 있습니다.

아래의 더블컨트롤에 따라 변수타입을 선택할 수 있습니다

Booleans	Integers/Reals	Timers	Messages	FB instances	Defined words
Name	Attrib.	Addr.	Comment		

 도구 바 아이콘 좌측의 텍스트 입력필드에 입력하는 것으로, 입력된 텍스트가 선두에 포함되어 있는 변수명 검색을 합니다. 검색대상은 표시중인 변수 전체입니다. 「찾기」 메뉴인 두개의 방향성을 가진 「찾기」 명령어에 따라 검색하고 싶은 문자열을 포함한 변수명 또는 명령어 내용을 검색합니다. 변수가 나타나면 포커스가 리스트 가운데서 검색된 변수명 으로 이동합니다. 검색에는 **대소문자의 구별은 없습니다**. 검색에는 **전방검색 (↓)** 과 **후방검색 (↑)** 이 있습니다.

A.10.2 변수 관리

「파일」 메뉴명령어 은 선택중인 변수그룹에 대한 기능을 합니다. 「파일」 - 「다문것」 명령어 에서는 선택중인 변수범위와 타임변경이 가능합니다.



인쇄

「파일」 - 「인쇄」 명령어 에 따라 현재 표시되어 있는 변수 (또는 정의워드) 의 인쇄가 가능합니다. 인쇄항목에는 변수에 정의한 내용도 모두 포함됩니다.



새로운 변수 등록

「편집」 - 「새변수」 명령어 에 따라 새로운 변수, Function BLOCKS instance, 정의워드를 작성할 수 있습니다. 새 변수는 리스트상에서는 현재 선택중인 변수 앞에 삽입됩니다. 이 명령어 이 선택되면, 변수정의 다이어로그박스가 열립니다. 필요한 사항을 기술, 선택후[저장]버튼을 선택하면 리스트에 삽입됩니다. 이때 다이어로그박스는 다시 열리고, 계속해서 변수를 추가합니다.[취소]버튼으로 변수설정을 취소 해서 되돌립니다.



수정

「편집」 - 「편집」 명령어 에 따라 리스트 가운데 선택중인 변수의 기술수정이 가능합니다. 「신규작성」 명령어 과 같은 모양으로 변수의 기술입력을 합니다. [저장] [취소]버튼 이외에 [다음으로] 및 [앞으로] 버튼이 있고, 지속적인 변수수정은 불가능합니다.



자르기/붙여넣기

ISaGRAF 변수목록 편집기에서는 **복수행의 동시선택**이 가능하고, 선택된 내용에 대해서 다음의 명령어 도 다를 수 있습니다.

복사 선택된 복수변수를 모음클립보드로 복사합니다.

잘라내기 선택된 복수변수를 모음클립보드로 복사해서 편집리스트에서 삭제합니다.

삭제 선택된 복수변수를 편집리스트에서 삭제합니다.

붙이기 모음클립보드의 내용을 리스트에 삽입합니다.

복사 / 잘라내기 / 붙이기 명령어 은 동일 타입, 변수사이의 조작에 한합니다.



네트워크 주소 설정

정의 워드의 편집중에 **도구 - TREU/FALSE 정의 가져오기** 명령어 에 따라 TRUE, FALSE 형 변수에 나눠부쳐진 TRUE, FALSE 상태의 키워드를 정의워드로서 바꿀 수 있습니다. 이들 변수(TRUE,FALSE 라고 정의된)는 주로 디버깅 시에 사용됩니다. 프로그램 중에 변수로 사용되어지는 것이 있으면 정의워드로 준비해 둘 필요가 있습니다.

이 명령어 은 설정종료의 TRUE, FALSE 형 변수 중에 TRUE/FALSE 문자열을 검색하는 것으로 TRUE/FALSE 정의워드를 작성합니다.



참거짓 문자열 가져오기

정의 워드의 편집중에 **도구 - TREU/FALSE 정의 가져오기** 명령어 에 따라 TRUE, FALSE 형 변수에 나눠부쳐진 TRUE, FALSE 상태의 키워드를 정의워드로서 바꿀 수 있습니다. 이들 변수(TRUE,FALSE 라고 정의된)는 주로 디버깅 시에 사용됩니다. 프로그램 중에 변수로 사용되어지는 것이 있으면 정의워드로 준비해 둘 필요가 있습니다.

이 명령어 은 설정종료의 TRUE, FALSE 형 변수 중에 TRUE/FALSE 문자열을 검색하는 것으로 TRUE/FALSE 정의워드를 작성합니다.

A.10.3 오브젝트 요약

변수목록 오브젝트 [변수, Function BLOCKS instance, 정의워드] 빠짐없이 기술할 필요가 있습니다. 기술 필드는 각각의 오브젝트 타입이라도 구별합니다.

다음에서는 변수에 대한 공통적인 필드를 나타냅니다.

- 이름** 변수명으로 최초의 문자는 반각영문자(a - z)로 후반각 영문자, 숫자 중에 하나는 필요합니다.
- 네트워크 주소** 16 진수 네트워크 주소는 옵션에서 다릅니다. 제로이외의 숫자가 입력된 변수는 타겟이 실행시에 외부의 SCADA 시스템에서 모니터가 가능합니다.
- 명령어** 변수명령어 의 자유영역
- 유지** 유지지정은 배터리백업된 메모리에 대해 scan 때마다 변수를 보관하는 것을 의미합니다. (입력, 출력변수에 이 보지지정은 할 수 없습니다.)



이진 변수 설정에 필요한 것들..:

- 속성** 내부, 입력, 출력, 정수 중에 설정
- "False" 문자열** 디버그 시에 FALSE 상태 시에 대입되는 문자열
- "TRUE" 문자열** 디버그 시에 TRUE 상태 시에 대입되는 문자열
- 초기화 TRUE** 이 옵션을 체크하면 초기화 치를 TRUE 로 설정가능합니다.



정/실수형 변수에 필요한 것들..

- 속성** 내부, 입력, 출력, 정수 중에 설정
- 형식** 정수 또는 실수선택. 이 형식은 디버그 시에도 활용가능합니다.
- 유니트** 디버그 시에 사용할 단위를 나타내는 문자열
- 변환** 입/출력변수에 설정할 변환함수, 변환테이블의 이름
- 초기치** 변수의 초기치 (정수, 실수 형식을 맞출 필요가 있습니다.)



타임변수에 필요한 필드:

- 속성** 내부, 정수의 설정
- 초기치** 변수의 초기치



문자열형 변수에 필요한

속성 내부, 입력, 출력, 정수 중에 설정.

최대장 최대 문자열 길이를 지정합니다.

초기치 변수의 초기치



정의위드에 필요한 필드:

이름 ST 스테이트먼트에 사용된 명칭, 최초 문자는 반각 영문자(a-z)로 후반각 영문자, 숫자 중에서 필요합니다.

정의 ST 언어 문법에 있던 문자열로 정의위드를 컴파일 앞에 이 문자열로 바뀌들 수 있습니다.(예: Name = PI - Equivalence = 3.14159)

명령어 자유로운 명령어



Function BLOCKSinstance 에 필요한 필드

이름 instance 명, ST 스테이트먼트로 사용됩니다. 최초의 문자는 반각영문자(a-z)로 후 반각영문자, 숫자 중에서 필요합니다.

타입 대응할 라이브러리 내의 Function BLOCKS 명

명령어 자유로운 명령어

A.10.4 빨리 만들기

[도구] - [빨리 만들기] 로 복수변수를 한번에 정의할 수 있습니다. **빨리만들기**로 작성된 변수에는 통과번호가 붙어 있습니다. 다음의 항목을 설정해 주세요.

-변수에 매겨진 처음과 마지막 번호

-변수명의 처음과 마지막에 기입된 문자열

-변수명에 사용된 숫자의 수 (문자수)

또한 변수의 기본속성(내부, 입력, 출력 0 과 변수타입에 따른 고유의 정보(정/실수형의 유지 와 가변문자열의 문자열 최대장) 등 도 설정 가능합니다.

변수명은 숫자로 시작할 수 없기 때문에 반드시 첫머리에 문자열을 설정해야 합니다. 「행수」 (숫자의 문자수) 가]auto]으로 되어 있으면 자동적으로 최소의 문자수가 할당됩니다. 「행수」가 설정되면 첫머리부터]0]을 넣어 행수가

정리됩니다. 이와 같이 변수명의 문자수를 고정시켜 두면, short 시에 편리합니다.
다음에 그 편의의 예를 열거합니다. :

예: 문자수가 변하는 예:

변호 매김:

시작: 1 끝: 1

Digits: auto

심벌:

이름: V ##

이 설정에서는 다음의 3 가지 변수가 가능합니다.

Var9xx Var10xx var11xx

예: 문자수가 고정되는 예

변호 매김:

시작: 1 끝: 100

Digits: 3

심벌:

이름: JYG ##

A.10.5 Modbus SCADA 주소

(네트워크 주소)는 ISaGRAF 과 SCADA 소프트웨어의 사이 등에서 Modbus protocol 을 사용해서 통신할 경우에 사용됩니다. 이 경우 SCADA 소프트웨어쪽이 마스터가 되고, ISaGRAF 타겟이 slave 가 됩니다. 네트워크 주소는 SCADA 쪽에서 컨트롤 되는 변수의 가상 mapping 해서 사용됩니다. **도구 - SCADA Modbus 주소」** 을 사용하면 각 변수를 간단히 매핑 할 수 있습니다.

SCADA Modbus 주소 map 다이어로그박스에는 두개의 리스트가 있습니다. 위의 리스트는 (segment) 0~*FFF=4096 개 분의 mapping 정보로 네트워크 주소가 할당되어 있는 변수가 표시됩니다. 아래의 리스트에는 mapping 되어 있지 않는 변수가 표시되어 있습니다. 네트워크 주소 에는 변수를 mapping 할 수 없습니다.

편집 - 선택변수 map 과 **map 에서 삭제** 로 변수가 리스트 사이로 이동해서 mapping 합니다. 리스트 되어 있는 변수를 더블클릭해도 같은 형식으로 처리 가능 합니다. segment dropdown 리스트박스에서 다른 segment 의 리스트로 바꿀 수 있습니다.

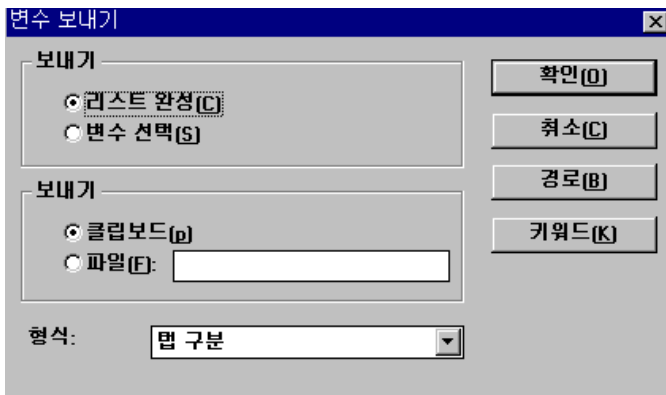
옵션 메뉴로 주소 10/16 진 표시를 바꿉니다.

「**편집**」 - 「**찾기**」 메뉴로 변수가 mapping 되어 있는지 어떤지를 검색할 수 있습니다.

A.10.6 다른 프로그램과 정보 교환

ISaGRAF 변수목록 편집기 에는 다른 어플리케이션과 정보교환을 목적으로 가져오기 / 보내기 를 갖추고 있습니다. 「**편집**」 - 「**텍스트 보내기**」 명령어 에 따라 변수리스트를 ASCII 텍스트 기술로 변환시켜, 클립보드 또는 파일 형식으로 보관합니다. 「**편집**」 - 「**텍스트 가져오기**」 명령어 에 따라 구조를 형식으로 클립보드 또는 파일 형식으로 보관된 내용에서 변수 설정내용의 필드를 꺼냅니다.

☞ 데이터 보내기



「**텍스트 보내기**」 명령어 에 따라 보내기 텍스트 다이아로그박스가 열립니다. 여기서 보내기 메커니즘을 설정합니다.

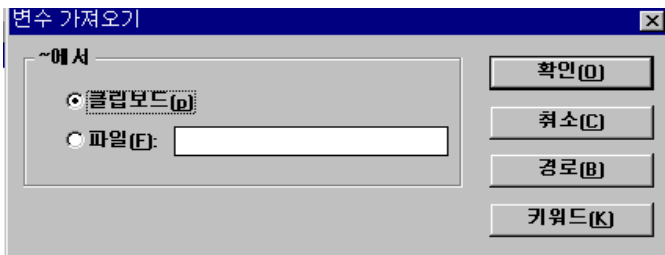
모든 리스트 를 ON 하는 것으로 현재 선택되어 있는 변수리스트는 무시되고, 리스트 전부가 보내기됩니다.**선택된 변수** 를 ON 하는 것으로 선택된(하이라이트 부분)변수만이 보내기 됩니다.

클립보드 를 ON 하는 것으로 보내기 되는 정보는 ASCII 텍스트 형식으로 창 클립보드에 보관됩니다. 이후 [베스트] 명령어 으로 다른 어플리케이션을 꺼 낼 수 있습니다. **파일** 을 ON 하는 것으로 보내기 되는 정보는 ASCII 파일로서 보관됩니다. 이때 pass 명부착 파일명을 지정할 필요가 있습니다.**일관** 버튼에 따라 디렉토리 내용을 체크하는 것도 가능합니다.

또한, 보내기 어스키 텍스트의 형식을 선택합니다. 형식의 종류에 대해서는 나중에 서술합니다. **확인**버튼으로 보내기되고,**취소** 버튼으로 보내기 되지 않게 다이아로그박스는 닫힙니다.

보내기된 테이타에는 필드명이 추가되어 있습니다. 데에타 최초의 행은 필드명을 나타내고 있습니다. 각각의 변수 오브젝트 데이터는 1행의 텍스트로 표현되고 있습니다. 행의 마지막은 MS - DOS 표준의[0d-0a]가 됩니다. 최초의 행 필드명으로 불리는 항목은[키워드]버튼을 선택하면 변경가능 합니다.[키워드]에 관해서는 나중에 서술합니다. 아무것도 하지 않는 경우는 기본파일명이 붙어 있습니다.

☞ 데이터 가져오기



「**텍스트가져오기**」 명령어 에 따라 가져오기 텍스트 다이아로그박스가 열립니다. 여기서 가져오기 메커니즘을 설정할 수 있습니다..

클립보드 을 ON 하는 것으로 가져오기된 정보는 ASCII 텍스트 형식 상태로 창의 클립보드에서 집어냅니다. **파일**을 ON 하는 것으로 가져오기된 정보는 ASCII

파일에서 읽어 냅니다. 이때 pass 명부착 파일명을 지정할 필요가 있습니다. **입관** 버튼으로 디렉토리 내용을 체크할 수 있습니다.

또한, 가져오기 ASCII 텍스트로 사용되고 있는 세퍼레이터는 자동적으로 인식해서 꺼냅니다. 형식의 종류에 대해선 나중에 서술합니다. **확인** 버튼으로 가져오기되고 **취소** 버튼으로 가져오기 되지 않게 다이아로그박스는 닫힙니다.

가져오기된 데이터에는 필드명이 추가 되어 있어야 합니다. 데이터 최초의 행은 필드명을 나타냅니다. 각각의 변수 오브젝트 데이터는 1 행의 텍스트로 표현됩니다. 행의 마지막은 MS - DOS 표준 0d-0a 가 됩니다. 필드명 나열순의 특별한 지정은 없습니다. 만약 필드가 빠져 있을 경우, 그 필드 데이터에는 기본치가 들어갑니다. 만약, 가져오기된 데이터가 이미 리스트 중에 있을 경우에는 겹표지에 쓸 것인지 여쭈어볼 것인지는 체크할 필요가 있습니다. 이때 빠져 있는 필드가 있으면 그 필드 데이터는 갱신되지 않습니다. 최초 행의 필드명으로 불리는 항목은[키 워드]버튼을 선택하면 변경 가능 합니다.[키워드]에 관해서는 나중에 서술합니다. 아무것도 하지 않는 경우는 기본 필드명이 붙어 있습니다.



텍스트 형식

다음에서는 보내기 명령어 에 따라 만들어진 텍스트 파일의 예를 나타냅니다. 보내기 명령어 에서 형식은 자동 인식됩니다.

□ 터부세퍼레이터

설명: 형식사이가 더블로 구분됩니다.

예:

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

□ comma 세퍼레이터

설명: 형식 사이가 comma 로 구분됩니다.

예:

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

□ 세미콘세퍼레이트

설명: 형식 사이가 세미콘으로 구분됩니다.

예: Name;Attribute;Comment
level;internal;internal calculated water level
alarm1;output;main alarm output

□ comma 와 쌍따옴표

□ 설명: 필드 사이가 comma 로 구분되고, 각 필드명은 쌍따옴표로 둘러싸입니다.

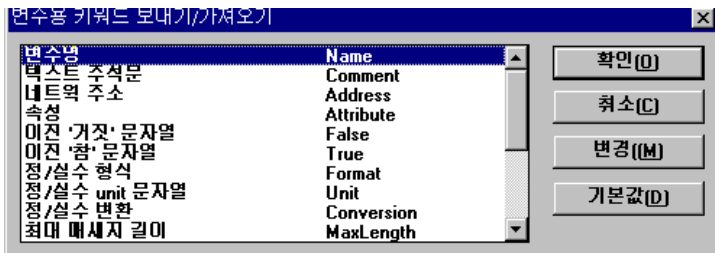
예: "Name","Attribute","Comment"
"level","internal","internal calculated water level"
"alarm1","output","main alarm output"



키워드

보내기/가져오기의 처음 행은 필드명이라 하고, 키워드에 등록되어 있습니다. 또한,

[키워드] 버튼으로 키워드 다이어로그박스가 열리고, 여기에서 변경 가능합니다



키워드 다이어로그박스는 오브젝트명과 키워드가 어 있습니다. 키워드 를 변경할 경우는 변경하고 싶은 필드를 선택해서 **수정** 버튼을 누릅니다. 여기에 새로운 키워드를 입력합니다. **기본** 버튼은 원래의 키워드로 변경합니다.이 키워드 붙이는 법에는 다음의 룰이 있습니다.

- 16 반각문자 이내
- 처음 문자는 반각영문자(a~z)
- 연결 문자는 영문자, 숫자 중에
- 동일 키워드는 다른 형식에는 적용 불가능합니다.

다음에서는 ISaGRAF 기본 키워드를 나타냅니다.

변수명.....	Name
텍스트 명령어	Comment
네트워크 주소	Address
속성 (내부, 입력, 출력).....	Attribute
'False' 문자열.....	False
'TRUE' 문자열.....	TRUE
정/실수 형식.....	Format
정/실수 단위.....	Unit
수치 변환명.....	Conversion
문자열 최대장	MaxLength
FB 라이브러리 타입	라이브러리
공식 정의	Equivalence
내부속성	Internal
입력속성	Input
출력속성	Output
정수속성	Constant
실수형 형식.....	Real
정수형 형식.....	Integer

A.11 I/O 연결 편집기 사용법

I/O 연결 편집기로 어플리케이션 내의 I/O 변수와 Target 의 I/O 보드채널을 매칭 할 수 있습니다. 그러므로 사용자는 I/O 보드 선택·설정을 해서 채널에 I/O 변수를 할당해 갑니다.

I/O 연결 편집기 좌측은 타겟 보드 슬롯을 나타냅니다. 슬롯은 빈 슬롯이라도 좋고, 싱글 I/O 보드 또는 I/O 장치 기기를 나타낼 수 있습니다. 각 슬롯에는 번호가 붙어 있습니다. **최대 255장** 의 보드를 나타낼 수 있습니다. 우측에는 보드의 파라미터와 보드 채널에 접속되어 있는 변수를 나타내고 있습니다. 한장의 보드에는 **최대 128 개의 I/O 채널**을 연결 할 수 있습니다



아이콘

보드 우측에 표시되어 있는 아이콘은 보드채널에 접속되는 변수타입과 속성을 나타냅니다. ISaGRAF 에서는 하나의 보드에 다른 타입의 변수를 접속할 수 없습니다. 다음 예서는 아이콘과 그 의미를 나타냅니다:

-이진(boolean) 타입
-실.정수 타입
-메세지(문자열) 타입
-입력 - 채널 접속이 않된 상태
-출력 - 채널 접속이 않된 상태
-입력 - 최소 1 채널 이상 연결 상태
-출력 - 최소 1 채널 이상 연결 상태

다음은 슬롯에 들어 있는 I/O 보드 또는 디바이스 타입을 나타내는 아이콘을 가리킵니다.:

-I/O 장치기기 (복수 I/O 보드 모여있음)
-실제 I/O 보드
-가상 I/O 보드

다음은 보드 파라미터와 채널을 나타내는 아이콘을 가리킵니다:

-보드 파라미터
-개방 채널

 연결 채널



리스트 내에서 보드 이동

「편집」 - 「위로 이동」, 「아래 이동」에 따라 선택중인 I/O 보드를 상하로 이동할 수 있습니다. 도구바 아이콘에서도 조작가능 합니다. 「편집」 - 「슬롯 추가」에 따라 선택중인 I/O 보드 바로 앞의 빈 I/O 슬롯을 삽입할 수 있습니다

A.11.1 I/O 보드 정의

「편집」 메뉴에는 선택된 보드를 정의하기도 하고, 보드 채널에 I/O 변수를 할당하기도 하는 명령어가 포함되어 있습니다.



I/O 보드 타입 선택

편집 - 보드/장비 설정에 따라 사용할 보드를 선택할 수 있습니다. 여기에서는 ISaGRAF 라이브러리에서 단일보드 또는 복수보드가 모여진 장치기기를 선택할 수 있습니다. 해당 슬롯을 더블클릭해도 똑같이 보드선택이 가능합니다.

하나의 보드의 모든 채널은 동일타입 (이진형, 정수/실수형, 메시지형) 과 동일 방향성을 (입력, 출력) 가집니다. 실수 아나로그, 정수 아나로그의 구별은 I/O 변수시 에는 고려되지 않습니다. I/O 장치기기의 경우에는 다른 타입과 방향성을 가진 채널을 포함하게 됩니다. 표현방법으로서 복수의 싱글 I/O 보드가 편성 됩니다. 단, 점유 슬롯수는 하나가 됩니다.



보드 삭제

「편집」 - 「슬롯해제」에 따라 선택되어 있는 슬롯에 할당된 보드를 삭제할 수 있습니다. 이미 그 보드의 채널에 I/O 변수가 접속되어 있는 경우는 그것은 모두 자동적으로 분리됩니다



실제 - 가상 보드

「편집」 메뉴의 「실제 / 가상보드」에 따라 선택된 보드 모드를 바꿀 수 있습니다.:

 실제 I/O 보드

 가상 I/O 보드

실제 모드에서 I/O 변수는 직접 접속되고 있는 I/O 디바이스에 연결 됩니다. 즉 어플리케이션 프로그램으로의 입/출력에 따라 real I/O 디바이스 상태가 대응하고 있습니다.

가상 모드에서 I/O 변수는 바로 내부변수로서 행동합니다. 디버그에 따라 모니터하기도 하고, 변경하기도 하는 것으로 I/O 처리의 시뮬레이션을 합니다. 실제 I/O 디바이스의 대응은 없습니다.



메모장

[도구] - [메모장]에 따라 선택된 I/O 보드 또는 장치기기의 메모장이 열립니다.

메모장 I/O 보드 (드라이버) 라이브러리에 따라 기술됩니다. 여기에서는 파라미터의 의미 즉 처음 I/O 보드관리에 관해 필요한 정보가 모두 포함되어 있습니다.



연결변수 연결해제

「도구」 - 「채널 분리」에 따라 선택되어 있는 슬롯 I/O 보드에 접속되어 있는 모든 I/O 변수를 분리 (해제) 합니다.

A.11.2 보드 파라미터 설정

보드 파라미터를 설정 하려면 우측 보드파라미터 아이콘 (I/O 보드에 따라서는 없는 경우도 있습니다.) 을 더블클릭 합니다. 또는 「편집」 - 「채널/파라미터 설정」에 의해서도 가능합니다. 보드파라미터의 아이콘은 아래에서 나타냅니다:

보드 파라미터

이 파라미터 아이콘은 I/O 보드 드라이브 개발시에 필요한 데이터 (예를들면 보드 I/O 주소 등) 를 사용자가 설정하기 위한 것으로 모든 I/O 보드에 대해서 존재하고 있는 것은 아닙니다. (상세한 것은 I/O 보드 메모장을 참조 바랍니다.)

I/O 채널 접속을 하기 위해서는 접속할 채널을 더블클릭 하든지, 「편집」 - 「채널/파라미터 설정」을 실행합니다. 채널 상태에는 아래의 명령어가 사용됩니다.:

개방 채널

연결 채널

I/O 채널 접속 다이어로그 박스에서는 그 채널의 타입과 방향성이 일치된 I/O 변수로 아직 접속되지 않는 것이 선택지로 리스트 됩니다. **[연결]**버튼으로 선택되어 있는 변수를 그 채널에 접속할 수 있습니다. **[해제]** 버튼으로 이미 선택되어 있는 변수를 채널에서 분리 가능합니다. **[다음]** **[이전]**버튼은 동일 보드 가운데서 다른 채널을 선택할 경우에 사용합니다. 다이어로그 박스의 타이틀에는 항상 선택되어 있는 채널번호가 표시되어 있습니다.

A.11.3 직접 표현 변수

I/O 변수가 접속되어 있지 않는 I/O 채널은 개방채널 이라 합니다. ISaGRAF **직접표현 변수** 는 이 개방채널을 프로그램 소스코드 중에서 직접 다룰 수 있습니다. 직접표현변수의 식별자로서 반드시 **[%]**문자로 시작되는 변수명이 됩니다.

다음에서 싱글 I/O 보드에 대한 직접표현변수 이름 할당 하는 방법에 대해서 나타냅니다. 여기서 **"s"**는 보드 슬롯번호를 나타냅니다. **"c"**는 I/O 채널번호를 나타냅니다.

%IXs.c..... 이진형 입력보드의 개방채널
%IDs.c..... 정수형 입력보드의 개방채널
%ISs.c..... 메시지형 입력보드의 개방채널
%QXs.c..... 이진형 출력보드의 개방채널
%QDs.c..... 정수형 출력보드의 개방채널
%QSs.c..... 메시지형 출력보드의 개방채널

다음에 복수 I/O 보드에서 성립되는 I/O 장치 기기에 대해 직접표현변수의 이름 할당 하는 방법에 대해서 나타냅니다. 여기서 **"s"** 는 보드의 슬롯번호를 나타냅니다. **"b"**는 I/O 장치기기 가운데 싱글보드의 index 번호를 나타냅니다. **"c"**는 I/O 채널번호를 나타냅니다

%IXs.b.c..... 이진형 입력보드의 개방채널
%IDs.b.c..... 정수형 입력보드의 개방채널
%ISs.b.c..... 메시지형 입력보드의 개방채널
%QXs.b.c..... 이진형 출력보드의 개방채널
%QDs.b.c..... 정수형 출력보드의 개방채널
%QSs.b.c..... 메시지형 출력보드의 개방채널

다음은 예제입니다:

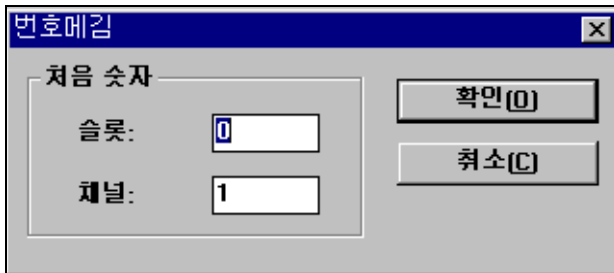
%QX1.6 보드슬롯 번호 1, I/O 채널번호 6 의 이진형 출력카드

%ID2.1.7 보드슬롯 번호 2 의 I/O 장치 기기로 보드번호가 1, I/O 채널번호 7 의
정수형 입력보드

[*] 직접표현변수는 실수형 데이터 타입을 가질 수 없습니다.

A.11.4 숫자 메깅

메뉴의 **[옵션] - [숫자메깅]**으로 번호 기입법을 설정합니다. 아래의 다이아로그 박스로 슬롯 개시번호와 각 보드마다 채널 개시번호의 설정이 가능합니다:



Default 로 슬롯번호에는[0], 채널번호에는[1]이 설정되어 있습니다.

주의: 직접표현변수에 사용되고 있는 심벌에 영향을 주어, 직접표현변수를 사용하고 있는 기존의 프로그램으로 컴파일러 에러를 일으킬 경우가 있기 때문에 번호변경에는 대단한 주의가 필요합니다.

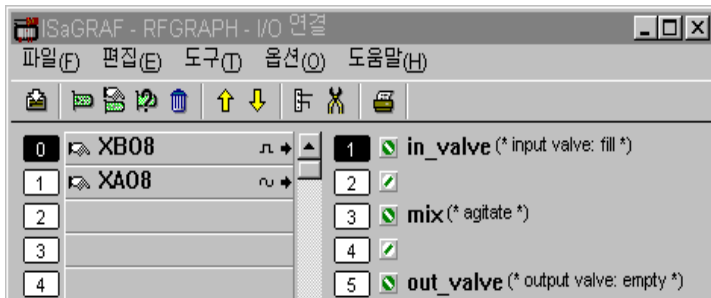
A.11.5 채널마다 보호설정

ISaGRAF workbench에서는 계층화 암호에 의한 보호 기능을 제공하고 있습니다. I/O 접속에 관해서는 암호에 따라 전역에 보호를 할 수 있습니다. 더구나 ISaGRAF에서는 I/O 보드의 채널 단위에도 보호를 할 수 있습니다.

채널 단위로 보호를 가설할 경우 프로젝트관리 편집기 [프로젝트] - [암호설정] 으로 이미 암호가 설정되어 있는 것을 전제로 합니다. 또 글로벌 I/O 보호 보다 채널 단위가 높은 레벨의 보호가 됩니다.

I/O 채널단위의 보호(protection)를 가설하고 있는 경우는 I/O 연결 편집기로 그 채널명 앞에 작은 아이콘이 표시됩니다.

I/O 채널단위의 Protection 을 가설하고 있는 경우는 I/O 연결 편집기로 그 채널이름 앞에 작은 아이콘이 표시됩니다



「편집」 - 「채널설정보호」, 「채널보호해제」로 개별 채널에 프로텍션의 설정과 삭제가 가능합니다. 어느쪽 명령어의 경우라도 채널이 있는 레벨의 프로텍션을 만들기 위해 그것에 맞는 적절한 암호를 입력합니다. 따라서 개별 프로텍션이 설정되어 있는 채널 접속을 변경할 때마다 적절한 보호레벨의 암호를 입력해야 합니다

주의: 채널에 보호(Protection)가 설정 되어 있고, 대응하는 암호가 삭제된 경우는 그보다 높은 레벨의 암호가 정의되어 있지 않으면 그 채널은 더 높은 레벨의 암호가 새롭게 정의되지 않는 한 변경할 수 없게 됩니다

A.12 변환 테이블 만들기

사용자는 수치변환 테이블을 작성할 수 있습니다. 변환테이블은 아나로그를 변환할 점의 집합입니다. 이 테이블은 정수범용置□ 입력 및 출력에 지정 가능합니다. 수치변환 테이블은 프로그램관리의 「도구」 메뉴에서 호출 가능합니다. 전기적인 신호 (센서에서 정보) 와 논리적인 신호 (어플리케이션 으로 사용할 값) 사이의 변환을 정의합니다. 변환치는 단조증가와 단조감소 중에 있어야 될 필요가 있습니다.

변환테이블은 변수목록 창의 [도구] - [변환테이블] 메뉴로 열리는 다이아로그 박스로 편집 가능합니다.

정의종료 변환테이블은 정/실수형의 입출력 변수에 대한 필터로서 사용 가능합니다. 변수에 변환테이블의 속성을 기입 할 때는 변수모음 편집기의 다이아로그를 사용합니다

A.12.1 메인 명령어

변환테이블 dialog box에서는 변환테이블 리스트와 기존의 변환테이블을 편집하거나 새로운 테이블 작성, 테이블명을 변경하기 위한 버튼이 있습니다. 변환테이블 다이아로그 박스를 종료하고, 편집내용을 저장하기위해서는 「확인」 버튼을 눌러 주세요



새 테이블 만들기

새파일 에 따라 새로운 테이블을 작성할 수 있습니다. 하나의 프로젝트에 최대 127 개의 변환테이블 작성이 가능합니다. 작성된 변환 테이블 중에서 실제로 정수.실수형의 입출력 변수에 attach 된 테이블만이 어플리케이션 코드 중에 편성됩니다. 변환 테이블 이름 작성법 에는 아래의 규칙이 있습니다.

- 최대 16 반각문자
- 최초의 문자는 영문자(a - z)
- 2 번째 이후는 영문자, 숫자 중에
- 대문자 소문자 구별없음



테이블 내용 수정

「편집」 명령어에 따라 변환테이블 리스트 중에 선택된 테이블 내용을 수정할 수 있습니다. 테이블명을 더블클릭하면 작동 가능합니다. 「편집」 명령어는 신규

테이블이 만들어진 때는 자동적으로 작동됩니다. 여기서는 변환테이블에 적어도 2 점(point) 이상의 변환포인트를 입력합니다

A.12.2 테이블 포인트 입력

「**편집**」 명령어에 따라 열려진 다이얼로그 박스 변환 테이블 포인트를 입력합니다. 좌측 box 가운데의 포인트리스트는 이미 입력종료 포인트를, 우측 아래 box 에는 정의중인 테이블을 절선 그래프로 표시합니다. 포인트는 수치 입력용 박스에 기입하면 정의 됩니다. 수치입력에는 룰이 있는데 이 절 후반에 설명합니다. 좌측 포인트리스트는 이미 입력종료 포인트를 나타내지만, 왼쪽 열은 전기신호(외부)수치를, 오른쪽 열은 어플리케이션(내부) 수치를 나타냅니다. 입력종료 포인트의 수정과 삭제의 경우는 포인트를 선택할 필요가 있습니다. 오른쪽 아래 박스에는 현재 편집 중인 테이블의 절선그래프가 표시되고 있습니다. 축에는 단위도 좌표도 없습니다. 단, 정의한 테이블이 이 그래프에 따라 바르게 입력되어 있는지를 간단히 체크할 수 있습니다.

새 포인트 정의

새로운 포인트를 입력할 경우에는 포인트 리스트의 마지막에 ("... ...")을 선택합니다. 여기서 외부 데이터와 내부 데이터의 대응을 입력합니다. 숫자는 유동 소수점으로 입력됩니다. **최소한 2점의 포인트** 입력이 필요합니다. 외-내부데이터 입력 후에[**저장**]버튼을 선택해서 테이블에 포인트를 추가합니다. **최대 3 2 포인트**가 변환 테이블에 입력 가능합니다.

포인트 수정

입력종료 포인트를 수정할 경우는 우선 변경할 포인트리스트에서 선택합니다. 그래서, 새로운 내부?외부 데이터를 입력해서[**저장**] 포인트를 선택합니다

포인트 삭제

삭제하고 싶은 포인트리스트에서 선택해서[**삭제**]버튼을 선택합니다. 단, 삭제한 결과 **최소한 2 점**의 포인트가 남아 있을 필요가 있습니다.

A.12.3 규칙과 제한사항

변환테이블을 설정할 때의 규칙을 다음에서 나타냅니다. 끝까지 변환테이블은 정수-실수형 입력변수 변환에 사용됩니다.

- 하나의 외부(전기신호)데이터에 대해 2 점 이상의 내부 데이터의 정의는 불가능합니다.
- 테이블에서 만들어진 그래프는 **단조증가** 또는 **단조감소** 중에 있을 필요가 있습니다.
- 하나의 내부 데이터에 대해 2 점 이상의 외부(전기신호)데이터를 정의할 수 없습니다.

하나의 프로젝트에 대해 변환테이블에는 다음의 제한이 있습니다.

- 최대 변환테이블 수는 하나의 프로젝트에 **127 개**입니다.
- 하나의 변환테이블 정의에는 최대 **32 점** 포인트 입력이 가능합니다.

A.13 코드 생성기 사용법

코드 생성기는 각종 ISaGRAF Workbench 창 「만들기」 또는 「검증」 명령어를 선택하면 어플리케이션 생성 창이 자동으로 열립니다. 코드가 생성된 후 이 창은 자동으로 닫히지 않습니다. 그 사이에 사용자는 코드만들기 메뉴에 액세스할 수 있습니다.

A.13.1 명령어

☐ **만들기(TIC)**

파일 - 만들기 명령어에 따라 어플리케이션 코드(중간코드, TIC 코드)를 생성합니다. 프로젝트 코드 생성 전에는 프로그램과 모음의 문법을 체크(검증) 합니다. 다음에 수치변환 테이블 처리후, I/O 접속의 처리, 또한 라이브러리 와 매칭을 합니다. 도중에 에러가 발생한 경우는, 에러를 수정하지 않는 한 다음 스텝으로 나아가지 않습니다. 이미 검증이 에러 없이 행해지고 있고, 그 후 프로그램 수정과 모음수정이 되지않는 경우에는 검증 스텝은 생략됩니다. 변수설정 및 프로그램 copy lent 체크는 매회 됩니다. **만들기** 명령어 실행중에 처리를 중단하고 싶은 경우는 ESC 키를 사용합니다. 주의 : 프로그램 local 변수가 변경된 경우는 그 프로그램이 검증 됩니다. 전역 변수가 변경된 경우는 모든 프로그램이 검증 됩니다.

☐ **프로그램 문법 검사**

파일 - 만들기 명령어로 프로젝트의 모든 모음설정내용 문법이 체크됩니다.

「파일」 - 「검증」 명령어로 프로젝트 내의 모든 프로그램이 검증 됩니다. 예를 들면, 마지막 수정시에 변경을 가하지 않는 경우에도 모든 프로그램이 검증 됩니다. 이 검증은 에러가 발생해도 도중에 중지 되지 않습니다. 모든 프로그램으로 검증하지 않는 에러리스트를 작성할 수 있습니다. 단, 검증을 중단할 경우에는 E S C 키를 사용합니다.

☐ **시뮬레이션**

[파일] - [Touch] 명령어로 다음의 **만들기** 실행시에 모든 프로그램이 검증 되도록 유사한 모든 프로그램의 수정을 더하게 됩니다. **파일 - 열기** 명령어로 마지막에 검증되어 있던 프로그램을 엽니다.

에러 발생시에는 에러 메시지를 더블클릭하면, 에러 프로그램을 열어, 에러 장소까지 커서를 이동시킵니다

A.13.2 컴파일러 선택

「옵션」 - 「컴파일러 선택」 명령어로 어플리케이션 코드생성시에 타겟 코드를 최적화 하기 위한 파라미터설정이 가능합니다. 생성된 타겟 코드 타입의 선택, 최적화 파라미터설정이 있습니다.

☐ **타겟 선택**

다이하로그 박스 상부 리스트는 타겟코드 종류를 나타냅니다.

"V"마크는 선택된 타겟코드를 표시합니다.(영문 WindowNT 해당, 한글 WindowsNT 에선 않 나타남)

ISaGRAF 코드 generate 는 한번 컴파일러로 최대 3 종류의 코드생성이 가능합니다.

선택 - 선택해제 버튼으로 생성하고 싶은 타겟코드를 선택합니다. 다음에서 표준으로 선택가능한 타겟코드를 나타냅니다.

SIMULATE:.....이 코드는 workbench 상에서 ISaGRAF 시뮬레이션 전용코드입니다. 시뮬레이션 전에 선택된 시뮬레이션용 코드를 생성해 둘 필요가 있습니다.

ISA86M:.....이 코드는 Intel 베이스 CPU 프로세스 상에서 실행되는 Target 용 TIC(Target Independent Code)코드입니다. 프로세스 타입은 바이트 나열만을 고려하고 있습니다.

ISA68M:.....이 코드는 Motorola 베이스 CPU 프로세스 상에서 실행되는 Target 용 TIC(Target Independent Code)코드입니다. 프로세스 타입은 바이트 나열만을 고려하고 있습니다.

SCC:.....이 코드를 선택하면 ISaGRAF 컴파일러는 프로젝트의 프로그램마다 구조화(소스코드를 생성하고) appli.c, appli.h 과 개별 프로그램의 xxx.c, xxx.h 파일값, Target 의 라이브러리와 컴파일러/링 하면, 컴파일러모드인 ISaGRAF Target 실행 모듈의

생성이 가능합니다. 구조화 C 와는 보기 쉬운 형식으로 기술된 C 소스입니다. [IL 언어에는 사용할 수 없습니다]

CC86M:이 코드를 선택하면 ISaGRAF 컴파일러는 프로젝트로 한 개의 C 소스코드를 생성합니다. (appli.c 와 appli.h 파일). 이 compile 파일을 Target 라이브러리와 compile/링하면, compile 모드인 ISaGRAF Target 실행 모듈의 생성이 가능합니다. 이 기능은 구조화 C 소스코드 생성기능이 지원되지 않았던 ISaGRAF V3.23 보다 전의 버전과의 호환성 때문에 남아 있습니다. CC86M 은 SCC 와 다르게, 전체면 쓰기의 C 소스 코드가 됩니다.

단, compile 하기 위해서 필요한 Target 라이브러리는 I/O 개발 kit : 옵션이 포함되어 있습니다.

주의 : I L 언어 프로그램에 관해서는 **SCC**로 C 소스가 생성되지 않기 때문에 **CC86M**을 사용해 주세요.



SFC 연산

[**embedded S F C engine 사용**]인 체크 박스를 체크 하면 ISaGRAF Target 에 S F C 엔진을 짜서넣는 것을 전제로 어플리케이션 코드를 생성합니다. 보통때는 이 체크박스를 **ON**해 둡니다. Target 실행시 퍼포먼스가 높기 때문입니다. 그러나 ISaGRAF Target 으로 S F C 엔진을 내장하지 않는 것이 있습니다. 이 경우에는 이 체크박스를 **OFF**함에 따라 S F C 프로그램 기술이 모두 low level 언어에 번역됩니다. 이러한 코드는 customized ISaGRAF Target 에서 필요로 하는 경우가 있습니다..



최적화

타겟 코드의 최적화를 위해 ISaGRAF 코드생성기를 다루는 파라미터는 **컴파일러 선택** 명령어로 선택 가능합니다. 기본에서는 compile 에서는 시간의 단축화를 위해모든 최적화 옵션은 풀어 놓고 있습니다.

◆ [**2optimizer passes 실행**] 코드 최적화를 2 회 행합니다. 단, 통상 2 회째 최적화는 1 회째만큼 중요한 의미를 가지지 않습니다.

- ◆ **[evaluate constant expressions" 정수표현 평가** 은 정수표현이 컴파일러에 따라 처리 됩니다. 예를 들면, 2 + 3 은 5 로 Target 에서는 바꿔 놓습니다. 만약 옵션이 set 되어 있지 않는 경우는 런타임 중에 2 + 3 을 계산합니다.
- ◆ **[미 사용 라벨 압축]** 프로그램 중의 미 사용 점프라벨, 라벨을 무시합니다.
- ◆ **[변수복사 최적화]** 중간결과 등을 store 할 temporary 적인 변수를 최적화합니다.**[표현의 최적화]** 옵션과 함께 set 됩니다. 예를 들면, 프로그램중에 2 회 이상 사용된 Expression 결과 등을 재 이용합니다.
- ◆ **[미 사용 코드의 압축]** 무의미한 코드를 없앱니다. 예를 들면, S T 프로그램 중에서 "**var := 1; var := X;**" 가 있어 재생되는 코드는 "**var := X;**"만 됩니다.
- ◆ **[산술연산의 최적화]** 산술연산을 최적화합니다. 예를 들면, 연산식 "**A + 0**" 는 "**A**" 로 바꿔놓게 됩니다. 같은 형식으로**[논리치 연산의 최적화]** 옵션은 논리치연산을 최적화합니다. 예를 들면, "**A & A**" 는 "**A**" 로 바꿔놓게 됩니다.
- ◆ **[Binary Decision Diagram 작성]** 논리식 (**AND**, **OR** , **XOR** , **NOT** 연산자의 편성)을 조건달기 점프로 바꿔 놓습니다. 단, 점프 전의 sequence 처리가 원래 처리보다 짧을 경우에만 적용됩니다.

다음 테이블에 최적화 옵션, 각각의 옵션에 대한 퍼포먼스 효과와 요구되는 compile time 을 정리합니다.

	속도향상	compile time
2 화 최적화를 실행	XXXX.....	(*)
정수표현 최적화	XXXXXXXX.....	XXXX
미 사용라벨 압축	XXXX.....	XXXXXXXX
변수 copy 최적화	XXXX.....	XXXXXXXX
표현의 최적화	XXXX.....	XXXXXXXX
미 사용 코드 압축	XXXX.....	XXXXXXXX
산술연산 최적화	XXXXXXXX.....	XXXX
논리치 조작 최적화	XXXXXXXX.....	XXXX
Binary Decision Diagrams 작성	XXXXXXXXXXXX	XXXXXXXXXXXX

(*)compile 시간은 2 배가 걸립니다.

A.13.3 C 소스 코드 생성

ISaGRAF Workbench 는 어플리케이션 코드를 C 소스 코드로 해서 생성할 수 있습니다. 이 경우 SFC 차트, Dataase 를 포함한 프로그램 모두가 C 소스 코드에 포함됩니다. 생성된 코드타입에는 아래의 2 가지가 있습니다.

CC86M : C 소스 코드 V3.04)비구조화 C 소스 코드를 생성합니다. 이 형식은 Target 이 ISaGRAF V3.23 이전의 버전이 경우에 사용합니다.

SCC : 구조화 C 소스 코드 :구조화 C 소스 코드를 생성합니다. Target 시스템이 V3.23 이후의 경우, 이 형식을 선택해 주세요.

[*] 프로그램에 IL 언어를 사용하고 있는 경우 SCC 에서는 C 소스 코드생성이 불가능합니다. CC86M으로 코드를 생성해 주세요.

CC86M 의 경우, 다음 2 가지 파일은 편집중인 프로젝트 서브디렉토리로 만들어 집니다.

APPLI.c공통 어플리케이션 소스 코드파일

APPLI.h공통 헤드파일

구조화 C 소스의 경우 **APPLI.c** 와 **APPLI.h** 파일에 추가해서 각 프로그램에 하나씩 확장자[.c] 소스파일과 [.h] 헤드파일이 만들어 집니다.

이들 파일은 ISaGRAF Target 용 실행모듈을 생성하기 위해 ISaGRAF Target 라이브러리와 함께 compile/링크 될 필요가 있습니다. 그래서 별도 Target 마다 I/O 개발 도구 kit (옵션)이 필요하게 됩니다. 상세한 것은 I/O 개발 kit 가이드를 참조 바랍니다.

표준 ISaGRAF Target 은, 중간 코드를 소프트웨어 처리(interpreter)해서 실행하기 위해 interpreter 형으로 하는데 이 C 소스 코드에서 컴파일러되는 경우는 compile 형이라 합니다.

주의 : 어플리케이션의 C compile 판에서는 몇 개의 디버깅 기능을 사용할 수 없게 됩니다. 예를 들면, 어플리케이션의 다운로드, 온라인 수정, 브레이크포인트 설정 ... 등 입니다.

A.13.4 정보 보기

「**편집**」 메뉴 명령어로 코드 생성중, 또는 검증 중에 작성된 텍스트파일을 열 수 있습니다. 코드생성 창은 코드생성과 검증 중의 메시지를 표시합니다. 화면표시 메시지는 뒤에 모니터 할수 있도록 하드 창에 보관됩니다.

⇒ **편집 명령어**

편집 - clear 명령어로 코드생성 창에 표시중인 텍스트를 clear 합니다. 각 코드생성, 검증 전에는 창은 자동적으로 clear 됩니다.

편집 - copy 명령어로 표시되어 있는 텍스트를 클립보드에 copy 합니다. 이 내용은 다른 텍스트 편집기 등에서 사용 가능합니다.

⇒ **컴파일러출력 메시지 모니터**

편집 - 메시지 실행 명령어로 마지막의 어플리케이션 코드생성 , 검증 명령어로 표시된 에러 메시지를 포함한 모든 메시지를 재표시합니다.

그 밖의 「**편집**」 메뉴 명령어로 코드생성과 검증 중에 생성된 파일을 모니터할 수 있습니다. 통상의 ISaGRAF 사용에 있어서는 쓰여지지 않습니다.

A.13.5 리소스 정의

「**옵션**」 - 「**리소스**」 명령어로 리소스를 정의할 수 있습니다. 여기서 말하는 리소스라는 것은 파일과 리스트 등의 형식의 사용자정의 데이터 (네트워크 구성, 하드웨어 설정등) 로 Target 코드와 merge 해서 다운로드 됩니다. 이 데이터는 ISaGRAF 커서를 직접조작 하지 않고, Target 측에 설치 된 다른 프로그램을 참조하기 위한 것입니다.

⇒ **리소스 정의 파일**

리소스 정의는 ISaGRAF 프로젝트 파일과 함께[**리소스 정의 파일**]로서 정의됩니다. 이 파일은 ASCII 텍스트 파일로 ISaGRAF 리소스 컴파일러로 처리됩니다. 리소스 컴파일러는 어플리케이션 코드 생성시에 자동적으로 작동합니다. 파일 내의 기술은 S T 언어 문법에 따라 적힙니다. 예를 들면 명령어는(* *)로, 문자열은 ' ' 로 둘러싸입니다. 상세한 문법은 해당사항을 참조 바랍니다.



언어 참조

다음에서는 리소스 파일 중에 사용되는 키워드와 그 기술방법을 나타냅니다.

ULONGDATA

의미: 정수치 리스트. Target 코드에 **unsigned 32 bit integers** 로 추가됩니다. 정수는 리소스 정의 파일로 정의된 순번으로 나열됩니다. 정수는 comma 로 구분됩니다. 리소스명은 최대 15 문자입니다.

문법: **ULONGDATA** '<resource_name>'
BEGIN
 ...target_selection...
 ...list of values...
END

예: ULongData 'MYDATA'
 Begin
 ...
 0, -1, 100_000, (* decimal *)
 16#A0B1, 2#1011_0101 (* hexadecimal, binary *)
 End

VARLIST

의미: 변수 주소를 나타내는 리스트입니다. 변수는 리소스 정의 파일 내의 변수명으로 식별됩니다. 변수 주소는 target 코드에서는 unsigned 16 bit integers 로 표현됩니다. 주소는 리소스 정의 파일 내에서 지정된 정의순이 됩니다. 변수는 comma 로 구분되고, 리소스 명칭은 최대 15 문자입니다.

문법: **VARLIST** '<resource_name>'
BEGIN
 ...target_selection...
 ...list of variable names...
END

예: VarList 'LIST'
 Begin
 ...
 Var100, My 파라미터, Command, Alarm
 End

BINARYFILE

의미: binary 파일 리소스 MS - DOS 파일형식으로 보관됩니다. 패스명으로 성립되어 있습니다. End of line 캐릭터는 리소스 컴파일러로 변환되지 않습니다. 리소스명은 최대 15 문자입니다.

문법: **BINARYFILE** '<resource_name>'
BEGIN
 ...target selection...
 FROM '<source_pathname>'
 TO '<destination_pathname>'
END

예: BinaryFile 'MYFILE'
 Begin
 ...
 From 'c:\사용자\config.bin'
 To '/dd/사용자/appl/config.dat'
 End

TEXTFILE

의미: 텍스트 파일 리소스. 어스키파일로서 보관됩니다. 패스명으로 성립되어 있습니다. End of line 캐릭터는 target 시스템 맞춤 방법으로 리소스컴파일러로 변환됩니다. 리소스 명칭은 최대꺼갓문자입니다.

문법: **TEXTFILE** '<resource_name>'
BEGIN
 ...target selection...
 FROM '<source_pathname>'
 TO '<destination_pathname>'
END

예: TextFile 'MYFILE'
 Begin
 ...
 From 'c:\사용자\config.bin'
 To '/dd/사용자/appl/config.dat'
 End

TARGET

의미: 지정할 리소스를 포함한 Target 코드의 명칭. **"Target"**스테이트먼트는 복수 Target 을 선택가능 하게 할 동일 리소스블록 내에 몇회라도

사용됩니다. "**AnyTarget**" 스테이트먼트가 사용되고 있는 곳에서는 다루지 않습니다.

문법: **TARGET** '<target_name>'

예: BinaryFile 'MYFILE'
 Begin
 Target 'ISA86M'
 Target 'ISA68M'
 ...
 End

ANYTARGET

의미: 지정 리소스 코드생성시에 모든 Target 코드에 머지 되는 것을 의미합니다. 「어플리케이션 코드생성」 명령어로 복수 Target zem 를 생성할 수 있습니다. 모든 Target 코드가 대상이 됩니다. "**Target**" 스테이트먼트가 지정되어 있는 경우는 다루지 않습니다.

Syntax: **ANYTARGET**

Example: ULongData 'MYDATA'
 Begin
 AnyTarget
 ...
 End

FROM

의미: 소스 파일명을 패스명의 부쳐서 지정합니다. ISaGRAF Workbench 가 설치 되어 있는 PC 내부의 binary 파일 또는 텍스트 파일의 파일명을 가리킵니다. 패스명의 기술시에는 MS-DOS 시스템 지정에 따를 필요가 있습니다.

문법: **FROM** '<target pathname>'

Example: BinaryFile 'MYFILE'
 Begin
 ...
 From 'c:\사용자\config.dat'
 To 'dd\사용자\appl\config.dat'
 End

TO

의미: destination(Target 시스템)에 있어 binary 파일 또는 텍스트 파일명을 가리킵니다. 파일명의 기술은 Target 시스템 지정에 따라 필요가 있습니다.

문법: **TO** '<target pathname>'

Example: TextFile 'MYFILE'

Begin

...

From 'c:\사용자\config.dat'

To '/dd/사용자/appl/config.dat'

End



예제

다음에서는 하나의 리소스 정의 파일의 예를 나타냅니다.

(*리소스 정의 파일 *)

ULongData 'DATA1' (* 데이터 리스트 *)

Begin

Target 'ISA86M' (* 지정된 Target 코드에 한정 *)

1, 0, 16#1A2B3C4D, +1, -1 (* 수치 *)

End

VarList 'VLIST1' (* 변수 리스트 *)

Begin

Target 'ISA86M' (* 이 Target 코드에 한정 *)

Valve1, StateX, Command, Alarm1 (*변수명 *)

End

BinaryFile 'FILE1' (*binary 파일 리소스 *)

Begin

AnyTarget (* 모든 Target 코드를 대상 *)

From 'c:\사용자\updatef.bin' (* 소스 파일명 *)

To 'updatef.cfg' (*Target 에서의 파일명 *)

End

```

TextFile 'FILE2'                (*텍스트 파일 리소스*)
Begin
  Target 'ISA68M'
  From 'c:\nw\nwbd.txt'          (*소스 파일명 *)
  To '/nw/dat/nwbd'              (*Target 에서의 파일명*)
End

```

리소스 compiling

"시작 " 버튼을 선택하면 리소스 compile 이 개시됩니다. 출력 메시지와 에러 메시지는 이 창에 표시됩니다. "Exit" 버튼을 선택하면 리소스 compile 가 빠집니다. 이 경우는 리소스는 Target 코드에 첨가되지 않습니다.

실행

리소스 수, 행수, 파일수는 ISaGRAF 에서 제한은 없습니다. 리소스는 Target 코드의 마지막에 리소스 디렉토리 부 치기가 첨가됩니다. 아래에서는 리소스 디렉토리 형식을 C 언어 문법으로 나타냅니다.

```

RESOURCE:
{
  long nbres;                      /* 정의되어 있는 리소스의 수 */
  {
    char name[16];                 /* 리소스명 */
    long type;                    /* 리소스 데이터 타입 */
    long size;                    /* 데이터 블록장 */
    void *data;
    char *path_offset;             /* 문자열로의 포인트 */
  } /*nb of records */
}

```

위의 "type" 파일에는 아래 가운데서 지정됩니다.

- ☐ 1 = binary 파일
- ☐ 2 = 텍스트 파일
- ☐ 3 = 데이터 리스트:ulong data (이 경우 path_offset 필드는 사용되지 않습니다.)
- ☐ 4 = 변수 리스트:varlist (이 경우 path_offset 필드는 사용되지 않습니다.)

텍스트 파일에 관해서 end of line 캐릭터는 Target 시스템을 합쳐서 리소스 컴파일러가 번역합니다. 모든 포인트는 32 비트 offset 됩니다. 리소스명과 패스명은 NULL로 종료합니다. 패스명과 data는 상기의 리소스 디렉토리 이후에 계속됩니다.

A.14 크로스 래퍼런스

ISaGRAF Workbench 에는 프로젝트 프로그램에 설정되어 있는 변수를 통괄해서 사용자에게 제공할 cross reference data 가 있습니다. cross reference 로 프로젝트에 설정된 모든 변수를 리스트 out 하고, 프로그램 소스코드에서 변수가 사용되어 지는 장소 특정지을 수 있습니다. cross reference 를 사용하면 하나의 변수를 전체적으로 보는데 상당히 도움이 됩니다. cross reference 는 I/O 변수와 I/O 보드의 나뉜 부치기와 프로젝트의 보존, 도큐먼트 작성에도 도움을 줍니다. 또한, cross reference 는 프로젝트 모음으로서도 사용됩니다. 미 사용 변수에서도 쉽게 발견할 수 있습니다.

리스트 왼쪽에는 프로젝트로 설정된 오브젝트(프로그램, 변수 및 정의된 단어)및 프로젝트로 참조되는 라이브러리 element 를 나타냅니다. 오른쪽 리스트는 왼쪽 리스트에서 선택된 오브젝트가 프로그램으로 사용되고 있는 장소(위치)를 나타냅니다.

사용장소의 기술에는, 프로그램명과 프로차트, SFC 스텝번호, Transition 또는 테스트 번호, 텍스트 언어에 대해서는 행번호, LD, FBD 언어에 대해서는 좌표등의 정보를 포함되어 있습니다. QuickLD 의 경우는 rung 번호가 표시 되어 있습니다. 만약, 변수가 출력(코일)되어 사용되고 있다면 rung 번호 앞에]*]의 문자가 첨가됩니다.

「옵션」 메뉴의 「미 사용변수 보기」 옵션을 설정하면 어플리케이션 중에 사용되고 있지 않는 변수라도 표시 가능합니다.

타입 선택

ISaGRAF 내에서 사용되고 있는 오브젝트가 많기 때문에 도구 바의 콤보박스는 창에 리스트 되어 있는 오브젝트 타입을 선택하기 위해 사용됩니다. 이 오브젝트 선택에 따라 지정된 오브젝트만을 볼 수 있습니다.

크로스 래퍼런스 재계산

「파일」 - 「다시 계산하기」 명령어에 따라 다른 ISaGRAF 편집 창에서 수정된 내용의 최신정보가 crossreference 에 반영됩니다. 이 명령어를 선택하면 오브젝트 타입선택은 [전부] 로 되돌아 갑니다.



크로스 래퍼런스 가져오기

「도구」 - 「보내기」 명령어로 크로스래퍼런스의 완전한 텍스트 리스트가 ASCII 텍스트 파일에 쓰여집니다. 이 파일은 Windows 메모장과 워드프로세스 등의 다른 어플리케이션으로 이 파일을 열 수 있습니다.



변수목록 에러

「편집」 - 「변수목록 에러」 명령어로 다이아로그 박스 중에 프로젝트 모음이 load 된때, 검출되는 에러 리스트를 표시합니다.



변수통계

「도구」 - 「변수통계」 명령어에 따라 프로젝트로 설정되어 있는 변수에 관해서, 변수 타입과 속성에 따라서 변수의 수를 행렬(매트릭스)로 표시합니다. 이 명령어에 따라 프로젝트로 설정되는 I/O 변수, 내부 변수의 총수를 파악할 수 있습니다. 이 기능을 사용하면, I/O 점수가 한정된 ISaGRAF workbench 가 사용될 때 compile 가능한지 판단해서 사용할 수 있습니다.



객체목록에서 찾기

「편집」 - 「찾기」 명령어로 오브젝트 리스트 가운데서 오브젝트를 검색할 수 있습니다. 리스트에 실제로 기재되지 않은 것에 관해서는, 오브젝트를 발견할 수 없기 때문에 오브젝트를 검색하기 전에 도구 바로 「모두」를 선택해 두길 바랍니다.



열기

오른쪽 리스트에는 오픈되어 있는 프로젝트 리소스 파일과 I/O 접속으로 포함되는 선택된 오브젝트를 포함하고 있습니다. 「편집」 - 「프로그램 열기」 명령어에 따라 선택되어 있는 오브젝트가 포함되어 있는 프로그램을 직접 열수 있습니다. 또한, 오른쪽 리스트 오브젝트를 더블클릭하면 프로그램을 열수 있습니다.

A.15 그래픽 디버거 사용법

ISaGRAF 에는 그래픽 또는 심벌링 디버거의 기능이 있습니다. 프로그램관리 「디버거」 명령어를 선택하면 타겟측에 다운로드된 어플리케이션이 제어됩니다. 이 기능에서는 디버거는 Serial(default)을 통해서 타겟과 통신합니다.

「시뮬레이션」 명령어는 디버거와 동시에 타겟 시스템을 시뮬레이션 합니다. 이 시뮬레이션으로 사용자는 타겟 I/O 가 아직 완성되지 않아도 어플리케이션을 테스트할 수 있습니다. 디버거 창에서 어플리케이션 조작이 가능합니다.

디버거가 작동되고, 타겟측의 어플리케이션이 workbench 측의 어플리케이션과 동일한 경우는, 자동적으로 **프로그램관리 창이 디버거모드로 열립니다.**(타겟과 workbench 측에서 어플리케이션이 일치할 경우는 프로그램관리 창은 열리지 않습니다) 또한, 프로그램관리 창 메뉴에서 열려진 다른 창 (그래픽 편집기, 텍스트편집기, 모음편집기, 변수리스트, I/O 접속 등) 는 모든 디버거모드로 열립니다. 즉, 여기서는 프로그램 변경, 편집 명령어는 무효합니다. 표시된 프로그램 콤포넌트 (스텝, Transition, 변수....) 는 상태표시와 함께 표시됩니다. 콤포넌트를 더블클릭하면 타겟 상태와 변수치만을 일시적으로 변경할 수 있습니다.

디버거 **시뮬레이션 모드**로 실행되고 있을 때, workbench 와 타겟간의 통신은 되고 있지 않습니다. 디버거는 I/O 시뮬레이터 창과 통신하고 있습니다. 타겟 자체가 존재하고 있지 않기 때문에 디버거메뉴 「다운로드」, 「어플리케이션-시작」, 「어플리케이션-정지」 명령어 등은 사용할 수 없습니다.

A.15.1 디버거 창

런타임 에러는 디버거창에 표시되지만, **옵션 - 에러 표시** 명령어에 따라 에러 메시지 표시를 표시/비표시 가능합니다.

디버거창 메뉴 (도구바) 의 아래 상태 표시란 에는 타겟 어플리케이션 상태와 사이클 타임정보가 표시됩니다. 다음에서는 타겟의 상태를 표시합니다.:

Logging:.....디버거가 타겟 시스템과의 통신을 설정 합니다.

연결종료:.....디버거가 타겟 시스템과 통신 불가능 상태. 연결 케이블 또는 통신 상태를 확인

- 어플리케이션없음:**통신은 정상이지만, 타겟에는 실행중의 ISaGRAF 어플리케이션이 없습니다. 어플리케이션을 **다운로드, 또는 갱신**해 주십시오..
- 어플리케이션동작:**통신은 정상으로, 타겟에 어플리케이션이 있고, 어플리케이션과 디버거가 타겟 어플리케이션과 통신하려고 합니다. workbench 와 타겟측의 어플리케이션이 같으면 프로그램 관리 창이 열립니다..
- RUN:**타겟이 **realtime 모드**로 실행중 입니다.
- STOP:**타겟이 **[Cycle to Cycle]모드**로 실행중 입니다.
- BreakPoint:**어플리케이션이 Break point 에 와 있기 때문에 타겟이 **사이클모드**로 실행중 입니다.
- 치명적 에러:**타겟 어플리케이션이 치명적인 의미불명 에러에 의해 실행 불가능합니다

사이클 타임에 관한 정보를 아래에서 나타냅니다:

- 허용치:**프로그램된 사이클 타임
- 현재치:**마지막 사이클 타임 측정치.
- 최대치:**어플리케이션이 RUN 에서의 사이클 타임 최대치.
- Overflow:**사이클 타임의 현재치가 허용치를 넘은 사이클 수.

모든 단위 ms 표시입니다. 사이클 타임 표시는 타겟 시스템에 따라 표시 정도가 바뀝니다. 상세한 것은 타겟사용자가이드를 참조 바랍니다. 디버거 시뮬레이션 모드로 사용되고 있을때 타임은 표시되지 않습니다

A.15.2 어플리케이션 제어

「파일」, 「컨트롤」 메뉴 명령어에 따라 타겟상에서 ISaGRAF 어플리케이션을 제어 할 수 있습니다..

Note: 다음의 몇 개의 명령어는 시뮬레이션 모드에서는 사용할 수 없습니다.
시뮬레이션 모드에서 어플리케이션은 workbench 내부에서 만들어지기 때문입니다.



타겟.어플리케이션 정지

「파일」 - 「정지」 명령어에 따라 현재 실행중인 타겟상의 어플리케이션을 정지 할 수 있습니다.



타겟 어플리케이션 동작

「파일」 - 「시작」 명령어에 따라 타겟상의 어플리케이션을 start 시킬 수 있습니다. 어플리케이션이 다운로드 된 때는 자동적으로 어플리케이션은 시작 합니다. 따라서 「시작」 명령어는 「정지」 명령어 후에 실행됩니다

Note : 새로운 어플리케이션이 타겟으로 다운로드 되기 전에는 타겟상의 어플리케이션은 stop 시킬 필요가 있습니다



어플리케이션 다운로드

「파일」 - 「다운로드」 명령어에 따라 workbench 측에서 생성된 어플리케이션 코드를 타겟측으로 다운로드할 필요가 있습니다. 다운로드 해야 할 코드 타입 (실행 TIC 코드, 어플리케이션 심벌 테이블) 을 선택해 주십시오



version 보기

「파일」 - 「Get Version NO」 명령어에 따라 workbench 위에 열려 있는 어플리케이션과 타겟상에 존재하고 있는 어플리케이션 version 을 확인할 수 있습니다. 다음의 항목이 표시됩니다.:

VERSION:.....이 번호는 코드 생성기가 count 하고 있는 과거에서의 코드 생성 횟수입니다..

DATE:어플리케이션 코드가 생성된 날짜와 시간을 표시합니다.

CRC:심벌 테이블에서 계산된 CheckSum 을 표시합니다. 이 숫자는 코드 생성기로 계산된 값으로 변수모음 내용으로 정해집니다. (모음테이블을 변경하면 C R C치는 변해 버립니다.)

Note: 「버전보기」 명령어는 시뮬레이션 모드로는 사용할 수 없습니다. 단, real 모드에서는 타겟과 접속하고 있지 않는 경우 역시 사용할 수 없습니다.



On-line 수정

「파일」 - 「갱신 (Update Application)」 명령어에 따라 실행중인 타겟 어플리케이션에 대해서 온라인 편집이 가능하게 됩니다. 뒤에 절로 상세하게 설명합니다. [*] 시뮬레이션 모드에서는 사용할 수 없습니다.

**Real Time 모드**

「컨트롤」 - 「real time」 명령어에 따라 통상 모드로 사이클 실행은 지정된 사이클 타임 때마다 trigger 가 걸려 작동됩니다.

**Cycle to Cycle 모드**

「컨트롤」 - 「Cycle to Cycle」 명령어에 따라 타겟 어플리케이션을 사이클 실행 모드합니다. 이 모드에서는 「컨트롤」 - 「한 Cycle 실행」 명령어에 따라, 1 사이클만 실행하고 정지합니다.

**한 cycle 실행**

타겟 어플리케이션 사이클 모드 시 「컨트롤」 - 「Cycle To Cycle」 명령어에 따라 1 사이클만 실행하고 정지합니다. 또한, 어플리케이션 시작할 때 실시 모드는 「어플리케이션 런타임 옵션」 내의 파라미터로서 set 되고 있습니다

**Cycle Timing**

「컨트롤」 - 「사이클 타임의 변경」 명령어에 따라 프로그램화된 사이클 타임을 타겟 어플리케이션이 실시 중으로 변경 가능합니다. 이 변경치는 **허용치**로서 디버거 콘트롤 바에 표시됩니다. 사이클 모드로 새로 바꿀 경우는 사이클 타임 변경 전에 set 해 둘 필요가 있습니다. 사이클 타임은 정수로 입력하지만, 단위는 ms 입니다.

**모든 breakpoints 삭제**

「컨트롤」 - 「모든 break point 삭제」 명령어에 따라 어플리케이션 전체에 걸쳐서 설정되어 있는 break point 를 모두 삭제 합니다. break point 는 이 명령어를 사용하지 않으면 디버거 창을 닫아도 자동으로 clear 되지 않습니다.

**I/O 변수 잠금 풀기**

「컨트롤」 - 「모든 I/O 변수를 잠금풀기」 명령어에 따라 디버그중에 어플리케이션 내에서 잠금되어 있는 I/O 변수의 모든 점을 un 잠금합니다. I/O 이 잠금되어 있을 때는 디버거명령어와 어플리케이션에 따라 I/O 변수 상태는 변경 가능하지만,

실 I/O 상태는 변화 하지 않습니다. 잠금된 I/O 변수는 이 명령어를 사용하지 않으면, 디버거를 닫아도 자동으로 un 잠금되지 않습니다.

A.15.3 옵션

「**옵션**」 메뉴에 따라 디버거 창에 표시되는 정보를 컨트롤할 수 있습니다.



통신 파라미터

명령어에 따라 통신 파라미터를 설정할 수 있습니다. 단, **통신 timeout** 파라미터만 여기서 설정 가능합니다. **통신 timeout** 란 workbench 에서의 요구에 대해, 타겟이 통신을 되돌리기 시작하기까지 타겟 시스템에 주어진 시간을 나타냅니다. **사이클 refresh 간격**은 디버거중의 열린 창에서 데이터 읽어 넣기를 위해 디버거 에서 나오는 읽어 넣기 요구시간 간격을 나타냅니다.

표시되어 있는 수치 모두는 ms 입니다. 시뮬레이션 모드 시엔 통신 파라미터는 설정할 수 없습니다



옵션 보기

[**옵션**] - [**사이클 타임 표시**] 명령어에 따라 디버거창에 표시되는 사이클 타임 표시를 유효/무효로 합니다.]사이클 타임 표시]가 set 되어 있을 때는 사이클 타임 (허용치, 현재치, 최대치, over load)이 표시됩니다. 설정 되어 있지 않을 때는 표시되지 않고, 타겟과의 통신 작업량은 줄어듭니다.

[**옵션**] - [**에러 표시**] 옵션이 설정 되어 있을 때는 RunTime 에러가 디버거창에 표시됩니다. set 되어 있지 않을 때는 에러 리스트는 닫혀 있고, 타겟과의 통신 작업량은 줄어듭니다. [**옵션**] - [**에러 clear**] 명령어에 따라 현재 디버거 창에 표시되어 있는 에러 표시가 삭제 됩니다.

「**옵션**」 - 「**창 최소화**」 명령어에 따라 디버거창의 사이즈를 최소화 해서, 필요한 조작이 가능한 panel 표시로 바뀝니다.

A.15.4 "적어넣기(Write)" 명령어

ISaGRAF symbolic 디버거 에 따라 어플리케이션 내의 **변수치**와 **상태**를 변경할 수 있습니다. 디버거창이 열려 있는 상태에서 프로그램 편집 창 등으로 명칭과 element 부분을 **다블클릭**하면 변경 가능한 상태가 됩니다

☐ 변수조건

다블클릭으로 변수 상태의 변경이 가능한 창에는 아래의 것이 있습니다.

- ☐ 변수목록 편집기
- ☐ 변수 리스트
- ☐ LD 또는 FBD 프로그램
- ☐ I/O 접속 편집기

다음의 명령어가 디버그 DialogBox 로 사용됩니다.

- ☐ 새로운 값 [적어넣기]
- ☐ 변수 [잠금] (I/O 변수에 한합니다)
- ☐ 변수 [잠금풀기] (I/O 변수에 한합니다)
- ☐ 타임변수 시작 또는 정지 (자동 Refresh 모드설정)

FALSE,TRUE 형 값으로 FALSE TRUE 치를 표현하는데 심벌 ic 변수가 사용됩니다. 심벌 변수와는 변수모음에서 FALSE,TRUE 형 값의 변수를 위해 정의된 문자열입니다. 아나로그 변수에 대해서[적어넣기] 를 할 경우는 변수모음 정의에 합쳐진 정수 또는 실수로 입력할 필요가 있습니다. 메시지 변수에 대해서[적어넣기]할 경우는 메시지에 주어진 최대 문자 수를 넘을 수 없습니다

☐ SFC 오브젝트

디버그시에 SFC 프로그램 처리를 모니터 하려면 프로그램 관리 창의 [파일] 메뉴 명령어를 사용합니다. 다음에서는 사용할 수 있는 명령어를 나타냅니다.

- 시작:**..... 선택된 SFC 프로그램 각각의 이니셜 스텝에 t 확인 en 을 set 하면 선택된 프로그램을 실행모드로 합니다.
- 정지(Kill)** 존재하고 있는 모든 t 확인 en 을 없애는 것으로 선택된 프로그램을 정지 시킵니다.
- 동결(Freeze)**..... 존재하고 있는 모든 t 확인 en 을 선택된 프로그램에서 없애지만 그 장소를 기억 시켜 둡니다. 재작동 명령어로 조작됩니다.
- 재작동(Restart)** 동결 되어 있는 프로그램에 대해서 한번 없앤 t 확인 en 을 원래 장소로 되돌립니다. 동결 명령어로 조작됩니다.

Child 프로그램으로서는 각각 "GSTART", "GKILL", "GFREEZE" 더욱이 "GRST" 스테이트먼트에 프로그램 중에서는 대응합니다.

SFC 스텝에서도 SFC-Child 중의 그래픽 심벌을 더블클릭하면 제어 상태를 컨트롤할 수 있습니다. 다음의 명령어가 다이아로그 박스에서 나타납니다.

- ☐ 스텝이 **활성시(상태)**에 break piont 설정
- ☐ 스텝이 **비활성상태**시에 break piont 설정
- ☐ 스텝에 설정되어 있는 break piont 삭제

Note: 활성 상태시 break piont 와 비활성시 break piont 는 동일 스텝에서는 설정할 수 없습니다.

SFC **transition** 에서도 SFC child 중의 그래픽심벌을 더블클릭하면 제어 상태를 컨트롤할 수 있습니다. 다음의 명령어가 다이아로그 박스에서 나타납니다:

- ☐ transition 통과시에 **break piont** 설정
- ☐ **설정된 break piont** 소거
- ☐ **수동으로 transition 통과** (token 이동 또는 추가)

무조건 통과:

transition 다음 스텝에 t 확인 en 을 이동(추가) 합니다.

이때앞 스텝에 있는 확인은 제거되지 않습니다.

조건부 통과: transition 다음 스텝에 확인 이동합니다. 이때 앞 스텝에 있는 확인은 자동적으로 제거 됩니다.

A.15.5 On line 수정

ISaGRAF 에션 어플리케이션 프로그램을 정지하지 않고, 온라인으로 프로그램을 수정하는 기능이 있습니다.이것을 **[온라인 수정]**이라 부릅니다. 이 기능은 세심한 주의가 필요합니다.

왜냐면, ISaGRAF 는 사용자변경에 따른 모든 문제점을 검출할 수 없을 수도 있기 때문입니다.

온라인 수정할 때 디버그모드에서 열려져 있는 프로그램 관리 창을 일단 닫고 나서 다시 열어, 필요한 프로그램을 수정하고, 다시 「어플리케이션 코드 생성」 명령어로 코드를 생성합니다. 또한 생성 된 코드를 다운로드해 두고 디버그창 어플리케이션

변경 명령어로 새로 다운로드된 어플리케이션으로 실행을 바꿉니다. 단, 변수 모음이 변경된 경우는 CRC 코드가 일치하지 않기 때문에 온라인 수정은 무효가 됩니다. 이 경우는 일단 어플리케이션이 정지한 후에 새 어플리케이션을 다운로드 start 필요가 있습니다.

☐ 코드 시퀀스

온라인 수정에 있어 적용범위는 **코드 sequence 의 변경**에 한합니다. 이 코드 sequence 는[사이클 최초 : Begin]또는[사이클 마지막 : End]섹션에 적혀 있는 ST, IL, LD, FBD 프로그램 또는 SFC 레벨 2에 기술되어 있는 프로그램을 가리킵니다. 온라인 수정에서는 어플리케이션 실행을 정지시키는 일 없이, 복수 코드 sequence 의 변경을 의미하지만, **SFC 에 있어서는 t 확인 en 컨트롤이 대단히 critical 하기 때문에 SFC 구조의 변경, 즉 스텝과 transition 의 추가, 삭제는 불가능합니다.**

☐ 내부 변수 변경

변수치에 관한 것은 어플리케이션에 대단히 중요한 위치를 차지하지만, 디버그에서 변수치의 변경은 항상 가능합니다. 하지만, **온라인 수정에 있어서는 변수를 새로 추가, 삭제할 수 없습니다.** 단, 미 사용변수(내부 또는 I/O)로 설정해 두고(모음에 미리 변수를 정의해 두고) 처음의 프로그램에서는 사용하지 않아도 후에 사용할 일이 있으면, 온라인 수정으로 가능합니다.

- 변수목록 등록 된 변수

이것들은 온라인 수정에서는 추가, 삭제, 이름 변경 등은 할 수 없습니다. 온라인 수정을 위해서는 지금 현재는 사용하지 않을 지 모르지만, 몇 개의 예비 변수를 미리 등록 해 두는 것을 생각해 볼 수 있습니다. 이러한 예비 변수는 예기치 않은 어플리케이션 변경 시에도, 어플리케이션 CRCCheckSm 이 바뀌지 않기 때문에 온라인 수정이 가능해 집니다.

- function Blocks

C 언어와 ICE 언어로 기술된 *FunctionBLOCKS* 은 ISaGRAF 타겟 데이터베이스 내에 확보된 데이터에 관련되어 있습니다. 따라서, *FunctionBLOCKS* instance 추가 또는 삭제되면 온라인 수정은 불가능합니다. 그래서 블록을 삽입하면 자동적으로 *FunctionBLOCKS* instance 가 자동 생성됩니다. QuickLD 와 FBD 편집기에서의

프로그램 보다도 *FunctionBLOCKS* instance 를 명시적으로 변수목록에 등록해서, ST 언어로 프로그램을 기술하는 쪽이 좋을 것입니다. 물론 변수목록 편집기에서의 *FunctionBLOCKS ?instance* 의 편집은 온라인 수정이 불가능 할 때 원인이 됩니다.

- SFC 스텝

각각의 SFC 스텝은 ISaGRAF 타겟 데이터베이스 내에 확보된 그 스텝의 다이내믹한 속성을 데이터에 관련되어 있습니다. SFC 스텝의 추가와 삭제와 같은 편집은 어플리케이션 데이터베이스 변경에 연결되고, 온라인 수정은 불가능합니다.

- 코드 generate 가 생성할 내부변수

ISaGRAF 코드 generate 는 복잡한 프로그램을 코드화 하기 위해 temporary 변수를 생성합니다. 때로는, 프로그램 표현 방법 변경이 이 temporary 변수 변화에 연결되는 경우가 있고, 그 결과, 온라인 수정은 불가능하게 됩니다. 이러한 상태를 막기 위해 아래에 나타난 것처럼 설정을 **ISA.INI** 파일로 강제적으로 각 프로그램마다 temporary 변수를 배당할 수 있습니다:

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

ISA.INI 파일에 이렇게 설정된 때에는 코드생성기는 어플리케이션 코드생성 결과와 지정 이상의 temporary 변수가 필요했던 경우에 경고메시지를 출력합니다.

☞ **입력/ 출력 변수의 변경**

ISaGRAF 에서의 I/O 시스템 변경은 하드웨어를 고려해서 넣은 I/O 개발 kit 에 따라 변경 가능합니다. 온라인 수정에서는 I/O 보드 기술을 변경과 I/O 보드 채널 배정을 변경 (추가, 삭제) 할 수 없습니다.

☞ **온라인 수정 조작 순서**

온라인 수정의 통상 변경 방법을 다음에 설명합니다.

- 타겟에서 어플리케이션이 실행 중에 한번 디버그창을 닫습니다.
- 프로그램 편집기로 어플리케이션 소스 코드를 변경합니다.
- [만들기]을 행합니다.

- 어플리케이션 코드를 **[만들기]** 명령어로 다운로드 합니다. **[다운로드]** 명령어는 사용 안합니다.
- **[갱신 실행]**명령어에 따라 오래된 어플리케이션에서 새 어플리케이션으로의 갱신을 사이클 타임간에 합니다.

그렇게 함으로서, 타겟 내부에서는 각 사이클에 있어서도 완전한 신뢰성이 있는 어플리케이션이 실행됩니다. 온라인 수정 빈도에 관한 제한은 없습니다.

온라인 수정은 결국 **정지 - 다운로드 - 시작** 매뉴얼로 하는 것에는 본질적으로 변함없지만, 다른점은 변수상태를 잃어버리지 않고 또, 어플리케이션 바꾸기 시간이 대단히 짧다

어플리케이션이 바뀌진 (변경된) 사이는 모든 변수 (내부,I/O) 상태는 값을 보지 하고 있고, SFC 에 있어서는 어떠한 액션도 하지 않고 SFC t 확인 en 는 움직이지 않습니다.

☞ 메모리 사양

온라인 수정이 가능한 조건으로 타겟 내부에 수정된 어플리케이션이 다운로드 되는 충분히 비어 있는 메모리가 필요합니다. 구, 신 두개 버전의 어플리케이션이 갱신 때는 동시에 존재합니다.

☞ On-Line 수정에 따른 제한

전에도 서술했지만, 온라인 수정에서는 Code Sequence 변경이 허용되지만, 변수 정의, 함수파라미터 변경,I/O 접속 변경 등은 불가능합니다.

수정된 어플리케이션이 다운로드 될 때에 ISaGRAF 는 위험한 변경을 미연에 체크할 목적으로 에러 검출을 합니다. 에러가 검출된 경우는 메시지를 표시하고 어플리케이션 경신은 중지됩니다.

체크 내용에는 심벌 테이블의 CheckSum Check 가 있습니다. 여기에서 변수 정의, 프로그램명, 끝끝 element 명의 수정이 있었는지 없었는지 체크하고 있습니다.

SFC 에 있어서 변경시에 스텝이 활성 상태인 경우 nomal(N) 액션은 놓치게 됩니다. 또한, 새로운 스텝 pulse 값스텝 올리기값 액션은 실행되지 않습니다. 새로운 스텝 P1(스텝 내리기)액션은 실행됩니다. 또한, 갱신 때는 transition 활성상태로 있으면, 거기에서 조건식은 새로운 어플리케이션 코드로 다시 한번 체크됩니다.

새로 다운로드된 어플리케이션은 SFC 내부에서의 백업(하드디스크로 보관)하지 않습니다. 갓 백업 되어 있는 어플리케이션은 통상의 다운로드 명령어에 따라 다운로드 된 어플리케이션으로 제한합니다.

주의: 어플리케이션 온라인 수정 후에 「어플리케이션 stop」 명령어를 실행 또는, 타겟을 종료한 경우에는 다음 회의 어플리케이션이 start 된 때는. 온라인 수정 이전의 구 어플리케이션이 start 합니다. 왜냐하면 어플리케이션 온라인 수정에 따라 어플리케이션 코드는 하드 디스크에 보관되지 않고, 메모리 상에 전개되고 있기 때문입니다. 따라서 다음 회에서도 수정된 어플리케이션으로 실행하고 싶을 때는, 온라인 수정 후에 「다운로드」를 실행을 필요로 합니다. (상세한 것은 타겟 매뉴얼을 참조 바랍니다.)



상세한 온라인 수정 조작

어플리케이션 변경시의 조작을 일반적인 주의사항을 포함하고 다음에 설명합니다.

- ☐ 어플리케이션 수정할 경우는 대상 프로젝트를 미리 별명으로 백업해 둘 필요가 있습니다. 수정은 복사하는 것이 현명합니다.
 - ☐ 프로그램 수정 시 에는 편집기 창으로 **수정이력 갱신** 을 **set** 해 두기 바랍니다. 이후 프로그램 보존에 대응 됩니다.
 - ☐ 코드 Switch 에 변경이 있는 경우는 반드시 새로운 **만들기** 명령어로 코드를 생성할 필요가 있습니다.
 - ☐ 「디버그」명령어에 따라 타겟측 (구 어플리케이션이 실행 중) 과 workbench 간을 접속해 둡니다. 이때 workbench 측과 타겟의 어플리케이션이 다르기 때문에 메시지 박스가 열립니다.
 - ☐ 디버거**파일 - 어플리케이션 갱신** 명령어를 실행합니다.
 - ☐ 수정된 어플리케이션을 갱신용에 다운로드 합니다. 이때 다운로드 다이아로그 박스에서는**[나중에]**를 선택합니다. (이 선택에 따라 데이터 다운로드 스피드가 조금 늦어집니다.)
 - ☐ 다운로드가 완료되면, **갱신실행** 명령어에 따라 어플리케이션을 갱신합니다. 갱신에는 1~2 사이클을 요합니다.
- 잘 어플리케이션이 갱신된 경우는 새로운 어플리케이션 프로그램이 디버그모드로 다시 표시됩니다. 만약, 어플리케이션 갱신을 실패했을 경우는 구 어플리케이션이 표시된 채로 있습니다.

A.15.6 DDE 통신

ISaGRAF 디버거는 DDE(Dynamic Data Exchange)서버기능을 포함하고 있습니다. DDE 중에 Adviser 와 P 확인 e 루프를 지원하고 있습니다. 따라서, ISaGRAF 이외 DDE 를 지원하고 있는 어플리케이션에 데이터를 넘겨줄 수 있습니다.

Advise 루프에서는 ISaGRAF 변수에 변화가 생겼을 때만 client task 에 데이터를 넘깁니다. Advise 루프로 변수를 모니터링 중에는 그 변수에 대해서 Request 루프도 취급할 수 있습니다. 모든 타입의 변수를 모니터링 할 수 있습니다. DDE 에서의 파라미터를 다음에 나타냅니다.

어플리케이션명... "ISaGRAF"

토픽명.....ISaGRAF 의 프로젝트명

아이템명 변수명

만약, 변수가 프로그램의 local 변수인 경우는 다음과 같이 괄호로 프로그램명을 명기할 필요가 있습니다.

변수명(프로그램명)

DDE Advise 루프에 따라 최대 2 5 6 변수인 동시에 모니터 가능합니다. DDE 는 디버거모드, 시뮬레이션 모드 중에서도 가용 가능합니다. Refresh 시간 간격은 workbench 측 타겟 과의 시간이 됩니다.

A.16 변수추적(Spying)

디버그창의 「스파이」 - 「변수 리스트」 명령어에 따라 비연속적인 변수를 모니터할 수 있습니다.

또한, 모니터 되어 있는 변수치의 강제적인 변경도 가능합니다. 변수리스트 정의에서는 **최대값개개의 변수**를 등록할 수 있습니다. 다른 타입의 변수를 편성하는 것은 자유입니다. □@또한, 글로벌, local 변수를 편성하는 것도 가능합니다. 등록된 리스트 정의는 하드디스크에 보관 가능합니다. 등록된 리스트는 편집중인 프로젝트 전용으로 다른 프로젝트에서 다룰 수 없습니다.

변수 리스트는 어플리케이션의 부분적인 테스트로 유효합니다. 어플리케이션 소스 코드와는 별개로 특정부분의 프로세스 변화를 볼 수 있습니다. S T, I L 언어에서도 사용되고 있는 변수를 그룹분리해서 프로그램 실행을 제어 및 모니터링이 가능합니다.



H/D 에 리스트 저장

파일 - 열기, 겹쓰기보존, 다른이름으로저장 명령어에 따라 변수 리스트를 하드디스크를 열기도 하고, 하드디스크에 보관 가능합니다. 파일명(변수리스트명)붙이는 법에는 다음의 룰이 있습니다. 파일명 수의 제한은 없습니다.

- 변수 리스트명의 최대장은 반각 8 문자
- 최초의 문자는 영문자 (a - z)
- 이후 문자는 영문자, 숫자 중에
대문자, 소문자 구별 없음



변수 추가

「편집」 - 「삽입」 명령어에 따라 리스트에 다른 변수를 추가 합니다. 변수명은 프로젝트 모음편집기에서 선택합니다. 신규 변수는 선택중인 변수 앞에 삽입됩니다. 최대 3 2 개의 변수를 등록할 수 있습니다. 동일 변수는 두번 정의 할 수 없습니다. 한번에 3 2 개 이상의 변수 리스트를 표시할 경우는 두개의 리스트 창을 열게 됩니다.



선택 변수 변환

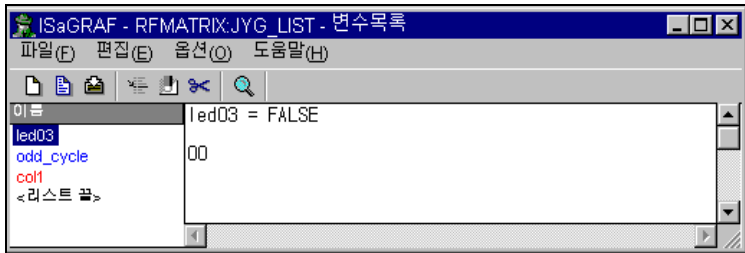
「**편집**」 - 「**편집**」 명령어에 따라 리스트 중에서 선택된 변수명을 변경할 수 있습니다.



Dump 표시

Dump/리스트 버튼, 또는 옵션 **dump** 명령어로 변수표시를 리스트 형식과 dump 표시형식의 바꾸기가 가능합니다.

dump 표시 모드시에는 하나의 변수에 대해서만 표시 됩니다. 창 상부에 그 값을 수치 또는 문자열로 표시하고, 그 아래에는 binary 형식으로 dump 한 데이터가 표시됩니다. 이 표시 모드를 사용하면, 변수 16 스파이가 가능하게 됩니다.



dump 표시는 인쇄 불가능한 캐릭터를 포함한 문자열 변수를 스파다음고 해석하기에는 상당히 편리합니다.

A.17 ST / IL 프로그램 디버깅

ST IL 프로그램 시뮬레이션, 디버깅 중에 직접 프로그램 변경을 첨가 할수 없습니다.

IL

IL 프로그램에서는 instruction 이 리스트 형식이 되고, 변수의 현재치가 같은 행에 표시됩니다. instruction 을 더블클릭하면 대응하는 변수치를 변경할 수 있습니다.

ST

ST 프로그램에서는 편집기 창에 스파이 리스트 창이 짜여집니다. 두개의 창 사이의 세퍼레이션을 마우스로 드래그하면, 표시 사이즈 변경이 가능합니다

변수 스파이 리스트에는 변수명, 현재치, 명령어가 표시된 각 파일의 폭은 마우스 드래그에 의해 변경 가능합니다.



스파이 리스트 보존

「파일」 - 「보존」 명령어에 따라 디버깅 중의 프로그램명과 동일명으로 스파이 리스트가 보존됩니다. 이 리스트는 다음번 ST, IL 프로그램이 디버깅상태가 될때 자동적으로 열립니다.

또한, 디버깅 창의 「도구」 - 「스파이 리스트」 메뉴명령어에 따라 이 프로그램명인 스파이 리스트를 표시하고 수정, 보존하는 것도 가능합니다.



스파이 리스트로 변경 추가

「편집」 - 「변수 삽입」 메뉴명령어에 따라 스파이 리스트에 변수를 삽입할 수 있습니다. 프로젝트 변수모음의 변수 선택용 다이아로그박스에서 삽입할 변수를 선택합니다. 이러한 변수를 매뉴얼 입력할 필요는 없습니다. 이 리스트에는 최대 3 2 개 까지의 변수 포함이 가능합니다. 동일 변수명이 리스트상에 표시되는 일은 없습니다.



ST 텍스트 창에서 변수가 선택 (하이라이트) 되어 있을 때는 「편집」 - 「스파이 리스트에서 추가」 또는 「스파이 리스트로 추가」 도구 바 아이콘에 따라 스파이 리스트에 변수를 삽입할 수 있습니다. 변수를 선택해서 오른쪽 마우스 클릭해서, 오픈되는 putup 메뉴에서의 선택도 가능합니다.



선택된 변수 변경

「**편집**」 - 「**변수 변경**」 메뉴명령어에 따라 스파이 리스트상으로 선택되어 있는 변수 변경이 가능합니다. 「**편집**」 - 「**변수 잘라내기**」 메뉴명령어에 따라 스파이 리스트상에서 선택되어 있는 변수 삭제가 가능합니다.

A.18 스포트라이트 사용법

ISaGRAF의 스포트라이트 도구 바는 디버그중에 모니터 하고싶은 정보를 그래픽과 리스트 스타일로 정의해서, 모니터링 가능합니다. 그래픽 항목은 ISaGRAF 프로젝트 변수에 링되어야만 합니다. 항목은 온라인 중 (디버그중) 에 신규로 정의되기도하고, 애니메이션하는 것도 가능합니다.

선택된 항목의 변수를 적어 넣을 때는 그래픽과 레이아웃상의 항목을 더블클릭할지를 선택되어 있을때에 Enter 키를 누릅니다.

A.18.1 그래픽 윤곽구축

그래픽 윤곽은 배경 그래픽스, 메타 파일상에 그래픽한 변수항목을 배치시켜 작성됩니다. 이 변수항목은 디버그중에 애니메이션 됩니다. 화면을 구성할 때는 우선 그래픽스 파일을 삽입, 그래픽 변수항목을 삽입, 변수항목과 프로젝트 내의 변수와 링합니다.



배경 픽처

빅마크 (BMP) 와 메타파일 (WMF) 윤곽상에 배치 가능합니다. 배치된 그래픽스는 이동, 사이즈 변경이 가능합니다. 이 파일들은 리스트 윤곽에서 바뀌었을때는 표시되지 않습니다. 스포트라이트는 페인트 도구가 아니기 때문에 이 파일들은 수정할 수 없습니다.

「옵션」 - 「배경색」 명령어로 윤곽배경을 지정한 단일 색으로 하는 것이 가능합니다.

주의 : 빅마크 파일은 큰 메모리를 점유하기 위해 가장 적당한 파일사이즈로 해 둘 것을 권합니다.

123

텍스트 표시

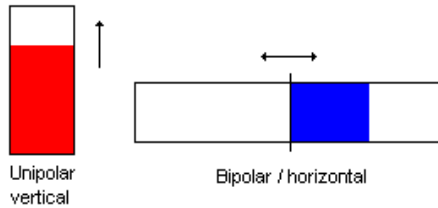
텍스트는 장방형 중에 관련있는 텍스트 항목입니다. 이 텍스트는 배경된 qust 치를 표시합니다. 이 항목에는 Boolean 형, 정수 / 실수형, 가변 장문자열형, 타임형 모두를 배정할 수 있습니다.

장방형 내의 색과 텍스트 색의 변경이 가능합니다. 더욱이 장방형 사이즈에 맞춘 문자 글꼴지정이 가능합니다.



봉 그래프, 양방향 봉 그래프

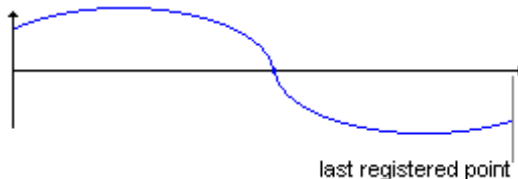
봉 그래프에서는 배정된 정수 / 실정수형 변수치가 지정된 색인 장방형의 형태로 표시됩니다. 수직, 수평방향을 설정할 수 있습니다. 또한, 통상의 봉 그래프는 상하좌우 중의 방향에서도 설정가능한 것에 대해, Bipolar 형 봉 그래프에서는 2 방향 (정방향, 음수방향) 을 가집니다. Bipolar 봉 그래프에서 스케일의 최대치는 2 방향 모두 같은 값이 됩니다.



트렌드 그래프

윤곽상에 **트렌드 그래프**를 삽입할 수 있습니다. 트렌드 그래프는 배정된 변수치의 이력을 표시합니다. 정확한 측정 도구는 아니지만, 복수 변수를 배치하면 변수간의 타이밍도 볼 수 있습니다.

트렌드 그래프는 최신의 200 점을 축척, 표시합니다. 트렌드 그래프의 사이즈를 변경해도 이 점수의 변경은 불가능합니다.



Boolean 형 아이콘

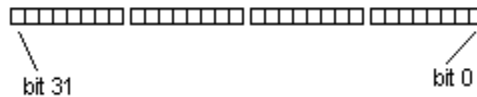
윤곽상에 **Boolean 형 아이콘**을 삽입할 수 있습니다. Boolean 형 아이콘은 Boolean 형 변수 상태를 아이콘 파일 상태에 따라 표현합니다. FALSE 시 아이콘 파일 (I C O) 과 TRUE 시의 아이콘 파일 배정하게 됩니다.

스포트라이트는 편집기 아이콘이 아니기 때문에 아이콘 파일 수정은 불가능합니다. 이 아이콘 파일들은 다른 도구로 준비해 둘 필요가 있습니다.



bit 전개

bit 전개는 정수형 변수를 bit 열 (32 bit vinally) 그래픽 표시로 나타낼 수 있습니다. L S B는 항상 우측입니다. 정수형 이외의 변수를 표시하는 것도 가능하지만, 직접 의미가 있는 표시는 되지 않습니다. bit 열이 바르게 표시되기 위해서는, 항목 표시 사이즈 좌우에 충분한 크기가 필요합니다.



선택, 이동, 사이즈 변경

윤곽상의 항목을 선택하기 위해서는 마우스로 직접 심벌을 클릭합니다. 복수 심벌을 모아서 선택할 경우는 마우스로 선택할 영역을 골라 항목이 선택된 때는 경계영역이 작게 ■ 으로 둘러 싸입니다.

선택 항목을 선택하지 않으려면, 항목 이외의 background 를 마우스 클릭합니다.

항목을 이동할 경우는 선택된 지역의 경계 부근에 마우스를 가까이 하면, 마우스 커서 형태가 이동용으로 변합니다. 여기서 마우스 드래그하면 항목을 이동시킬 수 있습니다.

심벌 사이즈를 변경할 때는 마우스를 선택영역 ■ 상으로 이동시켜 마우스 커서상태가 변화한 곳에 마우스 드래그합니다.

심벌에는 bit map 파일, 메타파일이 포함되어 있습니다.



그룹화 / 그룹 해제

윤곽상의 복수 항목을 그룹화 해서 하나의 항목으로 하는 것이 가능합니다.

「편집」 - 「그룹화」 명령어에 따라 그룹화할 수 있습니다. 그룹화에 따라 항목을 모아서 이동 가능하지만, 사이즈 변경은 할 수 없습니다.

그룹화된 항목을 되돌리기 위해서 그룹 해제합니다. 「편집」 - 「그룹 해제」 명령어에 따라 선택된 그룹을 해제할 수 있습니다.

그룹 내에 다른 그룹과 그래픽스를 포함시키는 것도 가능합니다.

그룹화된 항목에 관한 스타일 변경은 불가능합니다.

그룹화된 항목은 리스트 윤곽으로 바꾸면 하나의 항목으로 모여져서 표시됩니다.

A.18.2 리스트 윤곽



「옵션」 - 「리스트 / 그래프 윤곽」 메뉴 명령어에 따라 그래픽윤곽과 리스트윤곽모드 바꾸기가 가능합니다.

리스트윤곽모드에선 모든 항목이 리스트상에 나열됩니다. 하나의 항목 높이에 따라 미리 정해져 있습니다. bit map(?)과 메타파일 등은 이 모드에선 표시되지 않습니다. 이 모드에선도 항목 선택, 항목 스타일 설정은 가능합니다. 단, 복수 항목 선택, 그룹화 등은 불가능합니다.



「편집」 - 「리스트의 상 / 하로 이동」 메뉴명령어에 따라 선택된 항목을 리스트 내에서 상하 방향으로 이동시킬 수 있습니다.

A.18.3 스타일 설정

윤곽상의 항목인 스타일 변경, 정의가 가능합니다. 심벌을 선택해서, 「편집」 - 「항목 스타일 설정」 명령어를 선택하면 항목 스타일 설정이 가능합니다. 심벌 선택은 그래픽 윤곽, 리스트윤곽중에서도 가능합니다. 신규 항목을 삽입하면, 항목 스타일 설정 다이아로그박스가 열립니다.



스타일 설정:

항목 표시스타일 (단일 텍스트, 봉그래프, 트렌드 그래프...) 을ダイナ믹하게 변경 가능합니다. Boolean 형 아이콘이 선택된 경우, 지정할 아이콘 파일은 pass 명 붙이기로 지정할 필요가 있습니다. 「 . . . 」 버튼을 누르면 아이콘 파일을 일관해서 선택하는 것이 가능합니다.



스케일:

봉그래프와 트렌드 그래프에서의 최대치를 나타냅니다. 하이폴라(?)봉 그래프에서는 2 방향 모두 같은 최대치가 됩니다.



이름:

이름 파일이 선택되어 있을때 「 . . . 」 버튼을 누르면 프로젝트 내에 설정되어 있는 변수선택이 가능합니다.

**caption:**

caption 은 그래픽 윤곽상에서 그래픽항목 부근에 표시되는 명칭입니다. Caption 장소 (좌우상 하) 와 표시형식의 선택이 가능합니다. Caption 카스타마이즈는 리스트 윤곽모드로 바뀐 경우에는 영향을 받지 않습니다.

**변수 적어넣기:**

「변수 적어넣기」 옵션이 체크되면, 디버그중에 그래픽 오브젝트를 더블클릭하면, 이 오브젝트에 관련한 변수치의 변경이 가능하게 됩니다.

A.18.4 「파일」 메뉴 명령어

「파일」 메뉴명령어에는 윤곽도큐먼트 관리용으로 다음의 명령어가 포함되어 있습니다.



「파일」 - 「신규」 메뉴명령어로 신규 윤곽의 작성이 가능합니다. 하나의 프로젝트 내에서 작성할 수 있는 윤곽수의 제한은 없습니다. SportLight 에서는 동시에 두개 이상의 윤곽은 열수 없습니다. 단, 복수 프로젝트를 디버그중인 각 프로젝트에 대해서 윤곽을 열수 있습니다.



「파일」 - 「열기」 메뉴명령어에 따라 현재 열려 있는 윤곽을 닫고, 선택된 윤곽을 열수 있습니다. 또, 윤곽을 선택하는 다이아로그박스에선 「삭제」 버튼에 따라 등록되어 있는 윤곽의 삭제가 가능합니다. 단, 이 조작에 따라 관련되어 있는 아이콘, bit map 파일등이 삭제되는 일은 없습니다.



「파일」 - 「표서(겉쓰기) 보존」 메뉴명령어에 따라 열려있던 윤곽을 디스크에 보존 가능합니다. 만약, 윤곽에 이름 (타이틀) 이 주어져 있지 않는 경우는 이것을 입력할 필요가 있습니다. 윤곽명은 다음의 룰에 따를 필요가 있습니다.

- 이름은 최대 8 문자
- 최초의 문자는 문자 (a ~ z)
- 마지막은 문자 (a ~ z), 숫자, "_ "
- 대문자, 소문자 구별 없음

「파일」 - 「잠금」 메뉴명령어에 따라 열려 있는 윤곽으로 신규항목의 추가, 배경색의 변경, 항목 설정등을 할수 없을 경우가 있습니다. 메뉴명령어가 많으면

선택 불가능하게 됩니다. 단, 이 상태에서도 이미 윤곽상에 있는 항목으로 변수 적어넣기는 가능합니다. 틀린 윤곽을 변경시키지 않기 위한 유효한 수단입니다.

A.18.5 버전 3.2 사용자로서의 주의 사항

스포트라이트는 버전 3.0 또는 3.2 로 작성된 그래픽 데이터와 트렌드 그래프 데이터를 읽어 넣는 것이 가능합니다. 이 파일들은 「파일 오픈」 다이아로그에 작성된 버전 명령어 첨가로 리스트 업됩니다. 읽어 넣은 파일은 스포트라이트로 자유롭게 편집 가능합니다.

버전 3.2 로 작성된 그래픽 데이터는 자동적으로 **잠금**상태가 되어 오픈됩니다. 편집하기 위해서는 일단, 「파일 잠금」 명령어로 잠금을 삭제해야만 합니다.

버전 3.2 로 작성된 그래픽 데이터와 트렌드 그래프 데이터는 일단 오픈하면, 종료시에 스포트라이트데이터 형식으로 보존을 재촉하기 위해 메시지 박스가 자동적으로 오픈됩니다.

A.19 어플리케이션 갱신

ISaGRAF 는 타겟에 보관되어 있는 어플리케이션 코드를 upload 하는 기능이 있습니다. 갱신기능은 타겟 내에 보관되어 있는 압축된 어플리케이션 소스 (EZS: Embedded Zipped Source code) 를 workbench 에 집어넣어 풀면 가능합니다.

A.19.1 프로젝트 갱신

[갱신]다이하로그박스는 프로젝트관리[파일]메뉴명령어에서 열수 있습니다. workbench 상으로 선택되어 있는 프로젝트와 upload 되는 프로젝트와는 관계가 없습니다. 타겟에서 실행중인 어플리케이션을 upload 하기 위해서는 다음과 같이 할 필요가 있습니다.

- 1- 타겟과 workbench 가 바르게 접속되어 있을것
- 2- 통신 paramete 설정 (설정버튼으로 링 설정에 따라서)
- 3- **[실행]**버튼을 누른다.

압축 소스 (E Z S) 를 upload 해 풀기 위해서는 조금 시간이 걸립니다. 다이하로그박스의 메시지 표시에 따라 upload 완료를 알 수 있습니다. 그렇지 않으면 어려가 됩니다.

Upload 되는 ISaGRAF 프로젝트명은 타겟과의 통신을 통해서 읽기 시작합니다. 만약, 이미 workbench 내에 같은 프로젝트명이 있을 경우는 표시와 이름변경이 가능합니다. Upload 가 완료된 후에 프로젝트 등록 캔슬은 불가능합니다. 이것으로 upload 된 프로젝트를 열수 있습니다.

여러 가능성

프로젝트 upload 시에는 다음의 에러를 생각할 수 있습니다. 에러에 관해서는[upload]다이하로그박스에 표시됩니다.

- 타겟과 통신이 확립되지 않았다.
- 타겟이 Ver. 3.23 보다 이전 버전이다.
- 타겟에서 어플리케이션 이 실행되고 있지 않다.
- 타겟에 압축소스 (E Z S) 가 없다.

A.19.2 통신 설정

[설정]버튼으로 workbench 와 타겟간에 통신 설정할 다이아로그박스를 열 수 있습니다. 여기서의 통신 설정내용이 타겟측의 protocol 과 일치해야 할 필요가 있습니다.

A.19.3 갱신 준비

타겟측에서 workbench 측으로의 upload 가 필요할때 미리 어플리케이션 코드 생성시에, 압축소스 (EZS) 를 짜넣는 것을 설정할 필요가 있습니다.

때문에,[코드생성 옵션]다이아로그박스에서[upload]버튼을 선택합니다. 여기에 표시된 체크박스에서[압축소스 짜넣기]를 체크하면 프로그램 디버거, 어플리케이션 코드 생성을 위한 최소한의 소스 코드가 압축되어 넣어집니다. 그 이외에 짜넣어진 개별 정보도 체크박스에 따라 선택 가능합니다. 다이아로그박스에서 압축 소스 내용을 선택합니다.

주의 : 압축 소스에는 라이브러리가 포함되어 있지 않습니다. 라이브러리란 예를 들면, Function, Function 블록,I/O 보드,I/O 장치기기가 포함됩니다.

upload 기구의 이해를 위해서는 다음의 항목도 참조 바랍니다.

☞ 첨부 파일 (추가 짜넣기 파일)

압축 소스에 포함되는 최소한의 파일에 덧붙혀 다음의 파일도 짜넣기 가능합니다. 이 파일들을 선택하면 타겟측에서 요구하는 메모리량이 늘어나기 때문에 주의가 필요합니다.

프로젝트 기술

만약, 짜넣을 수 없을 경우는 프로젝트 기술은 upload 된 날짜만 나타납니다.

암호 프로텍션

upload 된 프로젝트는 암호 프로텍션되어 있지 않기 때문에 필요한 경우는, 선택해서 암호 프로텍션을 짜넣을 필요가 있습니다.

미접속 I/O 채널로서의 명령어

ISaGRAF 에는 미접속 I/O 채널에 대해서 명령어를 기술할 수 있습니다. 만약, 접속 종료 I/O 채널만 명령어가 필요한 경우는 체크하지 마세요.

프로젝트 수정이력

프로젝트의 수정이력 파일입니다.

수정이력

각각의 프로그램에 대한 수정이력과 사용자 메모 파일입니다. 이 파일은 프로젝트가 크진 타겟에 요구되는 메모리도 대용량이 될 가능성이 있습니다.

변수 리스트, 변수 trace

디버그중에 생성, 보존된 변수 리스트와 변수 trace 용 설정 파일입니다.

그래픽스, 아이콘, bit map 파일

workbench 상의 프로젝트 티폴트 내에 포함되어 있는 아이콘, bit map 파일 등의 그래픽스 파일 입니다. 이 선택에 따라 타겟측에서 대용량 메모리가 필요할 경우가 있습니다.

A.19.4 타겟에서 압축 소스 보관

압축 소스 (Embedded zipped source : 이후는 **EZS** 라고 표시합니다.) 는 리소스로서 타겟 내에 어플리케이션 코드 (중간 코드) 와 같이 보관됩니다. 따라서, 압축 소스를 선택할 경우는 다른 리소스 이름에 E Z S 를 지정할 수 없습니다. 다른 명칭의 리소스를 부가할 것의 제어와 이미 사용자가 짜넣기도 하는 리소스에 영향을 주는 일은 없습니다.

리소스에 관한 상세한 것은 workbench 가이드의 「코드 생성을 참조 바랍니다.

A.19.5 타겟에 필요한 메모리 용량

압축 소스(EZS)에는 타겟 내에 메모리 여유가 필요하게 됩니다. 최소한 E Z S 사이즈 (프로젝트만 선택한 경우) 는 어플리케이션 사이즈의 약 1 . 5 배 정도가 됩니다. 따라서, E Z S 를 포함한 전체 사이즈는 어플리케이션 코드의 약 2 . 5 배가 됩니다.

segment 된 메모리 제약을 가진 타겟 (예를 들면 D O S 에서는 6 4 K B 사이즈) 에서는 제약이 있습니다. E Z S 는 어플리케이션 코드와 동일 타겟 메시지 상에 존재할 필요가 있습니다

A.19.6 갱신 프로젝트에 관해서

upload 된 프로젝트에는 한번더 코드 생성을 위한 파일은 전부 포함되어 있습니다. 다운로드 전에 선택된 옵션에 따른 보충 파일 (프로젝트 기술, 수정이력 등) 이 포함되어 있습니다.

디버그전에는 한번 upload 된 프로젝트로 어플리케이션 코드를 생성할 필요가 있습니다.

workbench 상에서의 디버그를 작동시켜, 타겟과 버전이 (CRC 체크) 다른 것이 표시되지만, 필요하다면 생성된 코드를 타겟측에 다운로드함으로써 표시를 없애는 것도 가능합니다.

주의 : 라이브러리는 압축 소스와 함께 다운로드 되지 않습니다. 프로젝트를 upload 해서, 다시 코드 생성되기 전에 필요한 Function 과 Function 블록 라이브러리가 workbench 상에 설치되어 있는 것을 필요로 하게 됩니다.

이 upload 기능은 타겟측에서 요구하는 메모리를 가능한 적게 하기 위해 모니터 / 디버그할때 필요한 workbench 측의 파일이 압축 소스와 함께 다운로드되지 않습니다. 따라서, 프로젝트 모니터 / 디버그만을 하고 싶지 않을 경우에서도, upload 완료 후에 어플리케이션 코드를 생성할 필요가 있습니다.

A.19.7 버전 호환성

upload 기능은 ISaGRAF 버전 3.23 이후로 workbench 와 타겟과의 통신 protocol 확장에 따라 지원되었습니다.

타겟측이 버전 3.23 이전 (3.0x, 3.20) 의 경우에도, workbench 측에서 타겟측으로 압축 소스 (E Z S) 를 다운로드하는 것이 가능합니다. E Z S 는 단순한 어플리케이션 코드에 추가되기도한 리소스로 다뤄집니다. 단, 이 같은 타겟에서는 upload 기능을 위해 통신 protocol 확장이 포함되어 있지 않기 때문에 workbench 측으로의 upload 가 불가능합니다.

A.20 디버그 (진단) 도구의 사용법

「진단 도구」은 ISaGRAF 디버거서버 set 입니다. 진단 도구에 따라 이미 프로그램 내에서 사용되고 있는 변수의 조작이 가능합니다.

ISaGRAF workbench 에 포함되어 있는디버거와 다른 진단 도구에는 최종판 어플리케이션 maintenance 목적으로 사용합니다. 진단 도구 작동은 프로그램 매니저 내의 ISaGRAF 그룹 다음의 아이콘을 더블클릭하면 작동 가능합니다.



여기서는 이미 다운로드된 ISaGRAF 어플리케이션에 대해서 제한된 디버거의 작동이 가능합니다.

「확인」버튼은 선택된 프로젝트로 디버거를 오픈합니다. 「설정」버튼은 ISaGRAF workbench 와 타겟 사이의 통신 paramete 를 설정합니다. 상세한 것은 「프로그램 관리」장을 참조 바랍니다.

주의 : 이 진단 도구에서는 프로그램 다운로드, 정지, 갱신은 불가능합니다. 만약, 선택되어 열려진 프로젝트가 타겟 내에서 어플리케이션과 다를 경우는 조작이 불가능합니다.

진단 도구에서의 메뉴는 통상의 디버그 (시뮬레이션) 메뉴가 서버 set 되어 있어 아래 기능이 사용 가능합니다.

- 변수 스파이 리스트
- 스포트라이트에 따른 그래픽디버그

A.21 시뮬레이션 사용법

프로그램 관리 창 「디버그」 - 「시뮬레이션」 명령어에 따라 ISaGRAF 시뮬레이터가 디버거와 함께 작동됩니다. 시뮬레이션은 offline 디버그라 할 수 있습니다. 이 시뮬레이션에서는 ISaGRAF 타겟 시스템을 창 위에서 안전하게 시뮬레이션합니다. 단, Function BLOCKS 과 C 언어 함수 등에 관해서는 준비 set 에 관해서만 지원되고 있습니다. (사용자정의 함수 등을 시뮬레이션에 집어넣는 경우는 workbench 함수 kit : 옵션 소프트웨어에 따라 카스터마이징할 필요가 있습니다.) I/O 보드는 가상 I/O 패널로서 그래픽컬하게 창 상에 시뮬레이션됩니다. 어떤 종류의 I/O 보드 (가상 I/O 도 포함해서) 에 관해서도 시뮬레이션됩니다.

A.21.1 디버거와의 링크

시뮬레이션은 ISaGRAF 디버거간의 통신을 완전히 지원합니다. 온라인 디버그로 생각되는 조작이 가능하게 되어 있습니다. 단, 「파일」 - 「어플리케이션시작」, 「어플리케이션종료」, 「다운로드」, 「어플리케이션의 갱신」 명령어에 관해서는 무효가 됩니다.

시뮬레이션하기 위해서는 미리 시뮬레이션용의 어플리케이션 코드가 필요합니다. 그러므로 프로그램 관리 창의 「코드 생성」 - 「컴파일러 옵션」 설정으로 시뮬레이션용 코드가 선택되어 있는 확인한 후에 「어플리케이션 코드 생성」을 실행할 필요가 있습니다.

커널시뮬레이션창 (가상 I/O 패널) 와 디버그창을 닫는 것으로 디버그 세션으로 열려 있는 창은 전부 닫힙니다.

A.21.2 I/O 시뮬레이션

슬롯 번호마다 I/O 보드가 가상 I/O 패널 창에 그래픽컬하게 시뮬레이션됩니다. ISaGRAF 표준 I/O 보드 (Boolean 형, 변정/실수형, 가변 장문자열형) 를 다룰 수 있습니다. 보드 상에는 **슬롯 번호, I/O 보드명, 채널 번호**가 표시됩니다.

입력 보드는 버튼 또는 파일과 함께 표시되고, 출력 보드는 LED 또는 파일과 함께 표시됩니다. I/O 접속 편집기에서 **가상 I/O 보드**로서 설정되어 있는 경우도, 보드는 그래픽컬하게 시뮬레이션됩니다..

Boolean 형 입력 보드

Boolean 형 보드는 작은 버튼과 함께 표시됩니다. 채널 번호가 버튼 옆에 표시됩니다. 버튼을 누르면, TRUE 상태가 되고, 다시 한번 누르면 FALSE 상태가 됩니다. 오른쪽 마우스를 클릭하면, 누르고 있는 사이에만 TRUE 상태가 됩니다.

Boolean 형 출력 보드:

Boolean 형 출력 보드는 LED 표시와 함께 표시됩니다. 채널 번호가 LED 옆에 표시됩니다. 값이 TRUE 상태일때 LED가 점화됩니다.

정수형 입력 보드: 정수형 입력 보드는 입력용 수치 필드와 함께 표시됩니다. 수치 필드를 클릭해서 숫자를 입력하면 대응한 변수명에 아니로그 값을 대입할 수 있습니다. **ENTER**키의 입력은 필요 없고, 상하 버튼을 클릭하면 입력 가능합니다. 아니로그 값은 10 진수, 16 진수 중에서도 입력 가능합니다.

정수형 출력 보드:

정수형 출력 보드는 출력용 수치 필드와 함께 표시됩니다. 수치 필드에는 변수의 값이 10 진수 또는 16 진수로 표시됩니다. 이 값은 변경할 수 없습니다.

문자열형 입력 보드:

문자열 입력 보드는 텍스트 입력 필드와 함께 표시됩니다. 텍스트 입력 필드에 커서를 이동해서, 문자열을 입력하면 대응한 변수로의 문자열 입력이 가능합니다. **ENTER**키의 입력은 필요 없습니다.

문자열형 출력 보드:

문자열 출력 보드는 텍스트 출력 필드와 함께 표시됩니다. 이 문자열의 변경은 불가능합니다.

A.21.3 라이브러리 구성요소

ISaGRAF 시뮬레이터는 표준 변환함수, 함수, 표준 Function Block 잠금을 지원하고 있습니다. 다음에 지원되어 있는 오브젝트 리스트를 행합니다.

변환함수:

DCB B C D 코드 Binary 변환 (Decimal Coded Binary)

SCALE L E D meter 표시 (Volt meter led indicator)

Function:

abs	절대치 (absolute value)
acos	arc cosine (arc cosine calculation)
ArCreate	정수배열 생성 (create array of integer)
ArRead	실수배열 생성 (read array of integers)
ArWrite	정수배열 적어넣기 (write array of integers)
ascii	ascii code 에서 변환 (character => ascii code)
asin	arc sine (arc sine calculation)
atan	arc tangent (arc tangent calculation)
char	character 변환 (ascii code => character)
cos	cosine (cosine calculation)
delete	문자열의 일부 삭제 (delete sub-string)
expt	exponential (exponent calculation)
find	문자열 검색 (find sub-string)
insert	문자열 삽입 (insert sub-string)
left	문자열 좌측 추출 (extract left sub-string)
limit	값 제한 필터 (bound value between limits)
log	log (logarithm (base 10))
max	최대치 (maximum of two integers)
mid	문자열 추출 (extract middle sub-string)
min	최소치 (minimum of two integers)
mten	message 길이 (length of a message)
mod	여분 (modulo calculation)
mux4	4 입력 multiplexer (multiplexer 4 entries)
mux8	8 입력 multiplexer (multiplexer 8 entries)
odd	홀수 parity (odd parity)
rand	random 정수값 (random integer value)
replace	문자열 바꿔놓기 (replace sub-string)
right	문자열 우측 추출 (extract right sub-string)
rol	bit 좌회전 (rotate left)
ror	bit 우회전 (rotate right)
sel	selector (binary selector)
shl	bit 좌이동 (shift left)
shr	bit 우이동 (shift right)
sin	sine (sine calculation)

sqrt	평방근 (square root)
tan	tangent (tangent calculation)
trunc	잘라버림 (truncate decimal part)

⇒ **Function Block** **참금:**

average	평균치 (running average)
blink	blink (blinking signal)
cmp	비교 (full comparison)
ctd	down dounter (down dounter)
ctu	up counter (up counter)
ctud	up/ down counter (up/down counter)
derivate	미분기 (derivative regulation)
f_trig	내리기 검출 (falling edge detection)
hyster	hysteresis (hysteresis)
integral	적분기 (integral regulation)
lim_alm	lim_alm (alarm on limits)
pid	P I D 적분기 (PID regulation)
pid_cj	P I D 적분기 (PID regulation)
r_trig	오르기 검출 (rising edge detection)
rs	dominant bistable reset (reset dominant bistable)
sema	semaphore (semaphore)
sr	dominant bistable set (set dominant bistable)
stackint	정수 stack (stack of integer values)
tof	off timer (off-delay timer)
ton	on timer (on-delay timer)
tp	pulse timer (pulse timer)

사용자정의 C언어함수, Function BLOCKS 은 ISaGRAF 시뮬레이션에서는 표준으로 짜낼 수 없습니다. 사용자정의 함수 등은 타겟 시스템에 의존하고 있어, Windows에서 실행할 수 없는 경우도 있습니다.

ISaGRAF 시뮬레이터는 사용자정의 변환함수, 함수, Function BLOCKS 에 대해서 다음의 표준적인 행동을 제공합니다.

- 새로운 변환함수가 실행될때, NULL 변환됩니다. 즉, 입력신호가 완전히 변환되지 않고 출력됩니다.

- C언어 함수가 실행중 일때는 일절 처리하지 않습니다. 출력치는 항상 제로 (또는 NULL 문자열) 가 됩니다.
- CFunction BLOCKS 이 실행중 일때는 일절 처리하지 않습니다. Return prameter 는 항상 제로 (또는 NULL 문자열) 가 됩니다.

– 단, workbench 개발 도구 kit (옵션소프트) 에 따라 시뮬레이션에 사용자정의 함수 등의 표시를 짜넣을 수 있습니다.

A.21.4 옵션

「**옵션**」 메뉴에 따라 가상 I/O 패널 표시 변경이 가능합니다. 시뮬레이션중에 언제라도 명령어를 set, reset 할 수 있습니다.

- ⇒ 「**칼라 모드**」 명령어에 따라 I/O 채널 표시는 칼라 bit map 표시로 됩니다. monochro 디스플레이의 경우는 흑백을 표시하기 위해 이 옵션을 빼 주십시오.
- ⇒ 「**변수명**」 명령어에 따라 I/O 채널 오른쪽 옆에 변수명 라벨이 표시됩니다. I/O 보드 시뮬레이션판넬 사이즈를 작게 하기 위해 이 옵션을 빼 주십시오.
- ⇒ 「**16진수 값**」 명령어에 따라 입/출력 아나로그 채널이 16진수로 표시, 입력됩니다.
- ⇒ 「**항상 앞면**」 명령어에 따라 가상 I/O 패널이 항상 화면 앞면에 표시됩니다.

A.21.5 입력신호 상태의 보존 / 읽어내기

ISaGRAF 시뮬레이터를 사용해서 IO 시뮬레이션판넬 토글 버튼과 편집기 컨트롤박스를 사용한 조작에 따라 입력채널의 강제입력이 가능합니다. 이 입력채널들의 상태를 보존하기도 하고, 읽어 넣기도 하기 위해 아래의 명령어가 「**도구**」 메뉴에 있습니다.

입력패턴 load 파일에 보존된 입력채널 치(値)를 입력채널에 설정합니다.

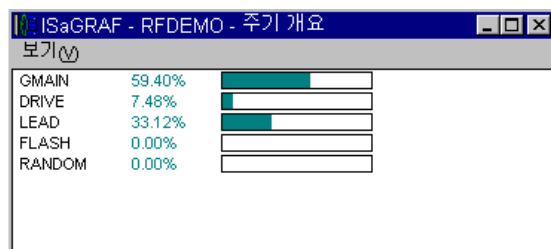
입력패턴 세이브 입력채널 상태를 파일에 보존합니다. 파일은 프로젝트 디렉토리에 보존됩니다.

메모 : 이름이 부쳐져 있는 입력채널 (변수 배정이 되어 있는 입력) 만이 보존됩니다.

A.21.6 주기개요

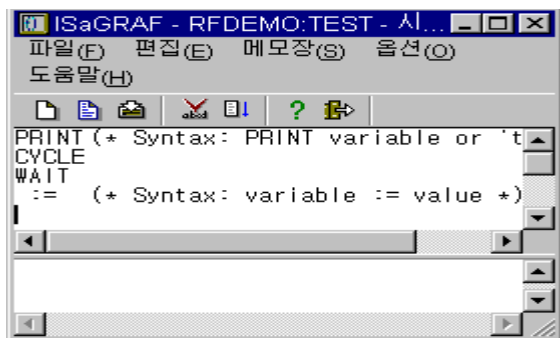
주기개요는 어플리케이션 중에서 사용되고 있는 복수 프로그램, 함수, Function BLOCKS 각각의 실행시간이 어느 정도 분포하고 있는가를 표시할 수 있는 상당히 편리한 진단기능입니다. 이 도구에서는 어플리케이션 퍼포먼스 체크 기능과 어느 부분의 코드가 가장 적당한지를 조사하려 할때도 도움이 됩니다.

주기개요의 작동은 시뮬레이터창 「도구 주기개요」 메뉴명령어로 합니다. 그래서 프로그램, 함수, Function Block 잠금 각각의 실행에 걸리는 시간을 퍼센테이지로 표시합니다.



「표시 - 평균」 옵션을 체크하면, 어플리케이션이 start 에서 또는 마지막 「표시 - reset」 명령어를 실행한 후 평균을 계산해서 표시합니다.

「표시 - 평균」 옵션이 체크되어 있지 않을 때는 바로 앞의 사이클 실행중에 관측된 정보를 표시합니다. 어플리케이션 실행 모드를 「사이클」에 설정해서 이 기능을 사용하면 어플리케이션 각각의 콘테스트에 따라 실행시간의 관측이 가능합니다.



「표시 - 복사」 명령어로 프로그래밍과 퍼센테이지를 클립보에 텍스트 형식으로 copy 가능합니다. 이 데이터를 도큐먼트와 서버레드시트에 붙일 수 있습니다.

주의: 이것은 정밀한 계측은 아닙니다. 퍼센테이지 계산은 TIC 코드 명령수 카운트에 따라 합니다. C 언어로 기술된 함수와 Function BLOCKS 처리시간은 포함되어 있지 않습니다.

함수, Function BLOCKS 에 표시되는 값은 1 사이클 중에 호출되는 모든 처리시간의 합계치가 됩니다.

또, C 소스 만들기를 사용해서 작성된 타겟의 경우는 신뢰할 수 있는 테이타를 표시할 수 없습니다

A.21.7 시뮬레이션 메모

ISaGRAF 시뮬레이션 에는 시뮬레이션 메모 를 작성해 실행할 도구가 있습니다. 시뮬레이션메모는 ST 언어와 비슷한 텍스트 언어로 기술됩니다.

윗부분의 창은 메모를 입력할 텍스트 편집기입니다. 사용방법은 다른 ISaGRFA 텍스트 편집기와 동일 합니다.

아랫부분의 창에는 시뮬레이션메모가 실행된때의 출력 메시지가 표시됩니다. 창 세퍼레이터는 자유롭게 드래그할 수 있고, 사이즈 변경도 가능합니다. 시뮬레이션메모 편집시는 이 창은 숨겨져 있을 경우가 있지만, 시뮬레이션메모가 실행되면 자동적으로 표시됩니다.

≡ 시뮬레이션메모의 편집

시뮬레이션메모 파일에 관한 조작은 「파일」 메뉴에서 합니다.

신규작성..... 신규시뮬레이션메모파일 작성

열기..... 기존의 시뮬레이션메모 파일을 엽니다.

표서 보존..... 시뮬레이션메모텍스트와 출력 창の内容을 보존합니다.

이름을 부쳐 보존... 시뮬레이션메모 텍스트와 출력 창の内容을 다른 이름으로 보존합니다.

각각의 시뮬레이션메모에 대해 두개의 파일이 프로젝트 디렉토리에 작성됩니다.

<시뮬레이션메모명>.SCC scrip 텍스트 (명령)

<시뮬레이션메모명>.SCO 출력 창 내용

여기서 <시뮬레이션메모명>은 시뮬레이션메모 이름을 가리킵니다. 이 파일들은 통상의 텍스트 파일로 일반 텍스트 편집기로 오픈 가능합니다.



시뮬레이션메모 편집중 「편집 - **심벌 삽입**」 명령어로 설정된 변수명을 커서 위치에 삽입 가능합니다.



시뮬레이션메모 실행

시뮬레이션메모는 실행하기 전에 문법 체크와 컴파일러할 필요가 있습니다. 「실행」 명령어가 선택될때 필요에 응해서 문법 체크가 자동적으로 됩니다. 아래의 「**시뮬레이션메모**」 메뉴명령어를 사용합니다.



체크 시뮬레이션메모의 문법 체크와 컴파일러



시뮬레이션메모 실행 시뮬레이션메모 실행

무제의시뮬레이션메모경우에는 문법 체크 하기 전에 이름을 부쳐 보존해야만 합니다. 이름이 부쳐진 시뮬레이션메모의 경우는 문법 체크 전에 자동적으로 파일로 보존가능합니다.

시뮬레이션메모가 실행되어 있을때는 그 내용은 편집할 수 없습니다. 시뮬레이션메모 실행이 종료한때는 메시지가 표시됩니다. 또한, 아래의 「**시뮬레이션메모**」 명령어로 실행중인 시뮬레이션메모를 중단하는것도 가능합니다.



시뮬레이션메모 중단 실행중 시뮬레이션메모 중단

시뮬레이션메모 실행은 타겟 사이클 사이에서 실행됩니다. 사이클 중에 무한 루프가 있을 경우는 시뮬레이터는 이 무한 루프를 분할해서 실행하고, 사이클 실행이 잇달아되고, 다른 ISaGRAF 어플리케이션이 잠금되지 않도록 하고 있습니다. ISaGRAF 시뮬레이션메모 interpreter 는 1 사이클 중에 같은[라벨]이 2회 카운트 되면 시뮬레이션메모실행을 중단합니다. 또한, 시뮬레이션메모 실행은 [사이클]과 [대기] 인스트럭션에 따라서도 중단됩니다.

시뮬레이션메모 기술언어

시뮬레이션메모 기술언어는 심플한 ST 언어와 비슷한 텍스트 언어입니다. 그러나, 각행 끝에 세미콘 (;) 는 요하지 않습니다. 도구 바 상의 아래 버튼을 사용해서, 사용 가능한 인스트럭션을 조사해, 커서 위치에 삽입 가능합니다.

? 인스트럭션 삽입 (키 워드와 도움말 명령어)

여러가지 인스트럭션이 있습니다. 값을 변수에 대입하는 인스트럭션

:=대입

출력 창에 메시지를 출력하는 인스트럭션

Print 텍스트와 변수치의 출력

PrintTime 현재 시각 출력

ISaGRAF 사이클과 같은 기간을 가지기 위한 인스트럭션

Cycle 1 사이클 실행

Wait.....지정 시간 기다림

시뮬레이션메모 의 흐름을 제어하기 위한 인스트럭션 :

라벨 시뮬레이션메모 중의 어디에서도 배치 가능함

Goto라벨로 무조건 점프

If goto.....라벨로 조건 점프

End..... 시뮬레이션메모 종료

시뮬레이션메모 언어에는 case 문은 없습니다. 명령어는 ST 언어와 같이 "(" "*" ")" 로 싸여 있고 ;] 세미콘을 처음에 부쳐 기술합니다.

":=" 대입

의미: 변수의 값을 강제적으로 대입합니다. 내부변수, 입력변수, 출력변수에 대해서 유효합니다.

문법: <varname> := <constant_expression>
<varname> = <constant_expression>

설명: <varname>은 변수모음에 등록된 변수명 또는]%]문자를 접두자로 사용하는 접두표현 변수입니다.

<constant_expression>는 지정한 변수에 따른 정수치 입니다. Ture / False 형 변수에 대해서는 1 / 0 를 사용합니다. 또한, 타임변수에 대해서는]T#]와]TIME#]의 접두문자는 생략 가능합니다.

메모: 시뮬레이션메모에 의한 입력변수 대한 값을 대입할 경우는 변수의 잠금은 불필요합니다.

주의: 변환 함수를 설정하고 있는 입출력 변수에 대한 값의 대입을해서는 안됩니다. 시뮬레이션메모에선 변환 함수와 테이블을 지원하고 있지 않습니다.

예: MyBooVar := 1 (* same as TRUE *)
MyIntVar := 1234
MyRealVar := 1.2345
MyMsgVar := 'Hello'
MyTmrVar := t#12s

Print

의미: 출력 창 의 문자열과 변수치의 출력. 출력 창 마지막 문자에서 바뀐행의 새로운 행에 표시됩니다.

문법: Print '<text>'
Print <varname>

설명: <text>는 싱글 quotation 으로 둘러싸인 문자열

<varname>은 변수모음에 등록된 변수명 또는 [%]문자를 접두자로 사용하는 직접 표현 변수입니다.

메모: 변수치 출력은 IEC 규격에 따른 형식에서 됩니다.

예: Print 'Hello'
Print MyBooVar

출력 : Hello
MyBoovar = TRUE

PrintTime

의미 : 출력 창에 현재 시각을 표시합니다. 출력 창 마지막 문자에서 바뀐행의 새로운 행에 표시됩니다.

문법 : **PrintTime**

메모 : 현재 시각의 출력 형식은 Windows 시스템 설정에 의존하고 있습니다.

예 : Print 'Time now is:'
PrintTime

출력 : Time now is:
15:45:22

Cycle

의미 : 다음의 ISaGRAF 사이클이 실행되기까지 시뮬레이션메모 실행을 일시 중지합니다.

문법 : **Cycle**

메모 : 시뮬레이션메모 실행은 ISaGRAF 사이클 처음에 됩니다. 만약, 시뮬레이터가 사이클 모드인 경우 이 [Cycle]명령은 사이클 실행후에 됩니다. 이후의 시뮬레이션메모는 디버거 「1 사이클 실행」 명령어가 실행된 후에 실행됩니다.

예 : (*ISaGRAF 프로그램에는 A 을 B 로 copy 하는 프로그램이 있습니다.*)
A := 0
Cycle
Print B
A := 1
Print B (* no cycle performed / B not set to 1 *)
Cycle
Print B

Output: B = 0
B = 0
B = 1

Wait

의미 : 지정한 시간이 경과하기까지 시뮬레이션메모를 일시 정지합니다.

문법 : **Wait** <delay>

설명 : <delay>는 delay 값으로 시간 정수를 나타냅니다. IEC 규격에 따른 표현 방법으로 기술합니다. "T#" 와 "TIME#" 의 접두문자는 생략 가능합니다. 이 값은 10mS 에서 1 시간 사이의 값이어야만 합니다.

메모 : "Wait" 인스트럭션 정도는 host 시스템 Windows 에 의존하고 있기 때문에 정밀한 것은 아닙니다. 또한, ± 1 사이클 실행시간,분의 오차를 고려할 필요가 있습니다.

"Wait" 인스트럭션이 실행되면 지정된 시간이 경과하기까지 ISaGRAF 사이클이 실행되고, 그후 시뮬레이션메모가 실행됩니다.

예 : PrintTime
 Wait 2s
 PrintTime

Output: 15:45:27
 15:45:29

라벨

의미 : 라벨은 시뮬레이션메모중의 어디에서나 배치 가능합니다. 라벨은]Goto[인스트럭션 점프 앞에서 사용하고 시뮬레이션메모의 흐름을 제어하는 것이 가능합니다.

문법 : <labelname>:

설명 : <labelname>은 영문자로 시작되고 영문자와 underscore 등의 16 문자 이내의 유니크한 문자열입니다. 설정시에는 coma ([:]) 를 뒤에 부쳐야만 합니다.

메모 : 라벨을 설정한 행에 다른 인스트럭션은 기술할 수 없습니다.
라벨명은 변수모음에 설정된변수명과 동일해야만 합니다.

예 : (* example of a 시뮬레이션메모 with an infinite loop *)
 loop:
 PrintTime
 Wait 1s
 Goto loop

Goto

의미 : 라벨

문법 : **Goto** <labelname>

설명 : <labelname>은 시뮬레이션메모 중에 설정된 라벨명

메모 : 역방향 점프도 가능합니다. 무한 loop 의 경우는 ISaGRAF 사이클 실행을 유지하기 위해 시뮬레이션메모실행은 loop 에 분할 되어 실행됩니다.

예 : Print 'Before Jump'
 Goto MyLabel
 Print 'Within Jump' (* instruction never performed *)
 MyLabel:
 Print 'After Jump'

Output: Before Jump
 After Jump

If Goto

의미 : 라벨로의 조건 점프. 조건에는 변수와 변수의 비교 및 변수와 정수의 비교를 사용합니다.

문법 : **If** <var1> **test** <var2> **Goto** <labelname>
If <var1> **test** <constant_expression> **Goto** <labelname>

test 에 사용가능한 것을 아래에 적으면

- = 양변치가 같은 경우에 참
- <> 양변치가 같지 않는 경우에 참.
- < 좌변이 우변보다 작을 경우에 참.
- <= 좌변이 우변 다음의 경우에 참.
- > 좌변이 우변보다 큰 경우에 참.
- >= 좌변이 우변 이상의 경우에 참.

설명 : <var1> <var2> 는변수모음에 등록된 변수명, 또는 [%]문자를 접두자로서 사용하는 직접 표현변수입니다.

<constant_expression>는 지정한 변수에 따른 정수치입니다. Boolean 형 변수에 대해서는 1 / 0 를 사용합니다. 또한, timer 변수에 대해서는 [T#]와 [TIME#] 접두문자는 생략 가능합니다.

<labelname>는 시뮬레이션메모중에 설정된 라벨명입니다.

메모 : 역방향 점프도 가능합니다. 무한 loop 의 경우는 ISaGRAF 사이클 실행을 유지하기 위해 시뮬레이션메모 실행은 loop 에 분할되어 실행됩니다.

예 : (* This 시뮬레이션메모 loops until MyVar is TRUE *)
 Loop:
 If MyVar = TRUE Goto TheEnd
 Print MyVar
 Goto Loop
 TheEnd:

End

의미 : 시뮬레이션메모 종료

문법 : End

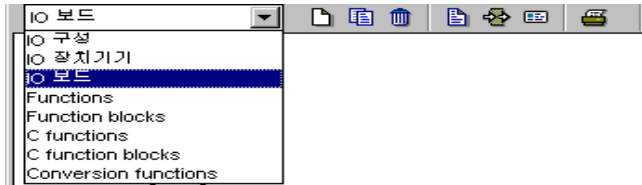
메모 : 시뮬레이션메모의 마지막 행에 [End] 인스트럭션을 기술하는 것은 필수 조건은 아닙니다.

예 : (* This 시뮬레이션메모 loops until MyVar is TRUE *)
 Loop:
 If MyVar = FALSE Goto Continue
 End
 Continue:
 Print MyVar
 Goto Loop

A.22 라이브러리 편집기 사용법

ISaGRAF 라이브러리관리 utility 는 각종 타입의 라이브러리군을 관리하고 있습니다. I/O 보드 드라이브에서 사용자정의 F B까지를 커버하고 있습니다. 라이브러리관리를 사용해서 사용자가 작성할 드라이브, Function 등의 오브젝트 파라미터등을 정의해서등록해 둘 수 있습니다.

ISaGRAF 라이브러리관리의 좌측 창은 선택된 라이브러리 **element** 리스트를 표시하고, 우측 창은 선택된 element **기술 메모**를 표시합니다. 라이브러리관리 메뉴에는 선택된 라이브러리 element 의 작성, 정의, 변경을 위한 명령어를 포함하고 있습니다.



다음의 ASCII 파일이 라이브러리관리에 따라 변경됩니다. 이 파일들에는 element 이름과 라이브러리중의 논리적인 index 관계가 표시되어 있습니다.

```

\isawin\lib\PLCNUMS ..... I/O 구성용
\isawin\lib\XIONUMS ..... I/O 장치 기기용
\isawin\lib\BRDNUMS ..... I/O 보드용
\isawin\lib\IFCNUMS ..... IEC 언어로 적힌 함수용
\isawin\lib\USPNUMS ..... C 언어 함수용
\isawin\lib\FBLNUMS ..... C Function BLOCKS 용
\isawin\lib\CNVNUMS ..... 변환 함수용
  
```

A.22.1 라이브러리 편집기 관리 : 「파일」 메뉴 명령어

「파일」 메뉴에는 오브젝트로서 라이브러리를 관리하는 명령어가 포함되어 있습니다.



신규 작성

「파일」 - 「새파일」 명령어에 따라 선택중의 라이브러리에 새로운 element 를 추가 합니다. 다음의 룰로 element 명을 입력합니다.

- 최대 8 반각 문자
- 최초 문자는 영문자(a -z)
- 이후의 문자는 영문자, 숫자 중에서
- 라이브러리 element 명에는 대소문자 구별없음

텍스트에 의한 element 에 대한 명령어 (기술메모) 입력이 가능합니다. 다음의 내용은 반드시 기술 메모에 포함할 필요가 있습니다.

- I/O 구성 정의,
- I/O 보드에 대한 파라미터,
- 함수와 Function BLOCKS 에 대한 사용자 interface

C 변환 function, CFunction, CFunction BLOCKS 이 정의되면 소스코드 template 가 자동으로 작성됩니다.



기존 element 관리

「파일」 - 「이름 변경」 명령어에 따라 element 이름의 변경이 가능합니다. 「copy」 명령어에 따라 동일 라이브러리 없이 copy element 를 작성합니다. 이미 같은 이름의 element 가 있을 경우는 표시됩니다. 「삭제」 명령어에 따라 element 가 삭제됩니다. 「이름 변경」, 「copy」, 「삭제」 명령어에 따라 element 다음의 component 가 대상이 됩니다.

- 메모장
- I/O 구성
- I/O 보드, 구성기기 파라미터

- Function, Function BLOCKS 의 interface 정의
- I E C 언어로 적힌 Function, Function BLOCKS 소스 코드
- C 변환 function, CFunction, CFunction BLOCKS 소스 코드



주의: 「이름 변경」, 「copy」 명령어에서는 element 가 C 변환 function, Cfunction, CFunction BLOCKS 기술메모, 소스 코드 내부 이름은 자동으로 갱신되지 않습니다.



주의: 「이름 변경」, 「copy」 명령어에서는 I E C 언어로 기술된 Function 출력 파라미터 (되돌리기 값) 는 자동으로 갱신되지 않습니다.



암호 보호설정

「파일」 - 「암호 설정」 명령어에 따라 라이브러리내의 선택된 element 에 대해서 각종 레벨 암호 보호할 수 있습니다. 암호는 선택되어 있는 element 에서만 유효하고, 이것은 와 다른 라이브러리에 대해서는 무효가 됩니다. 암호 protection 에 관해서는 다른 장을 참조 바랍니다.



Function, Function BLOCKS 검증

라이브러리에서 I E C 언어에 따라 Function, Function BLOCKS 이 선택되어 있는 경우 「파일」 - 「검증 (컴파일러) 」 명령어에 따라 선택된 변수 element 의 문법을 하지 않고, 대응한 오브젝트 코드를 생성합니다. 이들은 ISaGRAF 프로젝트 내에서 사용되기 전에 에러 없이 컴파일러되어 있을 필요가 있습니다. I E C 언어에 따라 Function, Function BLOCKS 이외의 라이브러리가 선택될 경우는 이 명령어는 사용할 수 없습니다.



메모장

「편집」 - 「 메모장 」 명령어에 따라 선택되어 있는 element 에 대한 기술메모 입력, 수정은 ISaGRAF 텍스트 편집기를 사용해서 가능합니다. 기술메모는 온라인 도움말로 ISaGRAF 프로젝트 프로그램 작성시에 편집기에서[정보]로 호출 가능합니다. 기술 메모에는 element 의 주된 목적, 프로그램 입출력 파라미터 interface, 제한 그밖의 필요한 정보를 포함하고 있을 필요가 있습니다.

「편집」 - 「 메모장 」 명령어에 따라 선택중 라이브러리의 모든 element 의 작성에 대해서 유효한 표준 기술 메모 형식을 작성할 수 있습니다. 신규 element 를 작성할 경우에 여기서의 형식이 기술 메모 template 로서 사용됩니다. 사용자오리지널 기술 메모 형식의 작성이 가능합니다.



라이브러리 요소 파라미터

「편집」 - 「파라미터」 명령어에 따라 라이브러리 element 의 **파라미터** 수정이 가능합니다. 라이브러리 element 의 **파라미터** 는 ISaGRAF 프로그램에서의 element 의 interface 부분을 정의하고 있습니다. 라이브러리마다 **파라미터** 가 가진 의미는 다릅니다.

I/O 보드 집합체의 구성을 정의하고 있고, I/O 채널에는 기본명이 붙어 있습니다.

보드의 물리적, 논리적인 구성을 정의하고 있습니다.

element S T 언어로 기술한 경우 인수에 해당하는 interface 부를 정의합니다.

표준 설정 종료의 interface 를 사용하기 위해 존재하지 않습니다.



라이브러리 구성요소 소스 코드

라이브러리관리 utility 에 따라 변환 function, Function, Function BLOCKS 라이브러리 element 의 소스 코드를 관리할 수 있습니다.

Function 의 소스 코드는 IEC 언어 (L D , F B D , S T , I L) 로 적혀 있습니다.

C 언어 Function, C 언어 Function BLOCKS, C 언어 변환 function 의 소스 코드는 두개의 파일로 나뉘어져 있습니다. 첫째는 소스 **header (appli.h)** 로 element 파라미터정의에서 온 interface 부분을 정의하고 있습니다. 둘째로는 본래의 **소스 코드(appli.c)**로 element 처리 실현 부분을 정의하고 있습니다.

C Function, C Function BLOCKS 소스 코드의 경우, 새로운 라이브러리 element 가 작성되면 라이브러리관리 utility 는 소스 코드의 template 파일을 자동 생성합니다. 파라미터정의에 기초한 소스 header 도 자동 생성합니다. ISaGRAF 텍스트 편집기로 소스 코드를 완성 시킬수 있습니다. C언어 소스 파일 정의에 관해서는 타겟 메뉴얼 (C 프로그램 테크닉) 를 참조 바랍니다.

라이브러리관리 utility 는 I/O 보드에 관한 소스 코드 관리를 포함하고 있지 않습니다.



라이브러리 보관함

「보관함」 명령어에 따라 라이브러리 element 의 저장 / 백업 하기 위해 ISaGRAF 의 보관함 메니저를 작동할 수 있습니다. 우선 처음에 라이브러리를 선택한 후에 보관함 명령어를 사용해 주십시오. 이때 선택된 라이브러리 리스트가 표시됩니다.

A.22.2 I/O 구성

I/O 구성 에 따라 **설정 종료의 I/O 구성을 편집**, 작성할 수 있습니다. 여기서는 다음의 내용을 정의합니다.

- I/O 보드 집어넣기
- I/O 보드 파라미터용 기본 값
- I/O 채널용 기본명

신규 ISaGRAF 프로젝트가 라이브러리 I/O 구성을 선택한 경우, I/O 보드 채널에 사용되어 있는 I/O 변수명의 모음등록, 변수 I/O 접속이 자동적으로 됩니다.



I/O 구성의 정의

I/O 구성의 정의는 ISaGRAF I/O 접속 편집기 (상세한 것은 I/O 접속 편집기 사용방법 참조) 로 합니다. I/O 구성에 새로운 I/O 보드를 삽입할 때에 보드의 모든 채널에 기본 변수명이 할당됩니다. (프로젝트 작성시 I/O 접속 편집기에 따라 각 채널에서 수동으로 변수를 할당할 필요가 있습니다.) 표준 기본 변수명 형식은 다음과 같이 됩니다.

<방향><타입><슬롯 번호>_<채널 번호>

< 방향 >

최초 문자는 I/O 채널 방향을 나타냅니다.

"I" 입력채널

"O" 출력채널

< 타입 >

2 번째 문자는 I/O 채널 타입을 나타냅니다.

"X" Boolean

"D" 정수형

"M" 문자열형

예로서 다음에 표준적인 I/O 변수명을 나타냅니다.

IX0_7 Boolean 입력으로 슬롯번호 0, 채널번호 7

QD2_4정수형 출력으로 슬롯번호 2, 채널번호 4

「편집」 - 「보드 파라미터 / 채널 set」 명령어 (또는 채널부분 더블클릭) 에 따라 I/O 채널에 할당되어 있는 기본 변수명을 변경할 수 있습니다.

A.22.3 I/O 장치 기기

I/O 장치 기기와는 다른 타입과 방향을 가진 I/O 보드가 편성된 I/O 디바이스를 나타냅니다. 각각의 보드는 동일 타입 (Boolean, 정수형, 문자열형) 으로 동시에 동일 방향 (입력, 출력) 을 가집니다. I/O 장치 기기는 복수 I/O 보드를 가지지만, rack 내의 하나의 슬롯밖에 점유하지 않는 장치입니다.

I/O 장치 기기 정의

I/O 장치 기기 정의를 하기 위해서는 우선 복수 I/O 보드 리스트를 정의할 필요가 있습니다. 각각의 I/O 보드에 대해서 I/O 보드 파라미터다이아로그박스를 사용해서 상세한 파라미터를 입력할 필요가 있습니다.

I/O 보드 리스트를 작성할 경우는 I/O 기기 다이아로그박스로[추가기록]버튼에 따라 I/O 보드 리스트 마지막에 새로운 보드를 추가할 수 있습니다.[삽입] 버튼에 따라 선택되어 있는 보드 앞에 새로운 보드를 삽입합니다.[삭제]는 선택되어 있는 보드 삭제를,[이름 변경]은 선택되어 있는 보드의 이름을 변경합니다.[파라미터]버튼에 따라 선택된 보드의 파라미터를 변경합니다. 파라미터설정은 다음의 항목을 참조하십시오.I/O 장치 기기는 **최대 16** 인 I/O 보드를 가질 수 있습니다. I/O 장치 기기내의 각각의 보드 이름은 최대 **반각 8 문자** 입니다.

A.22.4 I/O 보드

I/O 보드 라이브러리는 어플리케이션 프로그램변수와 타겟 하드웨어(I/O) 사이의 인터페이스(INTERFACE)을 행하는 부분입니다. 프로젝트 어플리케이션을 작성할 때에는 전입 출력변수는 I/O 보드의 채널에 할당됩니다. I/O 보드는 이름과 보드의 SUPPLIER 를 나타내는 OEM 코드를 가지고 있습니다. 그 외 I/O 보드의 채널 수, 방향, 타입이 기술되어있습니다.

I/O 보드 파라미터

I/O 보드 파라미터에는 2 개의 부분이 있습니다. 공통적인 부분과 보드 고유의 부분입니다. 공통적인 부분에 관해서는 어떤 I/O 라이브러리 보드에 대해서도 필요가 있고, I/O 보드 파라미터 정의 박스의 상부에 입력 되어집니다. 보드 고유의 부분은 웨어하우스(WAREHOUSE)에 따라 제공 됩니다.

OEM 키 코드는 하드웨어 지원을 정의 하는 번호입니다. 동시에 지원은 동일한 OEM 키 코드를 가질 필요가 있습니다. OEM 키 코드는 16 비트 **unsigned word** 고 **16** 진수로 표현 될 필요가 있습니다. ICS Triplex ISaGRAF Inc 에서 예약되어 있는 번호는 0, 1 입니다.

[채널갯수]는 보드에서 취급 가능한 채널갯수 입니다. **[타입]**은 채널에 접속된 변수의 타입 입니다. **[방향]**은 접속 되어진 변수는 입력과 출력을 표시합니다.

주의: 한장의 보드에서 다른 타입, 방향을 가진 채널을 동시에 연결 하는 것은 불가능합니다.

☐ **OEM 파라미터**

I/O 보드 파라미터 정의 박스의 하부의 OEM 파라미터는 보드의 지원에 따라 정의 되어집니다. 보드 고유의 파라미터가 되어집니다. 1 장의 보드에 최대 **16** 개의 OEM 파라미터 (나뉘널기 번호 I/O 주소 메모리주소 . . 등) 가 set 세트 가능합니다. 보드에 따라서는 OEM 파라미터가 불필요한 경우도 있습니다. 각각의 파라미터 형식과 식별자는 하드웨어 지원이 정의 할 필요가 있습니다.

좌측의 그래프 박스에는 OEM 파라미터의 리스트가 표시되어 있습니다. 각각의 파라미터는 파라미터명과 논리번호 (**0-15**) 를 가집니다. 우측에는 선택 되어진 파라미터의 주석이 표시되어 있습니다. **[삭제]**버튼에 따라 선택중의 OEM 파라미터 삭제가 가능합니다.

정의된 OEM 파라미터 데이터 입력은 입출력변수와 I/O 를 접속시키기 위해 I/O 접속 편집기에서 합니다. 파라미터명은 다음의 규칙에 따를 필요가 있습니다

파라미터명은 최대 16 반각문자
최초의 문자는 영문자 (a - z)
이후의 문자는 영문자,숫자 _ 중에서

OEM 파라미터 **내부 형식**을 갖고 있습니다. OEM 파라미터 내부형식과 I/O 접속 편집기를 사용중인 OEM 파라미터 데이터 입력 때의 입력 형식은 다릅니다.

다음에서는 내부형식 예를 나타냅니다.

word ... unsigned 16 bit word
long unsigned 32 bit word
word hexa unsigned 16 bit word
long hexa unsigned 32 bit word
Boolean 형 unsigned 16 bit word (최하위 bit 만 사용됩니다.)
문자..... unsigned 16 bit word (최하위 바이트만 사용됩니다.)
문자열 1 6 바이트 문자열로 Null 로 끊겨 있다.
유동 소수 single precision 32 bit 유동 변수

I/O 접속 편집기 사용시의 입력 형식은 다음과 같습니다.

word ... unsigned decimal word
long decimal long word
word hexa unsigned hexadecimal word
long hexa unsigned hexadecimal long word
Boolean 형 "TRUE" or "false"
문자 single character
문자열 ascii string (15 characters max)
유동소수 single precision floating value

[사용자정의]

..... 체크박스에 따라 사용자가 정의하는 파라미터와 설정종료를 결정할 수가 있습니다. 체크박스가 체크되면 I/O 접속 편집기를 사용 할 시에 사용자를 입력할 필요가 있습니다.

[읽기전용].....

..... 체크박스에 따라, 사용자는 I/O 접속 편집기에서 파라미터를 나타낼 수 있지만 변경은 불가능합니다.

[숨김].....

..... 체크박스에 따라 사용자는 I/O 접속 편집기에서 파라미터를 볼 수 없습니다.

[기본값]

..... 값은 I/O 접속 편집기의 입력필드에 적힌 초기치를 나타냅니다. 체크박스가 체크 되어 있지 않는 경우는 파라미터 값은 정수(기본 값으로서 입력된 값)이고 또한 I/O 접속 편집기에 표시되지 않습니다.

기본 값으로서 입력된 값의 타입은 형식에서 선택한 타입과 일치해야 할 필요가 있습니다.

A.22.5 IEC언어로 작성된 Function, FunctionBLOCKS

ISaGRAF 는 IEC언어로 쓰여진 Function 과 FunctionBLOCKS 의 라이브러리를 가질 수 있습니다. 취급하는 언어는 **F B D L D**, **S T**, **I L** 입니다. F B D와L D는 함께 다를 수 없습니다. **S F C언어**는 함수에는 취급 불가능합니다. 또한 사용언어는 Function 과 FunctionBLOCKS 을 작성시에 사용하지만 도중에서 변경은 불가능 합니다.

Function, FunctionBLOCKS 컴파일러

IEC 언어로 쓰여진 Function, FunctionBLOCKS 의컴파일러는 ISaGRAF 프로젝트 중에 사용 되기 전에 컴파일러 (검증)되어야 만 합니다. 등록된 후는 L D / F B D편집기의 Function 을 선택하는 복스 내에 포함 됩니다.



Function 에서 라이브러리내의 다른 Function 을 호출할 수 없지만, 자신을 호출하는 recursive 는 지원 되지 않습니다.

또한, I E C언어에서 기술 되어진 FunctionBLOCKS 은 다른 FunctionBLOCKS(C 또는 I E C언어) 에서 읽어내지 못합니다. 즉, FunctionBLOCKS 계층은 2 개가 됩니다.



소스 코드 입력

IEC 언어의 Function, FunctionBLOCKS 소스 코드 즉, 프로그램 언어와 합친 편집기로 입력됩니다. 코드 생성은 직접 편집기에서 호출한 라이브러리 Function 과 FunctionBLOCKS 컴파일러 가 가능합니다.



지역변수 모음

라이브러리 Function, Function 라이브러리는 지역변수, 지역정의 워드를 가질 수 있습니다. 변수 설정을 위해 편집기 창에서 「파일」 - 「모음」 메뉴를 사용할 필요가 있습니다.



라이브러리 Function, FunctionBLOCKS 은 전역 변수와 전역 FunctionBLOCKS Instance 에는 액세스가 불가능합니다. Function 내에서 사용되는 local 변수는 Function, body 에서 최소화될 필요가 있습니다.

I E C 언어로 기술된 FunctionBLOCKS local 변수는 프로젝트 내에서 블록이 사용될 때 마다 Instance 로서 copy 됩니다. 블록이 차례로 읽어질 때까지 copy 된 값을 Hidden Instance 로서 보존하고 있습니다.



Interface 정의

새로운 라이브러리 Function, FunctionBLOCKS 이 작성되면, 호출 Interface 즉, 입출 파라미터를 정의할 필요가 있습니다. 입·출력 파라미터의 이름과 타입을 정의하기 위해 파라미터정의 박스가 사용됩니다. 「편집」 메뉴인 「파라미터」 명령어에 따라 함께 최대 3 2의 입력 파라미터와 출력 파라미터를 정의 합니다. (Function 에서는 하나의 출력 파라미터만..)

파라미터 Dialogue Box 상부 창에서는 함수 파라미터 (입 출력 파라미터) 가 표시되어 있습니다. 하부 창에는 현재 선택되어 있는 파라미터에 관한 상세한 정보를 나타내고 있습니다.

- 파라미터명
- 파라미터방향 (입출력)
- 파라미터타입

다음의 어떠한 타입도 파라미터로서 다룰 수 있습니다.

- Boolean 형
- 정수형
- 실수형
- 타임형
- 문자열형
- 출력 파라미터는 파라미터리스트 마지막에 있을 필요가 있습니다. 파라미터명에는 다음의 룰이 있습니다.
- - 파라미터명 최대장은 반각 1 6 문자
 - 최초의 문자는 영문자 (a - z)
 - 이후의 문자는 영문자, 숫자 중에
 - 대문자, 소문자 구별 없음

하나의 Function, FunctionBLOCKS 에서 동일한 파라미터명은 허용되지 않습니다. 동일 파라미터명은 다른 Function 에서는 다를 수 없습니다. 출력 파라미터명과 Function 명과는 같을 필요가 있습니다.

[삽입]버튼은 새로운 파라미터를 선택중인 파라미터앞에 삽입합니다.**[삭제]**버튼은 선택중의 파라미터를 삭제합니다. **[정렬]**버튼은 자동으로 파라미터순번을 나열을 바꿔 출력 파라미터가 마지막에 오도록 합니다.**[확인]**버튼은 파라미터정의를 보관해서 Dialogue Box 를 종료합니다.**[취소]**버튼은 파라미터정의의 변경을 보관하지 않고 Dialogue Box 를 닫습니다.

A.22.6 "C" Functions / function Blocks

IEC 에 따라 Function Interface 정의와 같은 모양인 C언어로 적혀진 Function, FunctionBlocks 은 어플리케이션 F B D와 S T 언어에서 호출됩니다.

ISaGRAF 어플리케이션은 이 함수들이 호출되면 함수의 처리가 끝날 때까지는 대기 상태가 됩니다. 처리가 종료되면 다시 ISaGRAF 어플리케이션이 실행됩니다.

새로운 Function, FunctionBlocks 이 작성되면, 호출 Interface 즉, 입출력 파라미터를 정의 할 필요가 있습니다. 입/출력 파라미터이름과 타입을 정의하기 위해 파라미터정의 박스가 사용됩니다. 「편집」 - 「파라미터」 명령어에 따라 합계 최대 3 2 입력 파라미터와 출력 파라미터를 정의합니다. (Function 어플리케이션의 경우는 출력 파라미터는 하나만)



파라미터정의는 I E C 언어에 따라 Function, FunctionBlocks 은 같지만, 다음에서는 C언어에 따른 경우의 다른 부분을 기술합니다.

C언어에 의한 Function, FunctionBlocks 의 경우 파라미터타입에 관한 ISaGRAF 에서의 타입과 C언어에서의 타입 대응도 다음에서 나타냅니다.

Boolean 형	Unsigned long	unsigned 32 bit word: 1=TRUE / 0=false
정수형	Long	signed integer 32 bit word
실수형	Float	단정도 유동소수치
타입형	Unsigned long	unsigned integer 32 bit word(단위는 1 m s)
문자열형	char *	character string.

메시지 타입의 경우 NULL 캐릭터를 포함시킬 수 없습니다. C 코드에서는 NULL 캐릭터는 메시지 (문자열) 의 종료를 의미합니다.

C 언어 Function, FunctionBlocks 의 상세한 것은 타겟 사용자가이드의 「C 언어 프로그램」을 참조 바랍니다.

A.22.7 변환 함수

아나로그 입출력 변수에 추가되는 수치 변환 함수는 ISaGRAF I/O 드라이브가 호출됩니다.

수치 변환 함수를 입출력 변수에 할당하기 위해서는 변수모음을 정의합니다.

변환 함수에서는 **전기적인 값 (외부)** 을 **어플리케이션 값 (내부)** 으로 변환합니다. 즉, 아나로그 입력변수 및 아나로그 출력변수 두개로 나눌 수 있습니다.

변환 함수에서는 **정수 아나로그, 실수 아나로그 값**의 어느쪽에서도 다를 수 있습니다. 내부적으로 함수는 유동 소수점 처리를 하고 있습니다. 어떤 변환 함수에서도 함수 Interface 부분은 같습니다. 변환 함수 C언어에 따라 Interface 부분은 **GRCN0DEF.H** Header 파일 내에 기술되고 있습니다.

매뉴얼 「C 언어 프로그램」에는 C 언어 함수, FunctionBlocks ISaGRAF 상에서의 관리, 짜넣기에 관해서 설명되어 있습니다.

A.23 보관함 사용법

ISaGRAF archive 유틸리티에 따라 사용자는 ISaGRAF 프로젝트와 라이브러리를 디스크에 백업하는 것이 가능합니다. Archive 유틸리티는 ISaGRAF 프로젝트 메니저와 라이브러리메니저에서 호출되는 Dialogue Box 입니다.



신뢰성 있는 archive 을 작성하기 위해 다음의 가이드 라인을 지킬 것을 권합니다.

- 디스크에 보관 되어 있는 내용을 기술해 둘 것
- 프로젝트와 라이브러리 디스크는 따로따로 할 것
- 다른 프로젝트 디스크는 따로따로 할 것

A.23.1 보관함 관리자 호출

보관함 유틸리티 Dialogue Box 는 프로젝트 메니저· 창의

「도구」 - 「보관함」 메뉴에서 작동합니다. 프로젝트 데이터, 공통 데이터의 보존, 복원이 가능합니다.

보관함 Dialog Box 는 ISaGRAF 라이브러리 관리자 에서도 작동 가능 합니다. 이 경우는 라이브러리 관리 편집기 중에 각종 편집기의 보존· 복원이 가능합니다.



프로젝트의 보관함

「파일」 - 「프로젝트」에 따라 프로젝트 백업과 복원이 가능합니다. 프로젝트의 모든 컴퍼넌트가 하나의 파일 (* . pia) 파일에 모아져 보존됩니다.

「옵션」 - 「압축」명령어를 set 하면 보관함되는 파일 사이즈를 작게 할 수 있습니다.



라이브러리 element 보관함

ISaGRAF 라이브러리 element (I/O 구성, I/O 장치 기기 수치변환 함수) 는 각각 개별적으로 백업 가능합니다. 하나의 라이브러리 element 를 구성하고 있는 모든 모듈 (기술메모, 정의, Interface, 소스코드...) 은 모아져 하나의 보관함 파일이 됩니다.



공통 데이터 보관함

「도구」 - 「보관함」 - 「공통 데이터」 명령어에 따라 ISaGRAF 로 사용되는 공통 데이터를 백업, 복원할 수 있습니다.

다음에서는 이 명령어로 copy 되는 파일을 나타냅니다.

common.eqv.....공통정의 워드

oem.bat.....사용자정의 MS-DOS 명령어 파일

이 파일들을 원래대로 보관함 파일에 보존됩니다. 대응하고 있는 보관함 파일은 압축되지 않습니다.

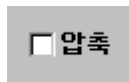
A.23.2 옵션

ISaGRAF 보관함 utility 가 사용하는 디렉토리는 Dialogue Box 하부에 표시됩니다. 「참조」 버튼을 눌러, 디렉토리를 Browser 로 다른 디렉토리와 디스크를 선택할 수 있습니다.



「옵션」 - 「압축 모드」가 set 되어 있는 경우는 백업 때에 생성되는 파일은 압축되어 보존됩니다. 보관함 파일의 사이즈를 작게 하는데 효과가 있습니다.

보관함 파일을 **복원할 때에는** 보관함 매니저가 압축모드를 자동 판별해서 복원하기 때문에 **압축모드**가 set 되어 있는지에 대한 것은 파일 복원 처리에는 관계없습니다.



A.23.3 백업 / 복구

「workbench」 리스트 (좌측 리스트) 는 ISaGRAF workbench 에 등록되어 있는 오브젝트를 나타내고 있습니다. 「보관함」 리스트 (우측 리스트) 는 지정된 보관함 디렉토리에 존재하는 오브젝트를 나타내고 있습니다.

☐ 백업

보관함 파일로서 보존 (즉, 백업) 할 경우에는 **좌측 리스트 박스** (ISaGRAF workbench 내의 파일) 에서 오브젝트를 선택해서 「**백업**」 버튼을 누릅니다. 동시에 하나 이상의 오브젝트를 선택할 수 있습니다. **우측 리스트 박스** (보관함된 파일) 의 오브젝트가 선택되어 있을 때는 「백업」 버튼을 선택할 수 없습니다. 「취소」 버튼은 백업, 복원 처리를 하지 않고, Dialogue Box 를 닫습니다.

☐ 복원

보관함 파일을 ISaGRAF workbench 에 복원할 경우에는 **우측 리스트 박스** (보관함 오브젝트) 에서 오브젝트를 선택해서 「**복원**」 버튼을 누릅니다. 동시에 하나 이상의 오브젝트를 선택하는 것도 가능합니다. **우측 리스트 박스** (ISaGRAF workbench) 의 오브젝트가 선택되어 있을 때는 「복구」 버튼은 선택할 수 없습니다. 「취소」 버튼은 백업, 복구 처리를 하지 않고, Dialogue Box 를 닫습니다.

A.23.4 보고한함 파일 확장자

ISaGRAF 보관함 utility 는 보존되는 보관함 파일에 다음과 같은 확장자가 부쳐집니다.

- .pia프로젝트
- .biaI/O 보드
- .iia.IEC 언어 Function
- .aiaIEC 언어 FunctionBLOCKS
- .uiaC Function
- .fia.CFunctionBLOCKS
- .ciaC 변환 함수
- .ria I/O 구성
- .xia.....I/O 장치 기기

A.24 인쇄

프로젝트관리 창「프로젝트」-「인쇄」명령어에 따라 프로젝트의 Document 인쇄가 가능합니다. 통상의「인쇄」명령어와 달라 단 한번에 복수 comperment 인쇄를 하지 않고, 목차, 페이지 번호 등도 자동으로 작성됩니다

「편집」메뉴 명령어는 인쇄되는 Document 내에 포함된 항목 선택에 사용됩니다. 즉, 인쇄 Document 목차를 작성하는 일에 해당합니다. 프로젝트에 관한 어떤 정보 (프로그램, 변수, 옵션, I/O 접속 . . .) 도 인쇄 Document 에 포함할 수 있습니다. 단, 다른 프로젝트 정보와 ISaGRAF 라이브러리정보는 포함되어 있지 않습니다.



「파일」-「인쇄」명령어에 따라 지정된 항목의 내용을 모아서 Document 인쇄를 합니다. Document 형식을 생성하기도 하는 관계에서 다소 시간이 걸리는 일도 있습니다. Document 인쇄는 하드 디스크에 여유 스페이스가 필요합니다. 만약, 하드 디스크 용량 부족으로 에러 메시지가 난 경우는 하드 디스크 여분의 파일을 삭제하든지 Document 내용을 축소해서 인쇄내용을 줄일 필요가 있습니다. 「인쇄」명령어 실행시에 Document 인쇄에 관한 메모를 기입할 수 있습니다. 이것은 인쇄 이력이 됩니다. 즉, 여기에 입력된 메모는 다음번 인쇄시에 인쇄 메모와 모아져 이력이 되어 첫째 페이지에 인쇄됩니다.

A.24.1 문서 목차 변환

「편집」메뉴에 따라 Document[목차]를 작성합니다. 모든 comperment (디폴트 설정) 또는 일부 comperment 선택이 가능합니다.



Default List

「편집」-「디폴트 리스트」명령어에 따라 Document 목차의 내용을 디폴트로 되돌립니다. 디폴트 설정에서는 모든 항목을 포함하고 있습니다.

- 프로젝트 기술
- 프로젝트 계층 (프로그램 사이의 Link 계)
- 소스 코드 (모든 프로그램)
- 수정이력 파일 (모든 프로그램)
- 공통 정의
- 전역정의

- Local 정의 (모든 프로그램)
- 전역변수 (모든 프로그램)
- 어플리케이션 옵션
- I/O 접속
- 변수 리스트
- 수치 변환 테이블
- Cross Reference 의 압축
- Cross Reference 의 상세함
- 설정 Summary
- Network Address Map
- 수정이력 (프로젝트)

목차는 「파일 - 보존」 명령어로 디스크에 보존 가능합니다. 이 명령어는 편집기 창에서 인쇄 명령어가 실행된 때에는 사용할 수 없습니다.



「잘라내기」, 「붙이기」

「편집」 - 「잘라내기」, 「붙이기」 명령어에 따라 목차 내용을 Customize 하기 위해 항목 리스트를 이동시킬 수 있습니다. 또한 한 번 만에 복수 항목 선택도 가능합니다. Document 작성 창에서의 클립보드는 표준 텍스트용 클립보드와는 다른 논리적인 정보를 가지고 있습니다.



항목 리스트 삭제

「편집」 - 「삭제」 명령어에 따라 항목 리스트 아이টে를 모두 삭제할 수 있습니다. 따라서 항목을 추가하는 것으로 새로운 목차를 작성하게 됩니다.



목차 항목의 추가

「편집」 - 「삽입」 명령어에 따라 아이টে를 추가 Dialog Box 가 열립니다. 여기에서는 프로젝트 인쇄항목의 추가가 가능합니다.

항목이 프로그램에 관한 경우는 프로그램마다 선택되기 때문에 프로그램 선택 박스에 프로그램을 지정해서, [추가] 버튼을 선택해 주십시오. 목차에는 같은 Component 는 한번 밖에 정의할 수 없습니다.

A.24.2 옵션

「옵션」 메뉴에 따라 문서를 Customize 할 수 있습니다.

기타 옵션은 문서 생성 창 버튼으로 선택 가능합니다.

「표지」 옵션은 프로젝트 타이틀과 인쇄 이력이 포함된 표지를 모두 문서 선두에 인쇄하기 위한 옵션입니다. 이 옵션이 설정되어 있지 않는 경우는 문서 선두 아이템이 첫째 페이지에 인쇄됩니다.

「목차」 옵션은 문서 마지막에 목차를 인쇄하기 위한 옵션입니다.

양쪽 옵션은 「인쇄」 명령어에서 문서 생성이 작성된 때는 디폴트에서는 설정되어 있지 않습니다.



SFC 차트 레벨 분배

「옵션」 - 「SFC 레벨 분배」 명령어가 set 되어 있을 때는 SFC 프로그램 인쇄시에 레벨 1 (차트와 명령어) 이 모두 인쇄된 후에 레벨 2 (프로그램 스테이트먼트) 가 인쇄됩니다. 이 명령어가 set 되어 있지 않는 때는 레벨 1, 2 가 같이 인쇄됩니다.



페이지 형식

「형식」 - 「페이지 형식」 명령어에 따라 인쇄페이지 형식을 파라미터를 설정합니다. 다음의 파라미터가 있습니다.

- 좌측 경계 : 1cm, 2cm, 없음 에서 선택가능
- 페이지 테두리 지정: 인쇄 페이지 테두리 지정

다음에서는 문서 인쇄의 예를 나타냅니다. 좌측 경계와 페이지 테두리 지정있음 (좌), 경계없음으로 페이지 테두리 없음 (우)

Hjllhkljllghk fskksjllhklf, hfkshms??i flmhkmlsfkhl hklkfhlmksmlf hfskljhlkfjsllf hfkiljllf hgfmkhmlk?rpkrio glfjklhkljllh htcmklfmhk hmlkfmkhmkfmh ICS	Hjllhkljllghk fskksjllhklf, hfkshms??i flmhkmlsfkhl hklkfhlmksmlf hfskljhlkfjsllf hfkiljllf hgfmkhmlk?rpkrio glfjklhkljllh htcmklfmhk hmlkfmkhmkfmh ICS
--	--



페이지 타이틀 박스 형식

「옵션」 - 「페이지 타이틀」 명령어에 따라 모든 페이지의 아래 부분에 인쇄할 타이틀 박스를 지정할 수 있습니다. 타이틀 박스 표준 윤곽을 아래에서 나타냅니다.

메인 타이틀의 첫째행 (ISaGRAF 프로젝트의 명칭) 과 날자, 페이지 번호에 관해선 문서 관리자가 자동으로 생성하기 때문에 변경 불가능합니다.

메인 타이틀의 둘째 행과 셋째 행의 **메모 1 ~ 메모 3** 에 관해서는 사용자가 설정할 수 있습니다. 또한 Dialogue Box 왼쪽 박스의 로그 (심벌) 도 변경 가능합니다. 이 로그를 변경할 경우는 변경하고 싶은 이미지 파일 (**BMP 파일**) 장소를 패스명 부치기에서 설정해 주세요. Bit Map 이미지는 정해진 사이즈에 돌아갈 수 있도록 자동적으로 리사이즈 됩니다. 인쇄시에 지정된 BitMap 이미지가 나타나지 않는 blank 가 되어 인쇄됩니다.



형식 선택

「옵션」 - 「텍스트 형식」, 「타이틀 형식」 명령어에 따라 인쇄되는 문서의 본문 (타이틀 페이지와 타이틀 박스부분 이외) 글꼴지정이 가능합니다. 타이틀 폰트는 인쇄 본문종이 항목명을 가리키고 텍스트 폰트는 이외의 본문을 가리킵니다. 글꼴지정은 Windows 표준 글꼴지정 DialogBox 에 따라서 합니다. 만약, 글꼴설정이 되어 있지 않은 경우는 표준 폰트가 선택되고, 더욱이 글꼴스타일로서

- 타이틀 : Bold 체
- 텍스트 : 레귤러체

가 있습니다.

A.25 암호 보호

보호레벨

하나의 프로젝트와 라이브러리 element 에 최대 **16** 개 액세스 레벨을 설정할 수 있습니다. 각 레벨에 대응한 암호가 존재합니다. 암호번호는 0에서 15 이고, **레벨 0 이 가장 높은 레벨**이 됩니다. 사용자가 암호를 알고있는 경우는 그 레벨에 대해서 등록되어 있는 조작은 물론 낮은 레벨 암호에 등록되어 있는 것의 조작도 가능합니다.

Pasrswd 가 사용된 방법의 예로서 새로운 어플리케이션 코드 생성을 할 경우 「어플리케이션 코드 생성」명령어에 대해서 Pasrswd 설정에 따라 프로젝트를 보호할 수 있습니다.

암호 보호정의

「**암호설정**」명령어에 따라 각각의 프로젝트와 라이브러리 element 에 대해서 암호와 그 access level 을 정의할 수 있습니다. 이 명령어는 ISaGRAF 「프로젝트 관리 창」 (프로젝트에 대해서) , 「라이브러리 관리 창」 (라이브러리 element 에 대해서) 메뉴에서 실행합니다. 암호가 아직 정의되지 않은 때는 이 「**암호설정**」명령어는 암호없이 사용가능하지만, 일단 하나라도 암호가 설정되면, 이후는 본인이 알고 있는 가장 높은 레벨의 암호를 입력한 후에 암호설정을 하게 됩니다. 이 경우 보다 높은암호는 변경할 수 없게 됩니다. 「**암호설정**」명령어에 따라 다른 액세스 레벨 설정과 정의된 레벨에서의 element 와 데이터 프로젝트를

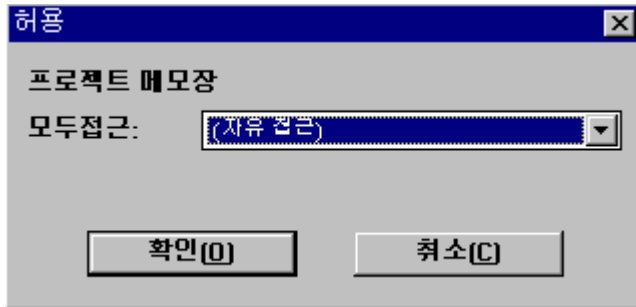


실행할 수 있습니다.

암호는 다이아로그박스 상부 텍스트 파일을 더블클릭해서 입력합니다. 아래 그림은 DialogBox 입니다.

다이아로그박스 하부에는 프로젝트 가능한 아이템 (데이터와 기능) 이 현재의 보호레벨이 표시됩니다. 보호레벨에는]리드 액세스]또는]Full 액세스]의 특권 레벨도 첨가되어 있습니다. 리드의 특권을 설정하면 그 암호를 모르는 사람이 대응하는 데이터를 보는 것도, 인쇄도 할 수 없도록 할 수 있습니다.

다이아로그박스 하부의 항목 필드를 더블클릭하면 특권을 설정하기 위해 아래 그림과 같은 다이아로그박스가 열립니다.



이것들의 특권은]free 액세스]에서도 정의한 보호레벨에서도 설정 가능합니다. Full 액세스 특권에는 Leader 액세스 보다 낮은 보호레벨은 설정할 수 없습니다.

프로젝트 기술 등의 문서는 원래 오픈된 것이기 때문에, 암호에 의한 프로젝트는 설정할 수 없는 것을 주의해 주십시오.

보호된 데이터 접근

암호가 설정된 경우에도 Workbench 가 시작될 때는 암호는 필요없게 됩니다. 사용자가 프로텍트되어 있는 기능과 파일에 액세스할 경우에만 다이아로그박스가 표시되고, 암호의 입력이 필요하게 됩니다.

사용자가 요구된 암호 (또는 그 이상의 레벨에 등록되어 있는 암호) 를 입력한 때는 조작을 계속할 수 있습니다. 한번 사용자가 암호를 넣은 후는 이 암호가 기억되고, 잇따른 조작으로 암호가 필요한 경우에도 재입력할 필요가 없습니다. 단, ISaGRAF Workbench 의 모든 창이 닫혀버린 경우는 다시 암호를 입력할 필요가 있습니다.

틀린 암호가 입력된 경우는 프로텍트되어 있는 기능은 사용할 수 없습니다.

☐ 보관함 과 연결

오브젝트 (프로젝트 또는 라이브러리 element) 를 보관함 디스크에 백업할 경우는 보호기능인[**보관함시의 백업**]이 유효하게 됩니다. (프로젝트션이 설정되어 있으면 암호의 입력이 됩니다.) 따라서 백업명령어도 프로젝션을 적용할 수 있습니다. .

오브젝트 (프로젝트 또는 라이브러리 element) 를 복원할 경우에 이미 Workbench 에 동일한 프로젝트가 있을 경우는 보호 기능인[**보관함 표시**]가 유효하게 됩니다. (프로젝트션이 설정되어 있는 경우는 프로젝트를 표시하기 위해서는 암호의 입력이 요구됩니다.)

☐ 변수와 I/O 채널에 대한 보호 설정

ISaGRAF Workbench 는 멀티레벨의 암호를 사용한 데이터 프로텍트·시스템을 제공합니다. 변수 설정과 I/O 접속에 대해서 데이터 모드를 모아서 암호에 따라 프로텍트하는 것이 가능하지만, 모든 변수와 I/O 접속에 대해 각각의 프로젝트를 설정하는 것이 가능합니다. 때문에 아래의 조건이 필요합니다. ;

- 암호가 「암호설정」명령어에 따라 설정 완료할것.

변수모음 및 I/O 접속에 대한 일괄의 보호레벨 보다도 높은 보호레벨을 가지고 있는 것.

변수 및 I/O 접속 각각에 프로젝트션이 설정되어 있는 경우는 모음편집기와 I/O 접속 편집기 변수명의 좌측에 작은 아이콘이 표시됩니다.

선택된 변수와 I/O 채널에 프로젝트를 설정하기 위해서는, 「편집」메뉴 「**보호설정**」 「**보호삭제**」명령어를 사용합니다. 어느쪽 명령어라도 첨가할 암호를 입력을 요구하기 때문에 그때는 각각 다른 레벨 암호를 입력할 수 있습니다.

주의: 만약, 변수와 I/O 채널이 암호프로텍션되어 있던 상태에서 관련있는 암호가 암호설정 다이아로그에서 삭제되어 버린 경우, 항상 동시에 그 이상의 암호가 설정되어 있지 않는 경우는 그 변수와 I/O 채널은 새로운 그것에 평형을 맞춘 충분한 레벨의 암호가 설정될 때까지 변경이 불가능하게 됩니다.

A.26 다른 테크닉..

본 장에서는 ISaGRAF Workbench 와 타겟 시스템에 대한 보다 상세한 정보를 해설합니다. 본 장을 읽기 전에 ISaGRAF 의 기본적인 지식을 준비해 둘 필요가 있습니다.

A.26.1 다른 ISaGRAF 도구

ISaGRAF 편집기 도구를 사용하고 있을때 **마우스 오른쪽 버튼**을 클릭하면 PopUp 메뉴를 열수 있습니다. 메뉴는 커서의 현재 위치에서 열기 위해 마우스 작동을 최소한 멈출수 있게 하는데 효과가 있습니다.

ISaGRAF Workbench 의 각 창은 처음은 디폴트 좌표에서 열리지만, 한번 창 사이즈와 장소를 설정해서 창을 닫으면, 이 정보들이 보관되기 위해서 다음 창을 닫을 때는 앞의 종료한 상태로 열수 있습니다.

ISaGRAF Workbench 편집기에서는, 다른 소스 파일의 편집은 동시에 창을 여는 일을 병행해서 작업을 진행할 수 있습니다. 단, 동일 소스 파일을 두개이상 열 수는 없습니다.

또한, 모든 프로젝트 내에서 도구바 공백 영역을 마우스로 더블클릭하면, 도구 바 커맨드 아이콘 전체 해설을 위해 PopUp 이 열립니다. 또한, 마우스 커서가 도구 바 커맨드 아이콘 위에 있을 때는 커맨드 텍스트 표시가 가능합니다.

A.26.2 잠금 I/O 와 가상 I/O

I/O 접속 편집기로 I/O 보드를 가상보드에 설정하는 것으로, 물리 I/O 채널 처리를 하지 않게 됩니다. I/O 보드가 가상과 정의되어 있어도 ISaGRAF 타겟에서의 처리에 변화는 없습니다. I/O 보드가 실 보드에 설정되어 있을 때와의 차이는 입력 device 로 읽어 넣기가 되지 않는 것, 출력 device 로 적어넣기가 되지 않는 것 뿐 입니다. 입력변수의 값 변경은 ISaGRAF 디버거를 사용하는 것에 따라 가능합니다. 가상 설정은 보드 단위로 하게 됩니다. (각각의 I/O 변수에 대해서는 불가능합니다.) 또한, 가상 보드의 설정은 I/O 접속 편집기로 정의한 후에 어플리케이션을 생성할 필요가

있습니다. 가상 보드의 설정 (속성 설정) 은 Static 이고 어플리케이션 코드에 포함되기 때문에 어플리케이션이 시작할 때 가상설정에서 시작합니다.

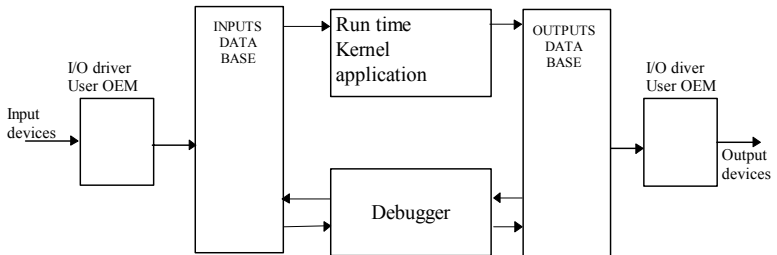
이것과는 달리 I/O 변수를 **잠금**하면 물리 I/O 와 분리하는 것도 가능합니다. I/O 변수의 잠금· Un 잠금은 ISaGRAF 디버거에서 커맨드로 합니다. 변수의 잠금은 **다이내믹한** 조작이고, 어플리케이션 코드에는 보존되지 않습니다. **변수의 잠금조작은 한번에 하나의 변수에 대해서 가능합니다.**

다음에서는 테이블에 가상 보드 설정, I/O 변수의 잠금에 대해 정리합니다.

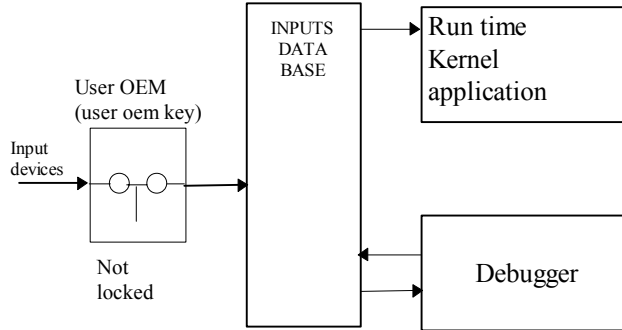
가상 I/O 보드 I/O 변수 잠금

설정 도구 I/O 접속 편집기	디버거
정의 Static다이내믹	
선택모드 보드 단위	변수 단위
적용 예 검증, 테스트	Maintenance

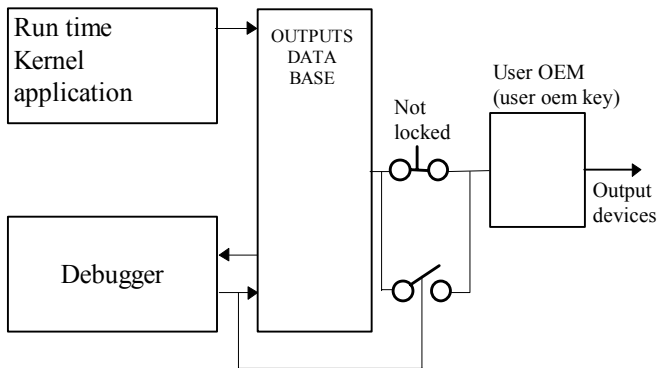
아래의 그림에서 ISaGRAF 에서의 I/O 데이터 흐름을 나타냅니다.



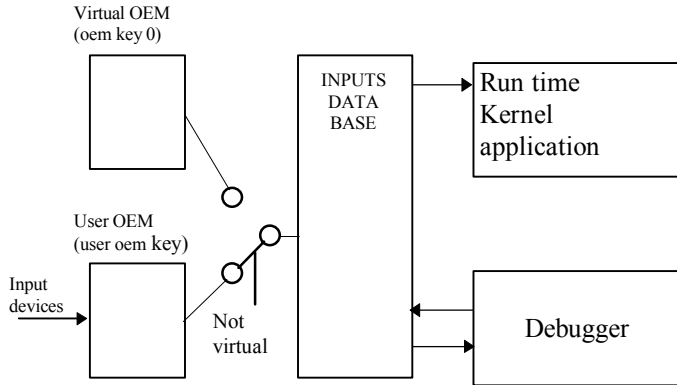
입력변수가 잠금 되어 있을 때는 입력 device 로 여러가지 액세스에 변경은 없지만, device 와 입력 device 는 잠금되어 (분리되어) 있습니다. 입력 device 에서는 디버거가 데이터 적어넣기를 하고, ISaGRAF Run-time Kernel 에 따라 통상대로 처리됩니다.



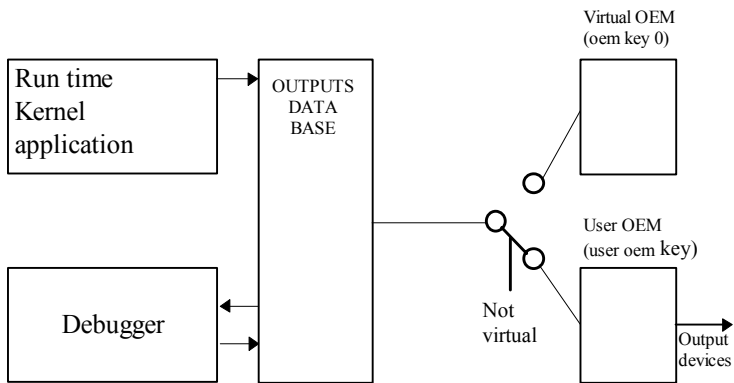
출력 변수 잠금 되어 있을 때, Runtime Kernel 과 출력 device 사이는 잠금 되어 (분리되어) 있습니다. 이 경우는 출력 device 을 통해서 출력 device 에서의 액세스는 디버거에 따라 가능합니다



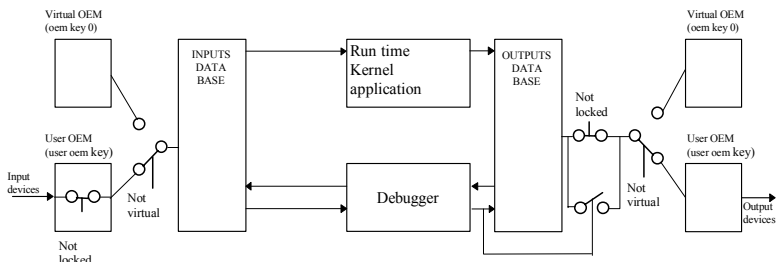
가상입력보드의 설정은 입력 device 와 사이가 분리되게 합니다. 가상/O 드라이브는 실제 I/O 드라이브를 대신해서 사용 됩니다.



출력 보드 가상의 설정은 입력보드 가상의 설정과 같은 모양으로 출력 device 와 출력 device 의 사이가 분리됩니다. Kernel 은 출력 device 을 변경시킵니다. 가상 I/O 드라이브가 실제 I/O 드라이브를 대신해서 사용 됩니다..



To summarize all possibilities:



A.26.3 PC-PLC link validation

통신 Link 설정 (Workbench 타겟) 동작 확인

디버거창의 상태 설정이[차단상태]가 되어 있는 경우는 대개 Workbench 와 타겟 간의 통신에러가나는 일이 많습니다. 원가 구체적인 프로그램 타겟 상에서의 동작 확인하기 전에 반드시 타겟 에서의 실행중인 어플리케이션이 없는 상태에서 Workbench 와 타겟 사이에 통신이 정상인가를 확인해 둘 필요가 있습니다.

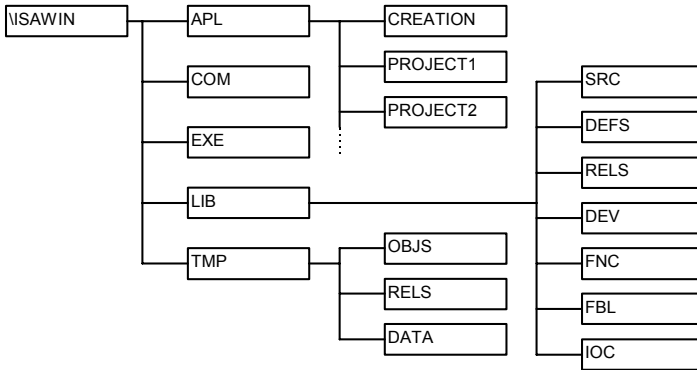
이 경우는 디버거메시지가[어플리케이션이 없습니다.]가 됩니다. 변환함수와 C언어 함수에서 사용자프로그램으로 C언어가 사용된 경우는 사용자프로그램이 타겟 시스템에 직접 액세스하면 타겟 시스템 에러가 되는 일도 생각할 수 있습니다.I SaGRAFI/O 개발 도구 kit 에서 I/O 드라이브를 C언어로 기술할 경우, 예를 들면 보드설정 어드바이스를 틀리게 한 것도 포함됩니다.

아래 표에 디버거메시지에서 판단할 수 있는 현상의 원인 예를 간단히 나타냅니다.

Status	모드	원인 (예)
"차단상태" (다운로드 전)		- 타겟 시스템이 올라가 있지않음. - 통신 케이블이 없고 정상이 아님. - 통신 설정 데이터 에러 - 타겟이 정상으로 인스트를 되어 있지 않음.
"차단상태" (다운로드 후)	사이클모드	I/O 구성 에러
	작동모드	타겟 시스템이 정상이 아님
	Real time 모드	I/O 구성 에러
	작동모드	- 타겟 시스템이 정상이 아님 - (C 프로그램 에러)
"어플리케이션 이 없습니다."		- 어플리케이션이 되어 있지 않음. - "어플리케이션이 RUN 하고 있지 않음 - (C 프로그램 에러) - Intel/Motorola 의 TIC 코드가 부정합 - 타겟 어플리케이션 No.가 불일치

A.26.4 ISaGRAF 디렉토리 구성

ISaGRAF Workbench 는 특정 디스크 디렉토리 구성의 근본으로 동작합니다. 루트 디렉토리명은 Workbench 스트롤시에 설정가능 합니다. 디폴트에서는 "Isawin" 디렉토리가 됩니다. 다음에서는 Workbench 가 만드는 표준 디스크 디렉토리 구성을 나타냅니다.



다음에는 ISaGRAF 서브 디렉토리에 대해 설명하고 있습니다.

디렉토리	내용
APL	ISaGRAF 프로젝트를 위한 루트 디렉토리명 프로젝트는 대응한 하나의 디렉토리를 가집니다. 다음의 서버 디렉토리에는 프로젝트에 필요한 데이터를 모두 포함하고 있습니다.
	기타 프로젝트 loop 를 작성하고, 다른 디렉토리에 프로젝트 디렉토리를 작성하는 것도 가능합니다. 설치러는 샘플 어플리케이션이 들어 있던]SMP]라는 디렉토리를 작성합니다.
COM	공통 데이터 프로젝트간에 공통으로 다루는 데이터
EXE	ISaGRAF 프로그램과 도움말 파일

LIB	ISaGRAF 라이브러리 데이터 - 라이브러리 element - 각 element 에 대한 파라미터와 Interface - 기술 메모
LIB\IOC	I/O 구성 소스 코드
LIB\FNC	I E C언어로 기술 된 Function 소스 코드
LIB\FBL	I E C 언어로 기술 된 FunctionBLOCKS 소스 코드
LIB\SRC	변환 함수 C언어 Function 소스 코드
LIB\DEFS	변환 함수 C언어 Function 헤드파일
LIB\RELS	변환 함수 C언어 Function 오브젝트 코드
LIB\DEV	C언어 라이브러리를 작성하기 위한 커맨드 파일 ("C" libraries, makefiles, link lists....)
TMP	Temporary 파일 ISaGRAF 코드 Generation Temporary 파일 을 보존하기 위한 영역으로 삭제할 수 없습니다.

이 서브디렉토리들은 다른 디스크 (또는 서버 디렉토리) 로 이동할 수 있습니다. 만약, 표준 이외의 디스크 디렉토리를 구성할 경우는 서버 디렉토리의 패스명은 EXE 서버 디렉토리내의 **ISA.ini** 파일인 **WS001** 섹션에 기술해야만 합니다.

다음에서는 테이블에 **WS001** 섹션 엔트리 포인트를 나타냅니다.

Isa	ISaGRAF Workbench 루트 디렉토리명
IsaExe	ISaGRAF 프로그램과 도움말 파일용 루트 디렉토리명
IsaApl	ISaGRAF 프로젝트용 루트 디렉토리명
IsaTmp	Temporary 파일용의 디렉토리명
IsaSrc	라이브러리 소스 코드용의 디렉토리명
IsaDefs	라이브러리 소스 헤드용 디렉토리명

주의 : 만약, IsaTmp 디렉토리를 다른 디렉토리에 변경할 경우는 반드시 서버 디렉토리 OBJS, RELS, DATA 를 아래에 작성할 필요가 있습니다. 다음에 **WS001** 섹션을 변경한 custom 디렉토리 구성의 경우의 설정 예를 나타냅니다.

```
;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
```

C 언어 Function 과 FunctionBLOCKS 을 추가할 경우는 \ISAWIN\LIB\DEV 디렉토리에 커맨드 파일, make 파일 등을 보존할 수 있습니다. 또한, \ISAWIN\LIB\RELS 디렉토리에는 C 컴파일러로 만들어진 오브젝트와 link 에 필요한 C 라이브러리를 보존할 수 있습니다.

A.26.5 어플리케이션 심벌 파일 (APPLI.TST)

ISaGRAF 어플리케이션의 각 오브젝트는 이름 (변수명) 과 **가상주소** (코드 생성시에 계산됩니다.) 에 따라 참조됩니다. 여기서 가상주소와 변수 선언시에 입력수 네트워크 주소와는 다릅니다. 가상주소는 I/O 드라이브 개발 kit 을 포함한 C 언어 어플리케이션에서 액세스할 때 드에 사용됩니다. ISaGRAF 코드 생성시에 프로젝트를 구성하고 있는 프로젝트명 (변수, 프로그램명, 스텝명 . . .) 과 가상주소와의 관계가 하나의 하나의 ASCII 파일에 보존됩니다. 이 파일에는 user 어플리케이션이 ISaGRAF 문자열데이터를 액세스하기 위한 정보가 포함되어 있습니다. 이 파일명은 "**APPLI.TST**"로 "**ISAWIN\APL\prname**" (prname 은 프로젝트명) 디렉토리에 포함되어 있습니다. 다음에 "**APPLI.TST**" 기술 형식에 대해 나타냅니다.

주된 용어로 다음의 3 개가 있습니다.

```
VA:.....가상주소
ATTR: .....변수 속성
USP:.....C 언어 Function
```

변수 속성으로서는 다음의 네가지가 있습니다. 이들은 "**변수 속성**" 파일에 적혀 있습니다.

```
+X.....내부 변수
+C.....정수 ( 일기전용내부변수 )
+I.....입력 변수
+O.....출력 변수
```

가상주소 이외의 모든 번호는 10 진수로 표현됩니다. 가상주소 (VA) 는 "!"가 선투로 16 진 4 행으로 표현됩니다.

예를 들면,

123 10 진수
!A003..... 16 진수 가상주소

"APPLI.TST" 파일 구성은 다음에 표시합니다. 파일은 복수 블록에서 이루어집니다. 블록은 복수 레코드에서 이루어집니다. 각 코드는 1 행의 텍스트로 기술됩니다. 각 블록은 1 행의 헤드에서 개시됩니다.

Start Blocks
De 시뮬레이션메모 ion Blocks
End Blocks

하나의 블록의 예를 나타냅니다.

```
@ <Blocks_name> <arguments>
#record...
#record...
...
```

start Blocks 에는 어플리케이션 기본 정보가 포함되어 있습니다.
다음에 그 예를 나타냅니다.

```
@ISA_심벌 S,<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
```

여기서 < >내의 약칭은 아래와 같습니다.

appli_crc.....어플리케이션 심벌 체크섬
appli_name.....어플리케이션명
version.....ISaGRAF workbench N O .
creation_date어플리케이션 작성 일시
nbprg프로그램 수
nbstep..... S F C 스텝 수
nbtra..... S F C transition 수
nbpro C언어 Function 수
nbvar.....모든 변수의 수

end Blocks 에는 파일 마지막의 의미가 포함되어 있습니다. 아래와 같습니다.

@END_심벌 S

de 시물레이션메모 ion Blocks 에는 어플리케이션 프로그램 기술이 포함되어 있습니다.
다음에서 그 예를 나타냅니다.

```
@PROGRAMS,<nbprg>
#<va>,<name>
#...
```

nbprg 이 블록에 정의되어 있는 프로그램 수
va 프로그램 가상주소
name 프로그램명

SFC 스텝을 기술하는 블록을 아래에서 나타냅니다. S F C 이외의 프로그램에는 가상적인
스텝이 정의 되어 있습니다.

```
@STEPS,<nbsteps>
#<va>,<name>,<father>
#...
```

nbsteps 이 블록에 포함되어 있는 스텝 수
va 스텝의 가상주소
name 스텝명
father Father 프로그램 가상주소

SFC transition 을 기술하는 블록을 아래에서 나타냅니다.

```
@TRANSITIONS,<nbtrans>
#<va>,<name>,<father>
#...
```

nbtrans 이 블록에 포함되는 transition 수
va transition 가상주소
name transition 명
father Father 프로그램 가상주소

Boolean 형 변수를 기술하는 블록을 아래에서 나타냅니다.

```
@BOOLEANS,<nb_boo>
#<va>,<name>,<attr>,<program>,<eq_false>,<eq_TRUE>
#...
```

변수의 수가 4095 을 넘은 경우는

```
X#(1.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_TRUE>
```

nb_boo..... 이 블록에 포함되는 변수의 수
va 변수의 가상주소
varno..... Boolean 형 변수 내에서의 주소
name 변수명
attr..... 변수속성
program Father 프로그램 가상주소
..... 또는 "**I0000**" (전역변수 때)
eq_false FALSE 시의 문자열
eq_TRUE TRUE 시의 문자열

정/실수형 변수를 기술하는 블록을 아래에서 나타냅니다.

```
@ANALOGS,<nb_ana>
#<va>,<name>,<attr>,<program>,<format>,<unit>
#...
```

변수의 수가 4095 을 넘은 경우는

```
X#(2.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_TRUE>
```

nb_ana..... 이 블록에 포함되는 변수의 수
va 변수의 가상주소
varno..... 정· 실수형 정수에서의 주소
name 변수명
attr..... 변수 속성
program Father 프로그램의 가상주소
..... 또는 "**I0000**" (전역변수 때)
format = "**I**" (정수형 변수)
..... = "**F**" (실수형 변수)

unit..... 단위를 나타내는 문자

타임형 변수를 기술하는 블록을 아래에서 나타냅니다.

```
@TIMERS,<nb_tmr>
#<va>,<name>,<attr>,<program>
#...
```

변수의 수가 4095 을 넘을 경우는

```
X#(3.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_TRUE>
```

nb_tmr 이 블록에 포함되는 변수의 수
va 변수의 가상주소
varno 타임형 변수 내에서의 주소
name 변수명
attr 변수 속성 (항상 +X: 내부변수)
program Father 프로그램의 가상주소
..... 또는 "!0000" (전역변수 때)

문자열형 변수를 기술하는 블록을 아래에서 나타냅니다.

```
MESSAGES,<nb_msg>
#<va>,<name>,<attr>,<program>,< max_len>
#...
```

변수의 수가 4095 을 넘을 경우는

```
X#(4.<varno>),<name>,<attr>,<program>,<eq_false>,<eq_TRUE>
```

nb_msg 이 블록에 포함되는 변수의 수
va 변수의 가상주소
varno 문자열형 변수 내에서의 주소
name 변수명
attr 변수속성
program Father 프로그램의 가상주소
..... 또는 "!0000" (전역변수 때)
lg_max 메시지 최대장

C언어 Function 을 기술하는 블록을 아래에서 나타냅니다.

```
USP,<nb_usp>
#<va>,<name>
#...
```

nb_usp 이 블록에 포함되는 C 언어 Function 의 수
va C 언어 Function 의 가상주소
name C 언어 Function 명

C 언어 FunctionBLOCKS 을 기술하는 블록을 아래에서 나타냅니다.

```
FB INSTANCES,<nb_fb>
#<va>,<inst_name>,<fb_name>
#...
```

nb_fb C 언어 FunctionBLOCKS instance 의 수
va C 언어 FunctionBLOCKS instance 의 가상주소
inst_name C 언어 FunctionBLOCKS instance 명
fb_name C 언어 FunctionBLOCKS 타입

A.26.6 ISaGRAF workbench I/O 무제한판 (W D L) 의 제한

ISaGRAF workbench (I/O 무제한판) 에 의한 제한을 다음의 표에서 나타냅니다.
 물론 이외에 workbench 에 사용하고 있는 컴퓨터 메모리, 하드 디스크 등의 제한,
 타겟 시스템에서의 메모리, 리소스 등의 제한도 있습니다.

프로젝트:

오브젝트	초/대값	메모
프로그램	255	
계층레벨 수	20	

프로젝트 수는 사용중인 컴퓨터 하드 디스크 용량에만 제한을 받습니다.

오브젝트 이름:

오브젝트 명	초/대값	메모
프로젝트	8 char	
프로그램	8 char	
변수	16 char	커맨드에 대해 다시 플러스 60 characters
정의 워드	16 char	

정의 워드의 내용	255char	커맨드에 대해 다시 플러스 60 characters
변환 테이블	16 char	
변수 리스트	16 char	
Function F B (라이브러리)	8 char	C 언어 Function F B, I E C 언어 Function F B
Function 파라미터 (라이브러리)	16 char	C 언어 Function, F B, I E C 언어 Function, F B
I/O 보드	8 char	
I/O 구성	8 char	
O E M파라미터	16 char	
변환 함수	8 char	

☐ 편집 (하나의 프로그램 내에서) :

오브젝트	초/대값	메모
S F C 행	600	
S F C 열	20	
S F C 스텝	4095	하나의 프로젝트 전체에서 (마이크로 스텝, 이니셜 스텝, end 스텝을 포함)
S F C transition	4095	하나의 프로젝트 전체에서
L D / F B D 편집기	200 열 2000 행	셀 편집 영역.
QuickLD 편집기	-	P C 의 제한에 따릅니다.
I L 라벨	251	동일 I L 프로그램내
텍스트	40KB	하나의 프로그램내의 텍스트 합계가 40KB 이내 (Win95 의 경우 24KB)

☐ 모음 (하나의 프로젝트 내에서) :

오브젝트	초/대 r 값	메모
Boolean 형변수	65535	
정수·실수형 변수	65535	정수형과 실수형의 합계
타입형 변수	65535	
문자열 형 변수	65535	
정의워드	4095	동일 범위 (local, 글로벌)

정의 워드	255	하나의 프로그램내
변환 테이블	127	하나의 프로젝트내
테이블내의 포인트	32	하나의 변환 테이블내

이상의 최대값은 각 속성 (내부, 입력, 출력, 정수) 의 합계를 나타냅니다. 또한 내부 temporary 와 컴파일러가 자동적으로 생성하는 것도 포함합니다. 모음 편집기로 한번에 편집가능한 변수는 각 변수 타입, 스코프마다 16000 개를 넘을 수 없습니다. PC 메모리의 용량에 따라 16000 다음이 되는 경우도 있습니다. 변수의 수가 4095 을 넘는 어플리케이션의 경우는, V2.21 이전 버전의 타겟에서는 실행할 수 없습니다. 또한, 네트워크 주소를 사용한 표준 MODBUS 통신은 4095 을 넘는 변수에서는 불가능합니다.

☐ **I/O 접속 편집기:**

오브젝트	최대값	메모
I/O 보드	256	하나의 프로젝트내 (보드 또는 장치 기기의 합계)
I/O 채널	128	동일 보드

I/O 보드의 수, 장치기기의 항목은 256 을 넘을 수 없습니다.

☐ **라이브러리:**

오브젝트	최대값	메모
I E C언어 Function	255	라이브러리에 등록 가능 최대 값
I E C언어 F B	255	라이브러리에 등록 가능 최대 값
C언어 Function	255	라이브러리에 등록 가능 최대 값
C언어 F B	255	라이브러리에 등록 가능 최대 값
C언어 F B instance	4095	하나의 프로젝트내의 동일 타입의 F B 에 대해
Function 입력 파라미터	31	C언어 Function, I E C언어 Function
함수 파라미터총수	32	적어도 하나의 출력 파라미터이외는 입출력할당은 자유
변환 함수	128	라이브러리에 등록 가능 최대 값
I/O 구성	255	라이브러리에 등록 가능 최대 값

I/O 보드	255	라이브러리에 등록 가능 최대 값
I/O 장치 기기	255	라이브러리에 등록 가능 최대 값
I/O 보드 O E M 파라미터	16	

B. 언어

B.1 프로젝트 구성 (Architecture)

SaGRAF 프로젝트는 복수 프로그램으로 성립되어 있습니다. 각각의 프로그램이 tree 구성으로 link 되고, 하나의 프로젝트를 형성하고 있습니다. 프로그램은 **SFC, FC (Flow Chart), FBD, LD, ST, IL** 의 **6-언어** 에서 만들어집니다.

B.1.1 프로그램

프로그램은 하나의 unit 한 것으로, 프로세스 중의 변수사이의 조작을 기술하고 있습니다. 프로그램에는 sequential 한 부분과 Cyclic 한 부분이 있습니다. Cyclic 한 프로그램과는 타겟의 Scan 마다 실행되는 프로그램이고, sequential 한 프로그램은 S F C 언어 또는 F C 언어의 룰에 기초해서 실행됩니다.

각 프로그램은 서로 계층적으로 link 되고 있습니다. top 레벨의 프로그램은 시스템에 따라 작동됩니다. 서버프로그램 (하위 계층) 은 이들의 Father 프로그램에 따라 작동됩니다. 각 프로그램은 6 종류의 언어의 장소에 따라서 그 중 하나의 언어로 기술됩니다:

Sequential Function Chart (SFC) 고급 레벨 프로그램

Flow Chart (FC) 고급 레벨 프로그램

Function Block Diagram (FBD) cyclic complex 동작

Ladder Diagram (LD) 2 진 값 동작만

Structured Text (ST) 모든 cyclic 동작

Instruction List (IL) 낮은 레벨 동작

LD 와 FBD 가 동일 프로그램 중에 같이 사용할 수 있다는 것을 제거하면, 동일 프로그램에는 복수 프로그램을 섞이게 할 수 없습니다.

B.1.2 Cyclic 및 sequential 동작

프로그램 계층은 5 개의 메인 그룹(Section)으로 나뉘집니다.:

Begin	매 사이클의 처음에 실행 되는 프로그램
Sequential	다이나믹한 룰에 따른 SFC / FC 프로그램
End	매 사이클 마지막에 실행 되는 프로그램
Functions	서브 프로그램 집합

'Begin' 과 'End' 섹션은 Cyclic 한 동작을 기술하고, 시간에 의존하지 않는 프로그램입니다. **Sequential** 섹션 프로그램은 sequential 한 동작을 기술하고, 같은 시기 등의 시간 의존형 프로그램입니다. 'Begin' 섹션 메인 프로그램은 runtime 사이클마다 처음에 실행됩니다. 'End' 섹션 메인 프로그램은 runtime 사이클 마지막에 실행됩니다. 'sequential' 섹션 메인 프로그램은 SFC 와 FC 다이내믹 룰에 기초해서 실행됩니다.

"역선" 섹션 프로그램은 프로젝트 내의 다른 프로그램에서 호출되는 서버 프로그램을 나타냅니다.

"역선" 섹션 프로그램은 같은 섹션 프로그램을 호출할 수 있습니다.

sequential 섹션에서의 메인 Child 프로그램은 반드시 **SFC** 또는 **FC**언어로 기술되어야만 합니다. 사이클릭 섹션의 **begin** 과 **end** 는 **SFC** 또는 **FC**언어에서는 기술할 수 없습니다. 어느 프로그램도 하나 이상의 서버프로그램을 가질 수 있습니다. sequential 섹션 프로그램은 하나 이상의 **SFC**, **FC** Child 프로그램을 가질 수 있습니다. 서버프로그램은 **SFC** 또는 **FC**언어로 기술할 수 없습니다.

Begin 섹션 프로그램은 입력 디바이스 필터링해서 집어넣는 초기적인 처리 (변환 관수 등) 에 사용되는 일이 많습니다. 이렇게 집어넣는 변수는 자주 **sequential** 섹션 으로 사용됩니다. **End** 섹션 프로그램은 **sequential** 섹션 으로 사용된 후 처리 (내부변수를 외부 출력 디바이스로 송출하는 것같이 처리) 를 할때 사용됩니다.

B.1.3 Child SFC / FC 프로그램

sequential 섹션의 어떤 **SFC**, **FC**프로그램도 다른 **SFC**, **FC**프로그램을 컨트롤할 수 있습니다.

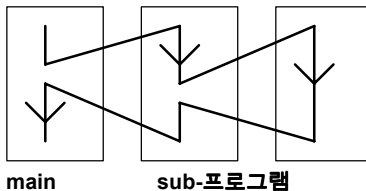
Child SFC 프로그램은 parallel 처리프로그램으로 다루어집니다. 즉, Father 프로그램에서 작동, 정지(kill), 동결(frozen), 재작동 시킬 수 있습니다. 단, Father 프로그램도 Child 프로그램도 **SFC**언어로 적혀져 있어야만 합니다. **Child SFC 프로그램**은 local 변수, local 정의를 가질 수 있습니다.

Father 프로그램이 Child **SFC** 프로그램을 작동 시켰을때 **SFC token** 을 Child **SFC** 프로그램 이니셜 스텝에 set 합니다. 이 작동 커맨드는 **GSTART** 스테이트먼트로 기술됩니다. 만약 Father 프로그램이 Child **SFC** 프로그램을 정지(kill)시키면 Child 프로그램 **스텝**에 존재하고 있던 모든 token 을 clear 시킵니다. 이 정지 커맨드는 **GKILL** statement 로 기술됩니다

Father 프로그램이 Child S F C 프로그램을 동결(frozen)시키면 Child 프로그램 존재하고 있던 모든 token 을 clear 하지만, 이들의 상태는 메모리에 보존됩니다. 이 동결 커맨드는 **GFREEZE** 스테이트먼트로 기술됩니다. Father 프로그램이 동결한 Child S F C 프로그램을 재작동하면, 동결시에 clear 된 token 가 복원됩니다. 이 재작동 커맨드는 **GRST** 스테이트먼트로 기술됩니다.

F C 프로그램에서 sequential 섹션 F C 프로그램은 다른 **F C 서버프로그램 (Child F C 프로그램)**을 읽어 낼 수 있습니다. S F C과 다른 Father F C 프로그램은 F C 서버프로그램이 실행 중에는 처리가 서스펜드되어 있습니다. F C 프로그램에서는 Father 프로그램과 F C 서버프로그램이 동시에 실행할 수 없습니다. 서버프로그램은 Father 프로그램에서 작동됩니다. 단, 이때 Father 프로그램의 실행은 서버프로그램이 종료할 때까지 suspend 됩니다.

B.1.4 Functions / 서버-프로그램



어떤 섹션 프로그램도 하나 이상의 서버프로그램을 가질 수 있습니다. 서버프로그램은 한사람의 Father 프로그램에 따라 소유됩니다. 서버프로그램은 local 변수와 local 정의를 가질 수 있습니다. S F C, F C 이외의 프로그램은 서버프로그램으로서 기술할 수 있습니다. **"Function"** 섹션 프로그램은 프로젝트내의 다른 프로그래에서 호출되는 서버프로그램으로서 사용됩니다. **"Function"** 섹션 프로그램은 같은 섹션의 다른 프로그램을 호출할 수 있습니다.

주의: I S a G R A F 는 Recursive 한 처리를 support 하고 있지 않습니다. 만약 **"Function"** 섹션 프로그램이 자신과 자기의 서버프로그램에서 호출되는 경우와 같은 runtime 에러가 발생합니다.

서버프로그램의 interface 는 타입과 유일한(unique) 이름을 입출력 parameter 를 갖고 정확하게 정의되어야만 합니다. ST 언어의 룰에 기초해서 출력 parameter 명은 서버프로그램과 같은 이름 이어야 할 필요가 있습니다..

다음에서 서버프로그램 본체에 출력 parameter 를 설정하는 방법을 언어마다 정리합니다.:

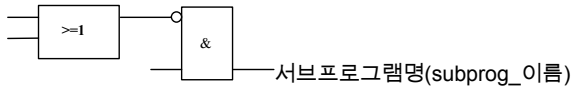
ST: 출력 parameter 는 서브프로그램명과 같게 합니다
(the same 이름 as the sub-program):

subprog_이름 := <expression>;

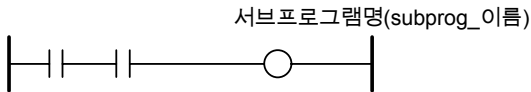
IL: sequence 마지막 에서 현재 결과의 값 (IL register) 이 출력 parameter 에 set 합니다:

```
LD    10
ADD   20  (* return parameter value = 30 *)
```

FBD: 출력 parameter 와 서브프로그램명을 같게 합니다:



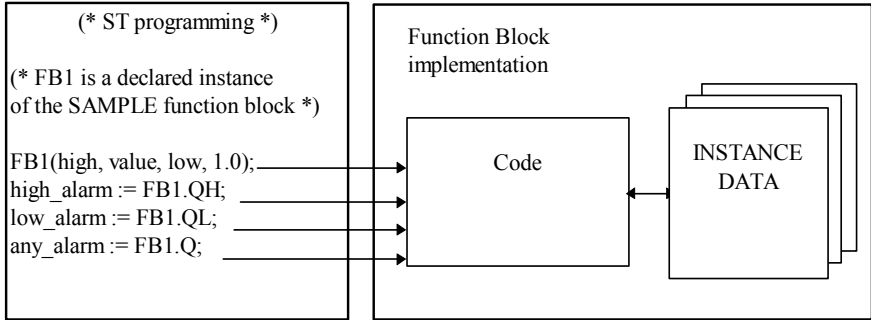
LD: 출력 parameter 와 서브프로그램명을 같게 합니다:



B.1.5 Function blocks

Function blocks 은 LD, FBD, ST 또는 IL 을 사용하여 작성 할 수 있다 :

function block 의 지역 변수들은 각각의 인스턴스를 복사 하면서 초기화를 한다는 의미이다.
프로그램 내에서 block 을 호출 한다는 것은 , block 의 인스턴스를 호출 하는 것 이다:



주의:

- function block 이 IEC 언어중 하나로 기술되었다면 다른 function block 을 호출 하는 것은 불가능 하다: instantiation mechanism 은 단지 block 의 지역변수 자체만 관여 하기 때문이다. 다음은 표준 function block 의 항목들 이다. 역시 IEC function block 내에서는 사용 불가능 하다:

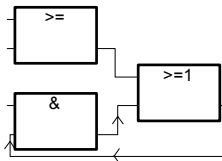
SR, RS, R_Trig, F_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM_ALARM, INTEGRAL, DERIVATE, BLINK, SIG_GEN

- 같은 이유로 다음은 사용이 불가능 하다 : Positive / Negative 연결 / 코일 / Set / Reset 코일.

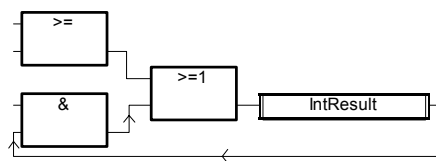
- 타이머를 시작/정지 시키기 위한 TSTART -- TSTOP functions 은 function block 에서 사용 가능 하다.(버전 3.2 이후 가능)

- function block 내에서 루프가 필요한 경우,루프를 사용하지전에 지역변수를 사용 하여야 한다.

동작 않함:



동작:



B.1.6 6-언어

프로그램은 다음의 6-언어(그래픽/문법)중 어느것으로도 작성 가능 합니다:

Sequential Function Chart (SFC) 고급 레벨 프로그램

Flow Chart (FC) 고급 레벨 프로그램

Function Block Diagram (FBD) cyclic complex 동작

Ladder Diagram (LD) 2 진 값 동작만

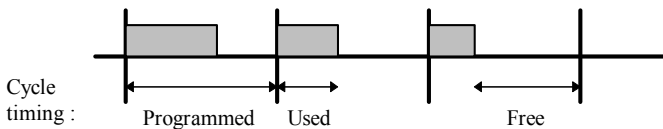
Structured Text (ST) 모든 cyclic 동작

Instruction List (IL) 낮은 레벨 동작

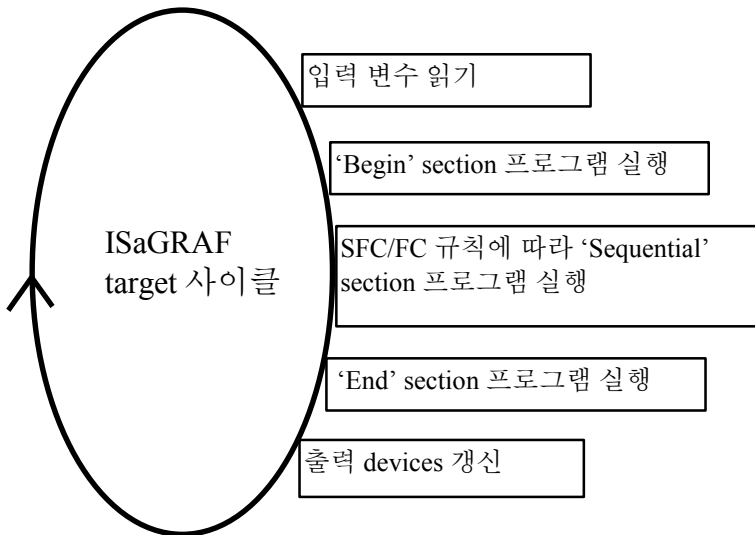
LD 와 FBD 가 동일 프로그램 중에 같이 사용할 수 있다는 것을 제거하면, 동일 프로그램에는 복수 프로그램을 섞이게 할 수 없습니다.

B.1.7 실행 규칙

ISaGRAF 는 동기형시스템(Synchronous System)입니다. 모든 처리는 기본 clock 에 따라 작동됩니다. 이 clock 의 주기를 여기서는 사이클 타임이라 부릅니다. (PLC 에서는 "ScanTime" 이라 부르는 경우도 있지만 **ISaGRAF** 에서는 Scan 에 사용하지 않는 Free 타임을 포함한 기간을 나타냅니다.) :



Basic operations processed during a target cycle are:



이 시스템은 다음을 가능하게 합니다.:

- 사이클 타임중엔 입력변수는 같은 값인 것이 보증 됩니다
- 출력변수는 사이클 타임중엔 1 회만 변경됩니다
- 다른 프로그램에서 글로벌 변수를 안전하게 다룰 수 있습니다
- 어플리케이션 전체 응답시간을 예측 또는 제어할 수 있습니다

B.2 공통 오브젝트

여기서는 ISaGRAF 프로그램 데이터베이스의 **공통 오브젝트**에 관해서 설명 합니다. 이 공통 오브젝트는 **SFC, FBD, LD, ST** 또는 **IL** 언어 중에 사용됩니다.

B.2.1 기본 타입

프로그램에서 사용되는 정수, 변수에는 이하의 기본 타입이 있습니다. 그래픽한 프로그램에서도 언어 base 프로그램에서도 이 타입은 같이 다루어집니다:

이진형(BOOLEAN):	참/거짓 논리값
실.정수형(ANALOG):	정.실수 연속 값
타임형(TIMER):	시변값
메세지형(MESSAGE):	문자열

주의: 타임형 값은 최대가 하루의 길이로 시간을 보존하기 위해 사용할 수 없습니다.

B.2.2 상수 표현

상수는 한 타입만 가질 수 있습니다.

다른 타입으로 같은 표현의 상수는 정의할 수 없습니다.

B.2.2.1 Boolean 이진형 정수

boolean 형 정수에는 이하의 두가지가 있습니다:

TRUE	정수값 '1'과 같다
FALSE	정수값 '0'과 같다

"True" 와 "False" 는 키워드에 등록되어 있고, 대소문자의 구별은 없습니다.

B.2.2.2 Integer analog 정수형

B.2.2.3 정수값은 번호붙인 32Bit 값으로 **-2147483647** 에서 **+2147483647** 의 값을 가집니다. 이하의 기술 방법으로 표현됩니다. Prefix 에 따라 base 를 정하고 있습니다.

base	Prefix	예
DECIMAL	(none)	-908
HEXADECIMAL	"16#"	16#1A2B3C4D
OCTAL	"8#"	8#1756402
BINARY	"2#"	2#1101_0001_0101_1101

BINARY 표현에서 사용되고 있는 언더스코어('_')는 구분에 목적이 있고, 의미는 없습니다. 2 진수를 읽기 쉽게 하기 위한 것입니다

B.2.2.4 Real analog 실수형

실수형 정수표현은 1 0 진수 및 유동 소수점이 표시됩니다. 정수값 과의 차이를 표현하기 위해 소수점('.')을 사용합니다. 유동 소수점 표현은 E 또는 F 를 사용해서 exponent 부분을 표시합니다. exponent 부는 **-37** 에서 **+37** 의 값을 가집니다. 이하에 실수형 정수표현 예를 나타냅니다.

3.14159、	-1.0E+12、
+1.0、	1.0F-15、
-789.56、	+1.0E-37

"123"은 실수형 정수가 아닙니다. 바른 표현 방법은 "123.0"입니다

B.2.2.5 Timer constant 타임형 정수

타임형 정수값은 0 초에서 23 시간 59 분 59 초 999 미리초 (23h59m59s999ms) 까지 정의 가능합니다. 최소단위는 ms 입니다.

Hour "h" 가 시간 뒤에 붙습니다.
Minute "m" 이 분 뒤에 붙습니다.
Second "s"가 초 뒤에 붙습니다.
Millisecond "ms"가 milisec 뒤에 붙습니다.

타임형 정수를 표기할 때는 반드시 **"T#"** 또는 **"TIME#"** prefix 를 첨가할 필요가 있습니다. 대소문자의 구별은 없습니다.

T#1H450MS..... 1 hour, 450 milliseconds
time#1H3M..... 1 hour, 3 minutes

예를 들면 "0"은 타임형 정수는 아니고, 단순한 정수형 정수값이 됩니다.

B.2.2.6 Message string 메시지형

문자열 또는 가변장 문자열형 정수는 반드시 ' ' (모두 반각문자) 로 정리할 필요가 있습니다. ST 프로그램 등에서 사용될 때 따를 필요가 있습니다. 예를 들면

'이것은 가변장 문자열입니다.'

주의: (') 는 가변장 문자열형 정수 안에 포함시킬 수 없습니다. 가변장 문자열형 정수는 소스코드 내에 1 행에 전부 담겨져야 합니다. 최대 255 반각 문자 (스페이스도 포함) 길이입니다.

공란의 가변장 문자열형 정수는 두개의 어프스토로피 (' ') 로 표현됩니다. 터브와 스페이스는 포함되지 않습니다.

" (* 공란의 가변장 문자열형 정수 *)

주의: 모음에데터에서 가변장 문자열형 정수의 초기치를 줄 경우에는 최대장을 지정하고, (') 을 부치지 않고 직접 가변장 문자열형을 씁니다.

인쇄 불가능한 문자를 표현할 경우에 ('\$')심벌을 사용합니다. 이하에서는 몇 개의 예를 나타냅니다.

문자표현	의미	Ascii (hexa)	예
\$\$	'\$' character	16#24	'I paid \$\$5 for this'
\$'		16#27	'Enter '\$Y\$' for YES'
\$L		16#0a	'next \$L line'
\$R		16#0d	' llo \$R He'
\$N	새로운 행	16#0d0a	'This is a line\$N'
\$P	새 페이지	16#0c	'lastline \$P first line'

\$T	터브	16#09	'이름\$Tsize\$Tdate'
\$hh (*)	임의 문자	16#hh	'ABCD = \$41\$42\$43\$44'

(*) "hh" 는 표현하고 싶은 문자 16 진수 ASCII 코드를 나타냅니다

B.2.3 변수

변수는 지역(local)과 전역(global)의 구별이 있습니다. local 은 편집중인 프로그램에서만 참조 가능합니다. global 은 동일 프로젝트의 어느 프로그램에서도 참조 가능합니다.

변수명은 이하의 규칙에 따를 필요가 있습니다.

- 최대 16 반각문자
- 최초의 문자는 (a - z)
- 이후 문자는 문자, 숫자 중에서.

B.2.3.1 예약어

예약어로서는 이하의 것들이 있습니다. 이 예약어들은 프로그램, 변수명, C 언어 Function, C 언어 FunctionBlock 등의 이름에는 사용할 수 없습니다:

A ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
 B 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,
 C CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
 D DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,
 E ELSE, ELIF, 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,
 F FALSE, FEDGE, FIND, FOR, FUNCTION,
 G GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,
 I IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME,
 J JMP, JMPC, JMPCN, JMPN, JMPNC,
 L LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD,
 M MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX,
 N NE, NOT,
 O OF, ON, OPERATE, OR, OR_MASK, ORN,
 P PROGRAM
 R R, REDGE, 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	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,
T	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,
U	UDINT, UINT, ULINT, UNTIL, USINT,
V	VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, VAR_OUTPUT,
W	WHILE, WITH, WORD,
X	XOR, XOR_MASK, XORN

언더스코어 ('_')로 시작되는 키워드는 내부 키워드로 인스트럭션에는 사용할 수 없습니다.

B.2.3.2 직접표현변수

I/O 변수가 접속되어 있지 않은 I/O 채널은 free 채널이라 합니다. ISaGRAF **직접표현 변수**는 이 free 채널을 프로그램 소스 코드 안에서 직접 다룰 수 있습니다. 직접표현변수의 식별자로서 반드시 "%"캐릭터로 시작되는 변수명이 됩니다.

이하에 싱글 I/OBoard 에 대한 직접표현 이름 부치는 법에 대해 나타냅니다. 여기서 "**s**"는 Board 슬롯 번호를 나타냅니다. "**c**"는 I/O 채널 번호를 나타냅니다.

```
%IXs.c.....boolean 형 입력 Board 프리채널
%IDs.c.....정수형 입력 Board 프리채널
%ISs.c.....문자열형 입력 Board 프리채널
%QXs.c.....boolean 형 출력 Board 프리채널
%QDs.c.....정수형 출력 Board 프리채널
%QSSs.c.....문자열형 출력 Board 프리채널
```

이하에 복수 I/OBoard 에서 성립되어 있는 I/O 장치기기에 대한 직접표현 이름 부치는 법에 대해 나타냅니다. 여기서 "**s**"는 Board 슬롯 번호 나타냅니다. "**b**"는 I/O 장치기기 안에서의 싱글 Board 인덱스번호를 나타냅니다. "**c**"는 I/O 채널 번호를 나타냅니다.

```
%IXs.b.c.....boolean 형 입력 Board 프리채널
```

%ID**s.b.c**.....정수형 입력 Board 프리채널
 %IS**s.b.c**.....문자열형 입력 Board 프리채널
 %QX**s.b.c**.....boolean 형 출력 Board 프리채널
 %QD**s.b.c**.....정수형 출력 Board 프리채널
 %QS**s.b.c**.....문자열형 출력 Board 프리채널

이하에 그 예를 나타냅니다.

%QX**1.6** Board 슬롯번호 1, I/O 채널번호 6 인 boolean 형 출력 Board
 %ID**2.1.7** Board 슬롯번호 2 인 I/O 장치기기로 Board 번호가 1, I/O 채널
 번호 7 인 정수형 입력 Board

직접표현 변수는 데이터 타입을 가질 수 없습니다.

B.2.3.3 이진변수

논리값 변수는 **TRUE**, **FALSE** 중에 boolean 형 값을 가질 수 있습니다. boolean 형은
 이하의 속성을 가질 수 있습니다.

내부:.....프로그램에 따라 변경되는 메모리 상의 변수
입력:.....입력 디바이스와 접속되는 변수 (시스템에서 refresh)
출력:.....출력 디바이스와 접속되는 변수

정수: 리드 only 내부변수

주의: boolean 형 변수를 설정할 때는 TRUE, FALSE 문자열을 다른 문자열로 바꿔두기가
 가능합니다. 단, 이 바꿔놓는 문자열은 프로그램에서는 다를 수 없습니다. **정의 워드로**
 설정되어 있는 경우는 이외적으로 다를 수 있습니다

B.2.3.4 아날로그 변수

정수형 변수에는 정수와 실수 두 타입이 있습니다. 각각의 포맷을 이하에서 나타냅니다.

정수	32 bit 부호불인 정수	-2147483647 에서 +2147483647
실수	IEEE 32 bit 유동 소수값 (단정도)	1 부호 bit + 23 유효숫자부 bit + 8 export 부 bit (실수 export 부 -37 ~ +37 사이가

		됩니다.)
--	--	--------

이하의 속성을 가집니다.

내부:.....프로그램에 따라 변경되는 메모리상의 변수
입력:.....입력 디바이스와 접속되는 변수값시스템에서 refresh 값
출력:.....출력 디바이스와 접속되는 변수
정수:.....Read only 내부변수

주의: 실수형 변수가 외부 I/O 디바이스에 접속되는 경우는 I/O 드라이브는 정수형 변수로서 다릅니다.

주의: 정수형과 실수형의 값은 동일 정수형 연산식으로 mix 하여 사용할 수 없습니다

B.2.3.5 시변수

타임형 변수는 타임과 카운터로 사용됩니다. 타임형 변수는 3 2 b i t 을 사용합니다. 타임형 변수는 정수로 최대 2 3 시간 5 9 분 5 9 초 9 9 9 m s 까지 저의 가능합니다.

주의: **타임형 변수의 속성은 내부에서만** 입출력 속성을 지정할 수 없습니다.

타임 값은 I S a G R A F 에 따라 자동적으로 변경됩니다. 타임과 카운터 때는타겟의 realtime clock 에서 읽어낸 신호로 자동 변경됩니다. S T 프로그램 에서는 이하의 스테이트먼트가 사용됩니다.

TSTART 타임 자동 refresh 작동
TSTOP 타임 자동 refresh 를 정지

B.2.3.6 메시지 변수

가변장 문자열형 또는 문자열 변수는 문자열을 포함하고 있습니다. 문자열 변수의 길이는 가변장 입니다. 단, 가변장 문자열형 변수를 모음 에디터로 설정한 때의 최대 길이를 넘을 수 없습니다. 최대 2 5 5 반각 문자까지 가능합니다. 가변장 문자열형 변수는 이하의 속성을 가질 수 있습니다.

내부:.....프로그램에 따라 변경되는 메모리상의 변수

입력:.....입력 디바이스와 접속되는 변수(시스템 refresh)
출력.....출력 디바이스와 접속되는 변수
정수:.....리드 only 내부변수

가변장 문자열형 변수는 ASCII 코드 0 ~ 255 까지 중의 코드를 포함할 수 있습니다. N U L L (0) 캐릭터를 문자열에 포함시키는 것도 가능합니다. 단, CFunction 에 따라 N u l l 를 포함하는 문자열을 처리할 수 없는 경우도 있습니다.

B.2.4 주석

S T 프로그램, I L 프로그램에서는 **커맨드**를 삽입할 수 있습니다. "("로 시작되고, ")"로 종료할 필요가 있습니다. S T 프로그램의 어디라도 커맨드 삽입이 가능합니다. 또한 복수형에 걸친 커맨드도 가능합니다. 이하에 그 예를 나타냅니다.

```
counter := ival; (* assigns the main counter *)
(* 2 행에 걸친 커맨드행 *)
```

```
c := counter (*커맨드는 어디 라도 삽입 가능 합니다. *) +
base_value + 1;
```

커맨드 내부에는 "("문자는 포함할 수 없습니다.

주의: I L 프로그램에서 커맨드 삽입 장소는 인스트럭션 마지막으로 한정되어 있습니다.

B.2.5 예약어

ISaGRAF 에서 정수표현, 이진형 변수인 TRUE, FALSE 표현, 키워드, ST 로 사용되는 복잡한 표현들을 별명으로 재정의할 수 있습니다. 정의의 워드에 따라 바꿔넣기가 가능합니다. 예를 들면

정의 워드	정의 내용
YES	TRUE
PI	3.14159
OK	(auto_mode AND NOT (alarm))

Y E S , P I , O K 는 예를 들면 S T 프로그램 어디에서나 다룰 수 있는 복잡한 표현을 바꿔넣을 수 있습니다. 이하에 S T 프로그램 정의의 워드 (식별자) 를 이용한 예를 나타냅니다.

```
If OK Then
  angle := PI / 2;
  isdone := YES;
```

End_if;

정의 워드는 그 프로그램에서만 참조 가능한 **local 정의**, 동일 프로젝트 내의 어떤 프로그램에서도 참조 가능한 **글로벌 정의**, 어떤 프로그램에서도 참조 가능한 **공통 정의**로 나뉘집니다.

주의: 만약, 동일한 정의 워드에 대해 2회 이상 정의된 경우는 마지막에 적힌 정의가 사용됩니다. 예를 들면

정의 붙이기:	(정의 워드)	(정의 내용)
	OPEN	FALSE
	OPEN	TRUE
와 2회 정의하면, 결국		
의미:	OPEN	TRUE

정의 워드명은 이하 룰에 따를 필요가 있습니다.

- 최대 16 반각 문자
- 최초 문자는 (a - z)
- 이후 문자는 문자, 숫자 중에

주의: 정의에서 정의 워드는 사용할 수 없습니다. 예를 들면 이하의 예는 무효입니다.

PI	3.14159	
PI2	PI*2	: 무효
PI2	6.28318	

B.3 SFC 언어

SFC 언어는 sequential 한 프로세스 operation 을 기술할 때 사용합니다.

프로세스 액션을 표현하는 부분 (스텝) 과 스텝간의 이동조건을 표현하는 부분 (transition) 의 두 가지 그래픽 기호로 이루어져 있습니다.

스텝 안의 액션은 다른 언어 (**ST**, **IL**, **LD**, **FBD**) 로 상세히 기술됩니다.

B.3.1 SFC chart 메인

SFC 프로그램은 **방향**을 가진 **link** 에 따라 접속된 스텝과 transition 구조에 따라 나타납니다.

Multiple link 는 분기와 결합을 나타내는데 사용됩니다.

Main Chart 프로그램의 일부를 Macro Step 이라는 기호로 바꿔 다른 장소에 분리해서 기술할 수 있습니다.

SFC 의 기본적인 규칙은 다음과 같습니다.

- 스텝 후에 따른 스텝을 연결할 수 없습니다.
- transition 후에 다른 transition 을 연결할 수 없습니다.

B.3.2 SFC 기본 구성

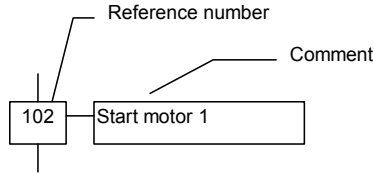
SFC 언어의 기본 구성요소는: steps / initial steps / transitions / oriented links / jump. 입니다

B.3.2.1 Steps / initial steps

스텝은 하나의 정방향에 따라 표현되고, 각 스텝은 정방향 안에 적힌 reference 번호에 따라 참조됩니다.

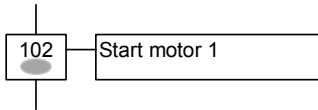
스텝의 주된 내용을 나타내는 커맨드가 스텝 기호에 link 된 장방형 안에 기술됩니다. 이 커맨드는 프로그램언어의 일부분이 아니기 때문에 마음대로 기술 가능합니다.

이러한 표현방식을 스텝 레벨 1 표현이라 합니다.

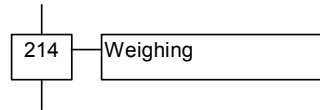


실행 시에는 활성화 상태 스텝에는 token 이 나타납니다.

Active step:

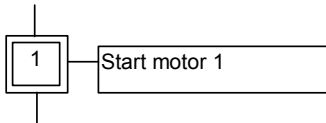


Inactive step:



SFC 프로그램의 초기 상태는 이중테두리로 싸인 Initial Step 기호로 나타냅니다. 프로그램은 자동적으로 각 Initial Step 에서 실행 개시됩니다.

Initial step:



SFC 프로그램은 적어도 하나의 Initial Step 을 포함하고 있어야만 합니다.

아래의 스텝이 가지고 있는 속성 데이터를 나타냅니다.

다른 프로그램 언어에서도 이 데이터들을 사용할 수 있습니다.

GSnnn . x 스텝의 활성화상태(boolean 값)

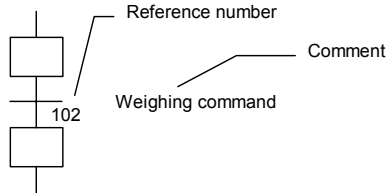
GSnnn . t 스텝 활성화지속 시간(타임 값)

(여기서 n n n 은 스텝 reference 번호)

B.3.2.2 Transitions

transition 은 접속 link 에 교차하는 수평 바에 따라 나타냅니다. 각 transition 은 transition 기호 아래에 적힌 reference 번호에서 참조됩니다.

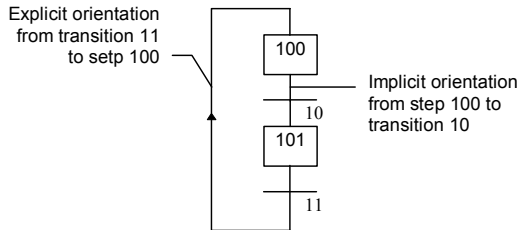
transition 의 주된 내용은 transition 기호 우측에 커맨드로 기술 가능합니다. 이 커맨드는 프로그램 언어의 일부분이 아니고 자유롭게 기술 가능합니다. 이러한 기술법을 transition Level 1 표현이라 합니다.



B.3.2.3 연결

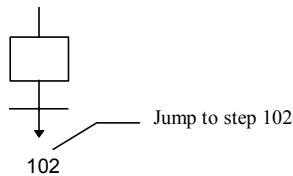
스텝과 transition 을 접속하기 위해 방향을 가진 Link (Oriented Link) 라는 하나의 라인이 이용됩니다.

Link 방향이 명시적으로 주어져 있지 않을때는 위에서 아래로 Link 됩니다.



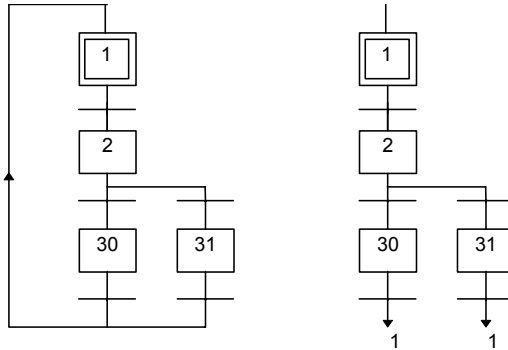
B.3.2.4 점프

transition 에서 스텝에 Link 을 접속할 때에 항상 접속 라인을 그릴 필요 없이 점프 기호를 이용할 수 있습니다. 점프 전에 목적스텝 번호를 기술할 필요가 있습니다.



스텝에서 transition 으로는 점프 기호를 이용할 수 없습니다.

점프의 예 - 아래의 두 가지 차트는 동일합니다.



B.3.3 분기/ 결합

분기란 하나의 SFC 기호(스텝 또는 transition)에서 기타 복수 SFC 기호에 분기하는 다중접속 link 을 의미합니다.

결합이란 복수 SFC 기호에서 하나의 SFC 기호에 모아서 묶어 놓은 다중접속 link 을 의미합니다.

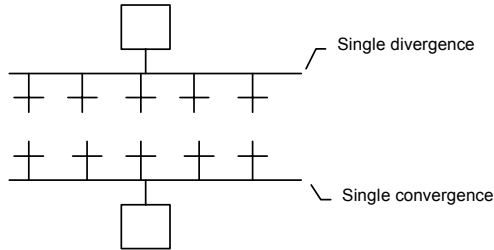
분기와 결합에는 각각 선택 (OR) 과 병렬 (AND) 2 종류가 있습니다.

B.3.3.1 선택 분기

선택 분기(OR)란 하나의 스텝에서 복수 transition 으로 분기로 복수 분기 내의 하나만을 선택하는 것을 허용합니다.

또한, 선택 분기와 복수 transition 에서 하나의 스텝에서의 결합으로 통상의 선택 분기에 따라 시작된 SFC 분기를 모으는데 (결합시키는) 사용됩니다.

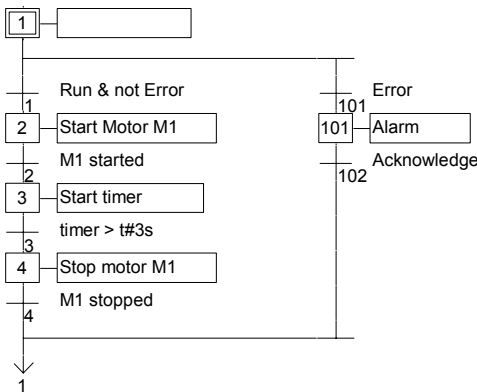
선택 분기와 선택 결합은 하나의 평행 선으로 나타냅니다.



주의 : 선택 분기 직후의 복수 transition 은 암시적인 배타조건에 해당되지는 않습니다. 배타성은 실행시에 복수분기 내의 하나만 선택되는 것을 보증하기위해 transition 조건중에 암시적으로 기술될 필요가 있습니다.

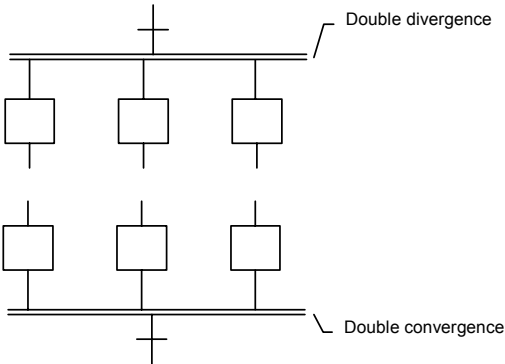
이하에 선택 분기와 선택 결합의 예를 나타냅니다.

(*선택 분기와 선택 결합 SFC 프로그램 *)



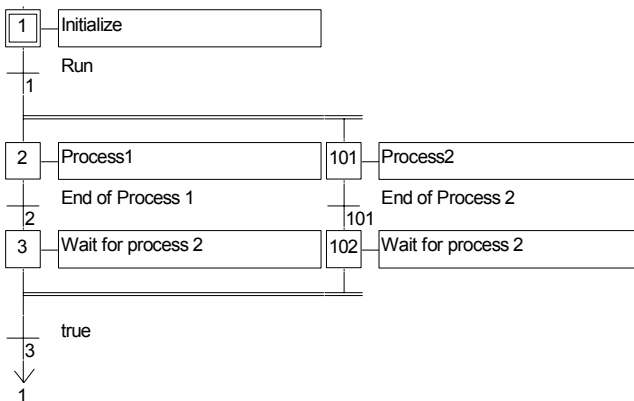
B.3.3.2 병렬 분기

병렬 분기는 하나의 transition 에서 복수 스텝으로의 분기이고, 프로세스의 병행실행을 의미합니다. 또한, 병렬 결합은 복수 스텝에서 하나의 transition 으로 결함입니다. 일반적인 병렬 결합은 병렬 분기에 따라 시작된 SFC 분기를 모으는데 (결합시키는) 사용 됩니다. 병렬 분기와 병렬 결합은 이중 수평선으로 나타냅니다.



병렬 분기와 병렬 결합의 예:

(*병렬 분기와 병렬 결합을 가진 SFC 프로그램 *)



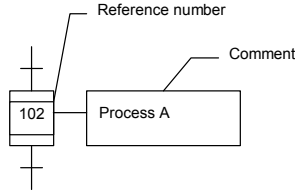
B.3.4 매크로 스텝

Macro 스텝은 복수 스텝과 transition 의 유니크하게 편성된 하나의 서브프로세스를 나타냅니다.

Macro 스텝 body (내용) 는 같은 SFC 프로그램 안의 다른 장소에 독립적으로 기술합니다.

Macro 스텝은 메인 SFC 차트에 한 개의 기호로 나타냅니다.

아래 그림은 Macro 스텝에 사용되는 기호입니다.



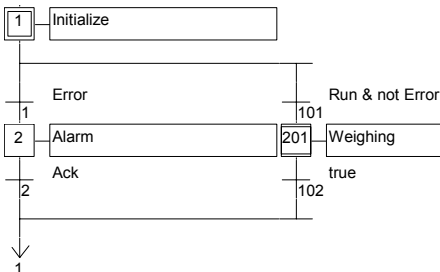
Macro 스텝기호 안에 적힌 reference 번호는 Macro 스텝 본체의 선두 스텝의 reference 번호입니다. Macro 스텝 body 는 개시 스텝으로 시작되어 종료 스텝으로 끝나야 하며, 스스로 완결되어야 합니다.

시작 스텝은 어떠한 위쪽 link (직전의 transition)도 가지지 않고 또한, 종료 스텝은 어떠한 아래쪽 link (직후의 transition)도 가지지 않습니다. Macro 스텝 기호를 다른 Macro 스텝에서 이용할 수 있습니다.

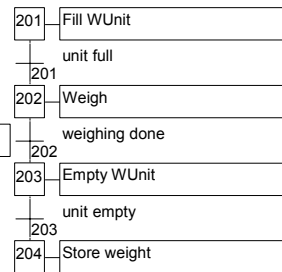
주의:Macro 스텝은 복수 스텝과 transition 의 unique 하게 편성된 하나의 SFC 프로그램에 따라 같은 Macro Step 을 2 회 이상 읽어낼 수 없습니다.

Macro 스텝의 예 : (*Macro 스텝을 갖는 SFC 프로그램 *)

(* Main chart *)



(* Body of the macro step *)



B.3.5 스텝 내의 액션

SFC 스텝의 레벨 2 표현은 그 스텝이 활성상태 사이에 실행해야 할 액션의 상세한 기술 방식입니다.

이 기술은 SFC 의 서식 규칙과 예를 들면 S T와 같은 다른 언어를 이용해서 표현합니다.

하나의 스텝 안에 같은 형 또는 다른 형의 복수 액션을 기술할 수 있습니다.

다른 언어의 사용을 가능 하게 하는 방법은 아래의 두 가지 입니다.

B.3.5.1 Boolean

boolean 액션이란 현재 **스텝의 상태**를 내부 _ 외부의 boolean 변수에 할당하는 것입니다. 그것은 스텝 활동의 시작되거나 멈출때 매회 할당됩니다.

이하에 기본적인 boolean 액션 문법을 나타냅니다. (<>는 입력하지 않습니다.)

< boolean 형 변수명 > (N); 스텝 활동상태를 변수에 할당
< boolean 형 변수명 >; .. (N 속성은 옵션)
/ < boolean 형 변수명 >; 스텝 활동상태의 반전을 변수에 할당

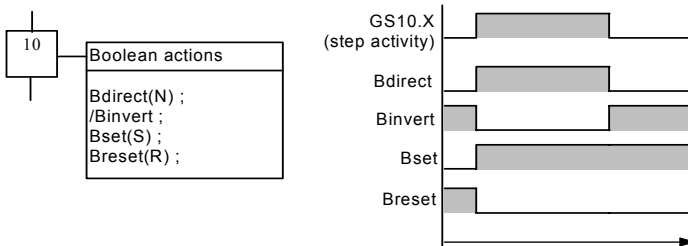
그 밖에 스텝 상태가 활성화된 때의 변수치를 set 또는 reset 할 수도 있습니다.

이하에 이 문법들을 나타냅니다.

< boolean 형 변수명 > (S);
 스텝 상태가 TRUE 가 되었을때 변수치를 TRUE 로 한다.
< boolean 형 변수명 > (R);
 스텝 상태가 TRUE 가 되었을때 변수치를 FALSE 로 한다.

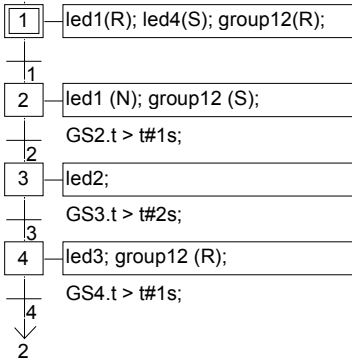
boolean 형 변수는 내부 또는 출력 변수이어야만 합니다.

이하에 나타나는 프로그램 예와 그 행동을 나타냅니다.



boolean 액션의 예 :

(*boolean 액션을 사용한 SFC 프로그램 예 *)

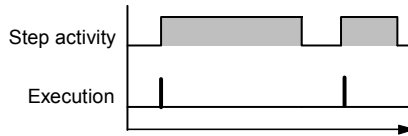


B.3.5.2 ST 언어로된(Pulse)

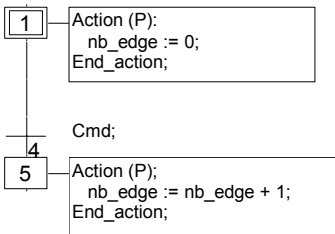
pulse 액션 (P) 는 스텝 상태가 활성화 상태가 된 때 **일회만** 실행되는 **ST 또는 IL** 로 적힌 명령입니다. 다음과 같은 구문으로 적을 필요가 있습니다.

ACTION (P) :
(* S T Instruction *)
END_ACTION ;

이하에 pulse 액션 행동을 나타냅니다.



Example of pulse action:

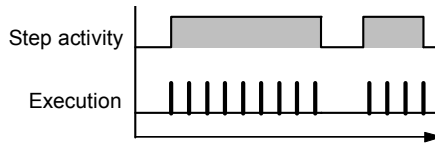


B.3.5.3 ST 언어로된(Non-stored)

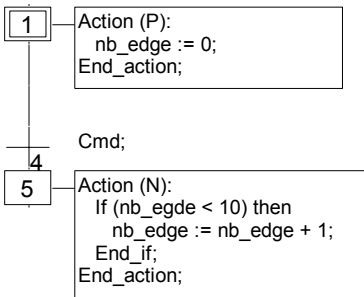
normal 액션은 스텝 상태가 활성화시 **Target Cycle** 실행되는 **S T** 또는 **I L**로 적힌 명령입니다. 다음과 같은 문법으로 사용 합니다.

```
ACTION (N) :
    (* Instruction *)
END_ACTION ;
```

이하에 Normal 액션 행동을 나타냅니다. :



예제



B.3.5.4 SFC actions

SFC 액션에서는 child SFC sequence 을 커맨드 roll 합니다. 즉, 스텝상태에 따라 작동, 정지합니다. SFC 액션에는 **N** (Normal), **S** (Set), **R** (Reset) 식별자가 붙습니다.이하에 문법을 나타냅니다.

< child 프로그램 > (N) ;..

스텝이 활성상태 시에 child sequence 을 작동하고, 비활성 상태시에 child sequence 을 정지(Kill)시킨다.

< child 프로그램 > ; (N 식별자는 옵션)

< child 프로그램 > (S) ;..

스텝이 활성화상태된 때에 child sequence 을 작동하고, 비활성 상태때 child sequence 에 대해 아무 것도 하지 않는다.

< child 프로그램 > (R) ;..

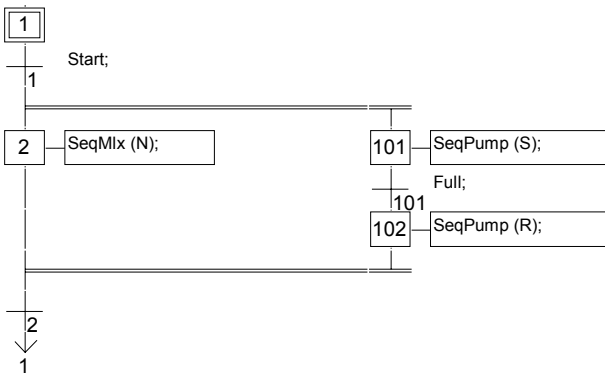
스텝이 활성화상태된 때에 child sequence 을 정지값 kills 값하고, 비활성 상태때 아무 것도 하지 않는다.

액션으로 지정되어 있는 SFC 는 현재 편집중인 프로그램 **child SFC 프로그램**일 필요가 있습니다.

주의: SFC 액션으로 사용되는 **S** (Set), **R** (Reset) 식별자는 S T 언어에 따라 pulse 액션으로 프로그램된 **GSTART**, **GKILL** 스테이트먼트와 같은 의미를 가집니다.

아래는 SFC 액션의 한 예입니다. FatherSFC 프로그램은 SeqMlx 와 SeqPump 라는 두개의 Child-SFC 을 가지고 있습니다.

(* SFCFunction 을 가진 SFC 프로그램 *)



B.3.5.5 서브 프로그램 호출

S T , I L , 또는 F B D 언어로 기술된 서브프로그램, Function, Function 블록 또는 C 언어 Function 과 C 언어 Function 블록은 이하와 같은 구문에 따라 SFC 액션 블록에서 직접 call 가능합니다.

서브프로그램, Function, "C"Function:

```

ACTION (P) :
    result := sub_program ( ) ;
END_ACTION;

```

or

```
ACTION (N) :
    result := sub_program ( ) ;
END_ACTION;
```

"C" 또는 ST, IL, LD, FBD 언어로 기술된 FunctionBlock:

```
ACTION (P) :
    Fbinst(in1, in2);
    result1 := Fbinst.out1;
    result2 := Fbinst.out2;
END_ACTION;
```

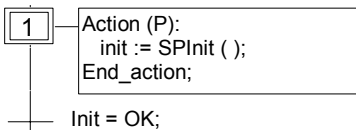
or

```
ACTION (N) :
    Fbinst(in1, in2);
    result1 := Fbinst.out1;
    result2 := Fbinst.out2;
END_ACTION;
```

상세한 문법에 대해서는 S T 언어 섹션을 참조 바랍니다.

이하에서는 FunctionBlock 에서 Sub 프로그램을 call 한 예를 나타냅니다.

(*FunctionBlock 에서 서브프로그램을 읽어낸 S F C 프로그램 *)



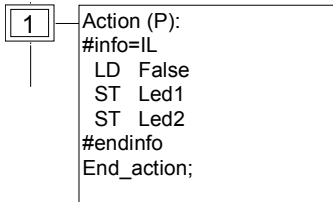
B.3.5.6 IL 언어 표현

아래 구문을 이용해서 SFC 액션 Block 중에 I L 언어로 기술된 프로그램을 직접 끼워 넣을 수 있습니다.

```
ACTION (P) :          (* or N *)
#info=IL
    <instruction>
    <instruction>
    ....
#endinfo
END_ACTION;
```

"#info=IL" 와 "#endinfo"라는 특별한 키워드를 사용할 때는 정확하게 위의 구문을 지키십시오. 키워드 전후에 스페이스와 Tab 을 삽입할 수 없습니다. 이하는 FunctionBlock 중의 IL 프로그램의 예를 나타냅니다.

(* FunctionBlock 에서 IL 프로그램을 읽어낸 SFC 프로그램 예 *)



B.3.6 transitions 조건

각 transition 에는 transition 을 clear (통과) 하기 위한 조건을 규정할 논리식이 attach 됩니다. 통상 이 조건기술에는 S T 언어가 이용됩니다. 이것을 transition 레벨 2 표기라고 합니다. 또한 다른 언어를 사용하는 것도 가능합니다.

ST 언어

LD 언어

IL 언어

transition 에서 서브프로그램 호출

주의: transition 에 조건기술이 없는 경우 디폴트 조건으로서 T R U E 가 설정됩니다.

B.3.6.1 ST 언어

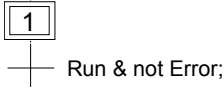
transition 조건을 기술하기 위해 S T 언어를 사용할 수 있습니다. 조건은 boolean 이어야만 하고, 아래 구문 규정에 나타나는 것처럼 세미콜론 으로 끝나야 합니다.

< boolean 형 변수 > ;

조건 기술은 참 / 거짓 상수, 하나의 입력 또는 내부 논리형 (Boolean) 변수 , 또는 boolean 을 도출하는 변수의 구조를 나타냅니다.

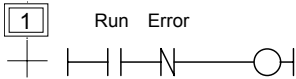
이하에서는 transition 조건을 기술하는 S T 프로그램의 예를 나타냅니다.

(*transition 에 S T 프로그램을 사용한 SFC 프로그램의 예 *)



B.3.6.2 LD 언어

transition 조건을 기술하기 위한 LD 언어를 사용할 수 있습니다. 여기서는 L D 프로그램은 하나의 rung 만 됩니다. 코일에는 변수명을 할당할 필요가 없습니다.
 이하에 transition 조건을 기술하는 LD 프로그램의 예를 나타냅니다.



B.3.6.3 IL 언어

아래의 구문에 따라 transition 을 I L 언어로 직접 기술할 수 있습니다.

```

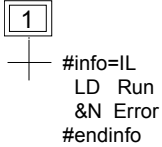
#info=IL
    <instruction>
    <instruction>
    ....
#endinfo
  
```

I L sequence 실행 결과 (I L 레지스터) 에 보존되는 값에 따라 transition 조건 판정 결과가 결정됩니다.

current result = 0	→	FALSE
current result <> 0	→	TRUE

"#info=IL" 와 "#endinfo"라는 특별한 키워드를 사용할 때는 정확하게 위의 구문을 지켜주십시오. 키워드 전후에 스페이스와 Tab 의 삽입은 허용되지 않습니다. 이하는 FunctionBlock 내의 I L 프로그램의 예를 나타냅니다.

(*transition 에 I L 프로그램을 사용한 SFC 프로그램의 예 *)



B.3.6.4 transition 에서 서브프로그램 호출

transition 조건을 평가 판단하기 위해 F B D와 L D , S T , I L로 기술된 서브프로그램을 call 가능합니다.

구문은 아래와 같습니다.

< sub_program > () ;

서브-프로그램 되돌리기 값에 따라 조건 판단이 결정됩니다.

return value = 0	→	FALSE
return value <> 0	→	TRUE

transition 중의 서브프로그램 예

(*transition 에서 서브프로그램을 읽어낸 SFC 프로그램 예 *)



B.3.7 SFC 다이내믹 규칙

SFC 언어의 5 가지 다이내믹 규칙을 아래에 나타냅니다.

 초기 상태

초기 상태는 초기 스텝에서 표현됩니다. 처리 개시 때는 이 스텝은 활성 상태로 됩니다. 각 SFC 프로그램에는 적어도 하나의 초기 스텝이 있어야만 합니다

 transition 통과

transition 에는 enable 상태와 disable 상태가 있습니다. 여기서 enable 상태란 직전 스텝이 활성 (token 이 존재) 상태일 때를 말합니다. 직전 스텝이 활성이 아닐때

disable 상태라고 합니다. transition 은 이하의 조건이 갖춰져 있을때 token 이 통과합니다.

- 직전 스텝이 활성화일 때 동시에
- transition 조건이 TRUE 일때

활성화 스텝의 상태변화

transition 을 token 이 통과하는 동시에 다음 스텝으로 token 이 이동하고서 활성화상태가 됩니다. 그 결과 transition 직전의 스텝은 비활성 상태가 됩니다.

transition 의 동시통과

병렬분기는 token 이동이 동시에 되는 것을 나타냅니다. transition 가 따로 표시되어 있을 때는 직전의 스텝 활성화상태를 조건으로서 해서 사용됩니다.

스텝의 동시 활성화와 비활성화

처리 중에 스텝이 동시에 활성화, 비활성이 되는 경우에 우선 순위는 활성화에 있습니다.

B.3.8 SFC 프로그램 계층

ISaGRAF 에서는 SFC 프로그램의 계층적인 구조를 구축할 수 있습니다. 각 SFC 프로그램은 다른 SFC 프로그램에 대해 작동, 정지 등을 컨트롤할 수 있습니다. 컨트롤 되는 프로그램은 child SFC 프로그램이라 합니다. SFC 프로그램은“ Father - Child” 관계로 계층적으로 link 되어 있습니다.

계층적 구성에는 이하의 기본적인 규칙이 있습니다.

- Father SFC 프로그램을 가지고 있지 않은 SFC 프로그램은 메인 SFC 프로그램이라 합니다.
- 메인 SFC 프로그램이 sequential 섹션으로 처음에 실행됩니다.
- 하나의 SFC 프로그램은 복수 child SFC 프로그램을 가질 수 있습니다.
- child SFC 프로그램은 두 가지 이상의 FatherSFC 프로그램을 가질 수 없습니다. (하나의 Father 프로그램만)
- child SFC 프로그램은 Father SFC 프로그램에서만 컨트롤 됩니다.
- Father SFC 프로그램은 자신의 child SFC 프로그램의 child SFC 프로그램 ("손자" SFC 프로그램) 을 컨트롤할 수 없습니다.

Father SFC 프로그램이 그 child SFC 프로그램에 대해 컨트롤 가능한 액션을 이하에 나타냅니다.

Start	(GSTART) child SFC 프로그램 작동 : child 프로그램의 각 초기 스텝을 활성상태로 합니다. 손자 프로그램은 자동으로 작동되지 않습니다.
Kill	(GKILL) child SFC 프로그램의 정지 : child 프로그램의 활성상태 스텝을 모두 비활성 상태로 합니다. 이때 손자 프로그램도 동시에 정지됩니다.
Freeze	(GFREEZE) child SFC 프로그램 동결 : child 프로그램의 활성 스텝을 동결 상태로 합니다. 단, 이 상태로 메모리에 보관한 후 다시 시작 가능한 상태로 듭니다. 손자 프로그램도 동시에 동결됩니다.
Restart	(GRST) 동결된 SFC 프로그램 재작동 : 동결되어 있던 스텝을 전부 활성화 되면 프로그램을 재작동 시킵니다. 손자 프로그램은 자동으로 재작동 되지않습니다.
Get status	(GSTATUS) child SFC 프로그램의 현재 상황을 취득합니다. (활성 상태, 비활성 상태, 동결 상태 중에)

B.4 Flow Chart 언어

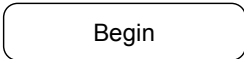
Flow Chart(FC)는 sequential Operation 을 기술할 그래픽컬 언어입니다. Flow Chart 는 Function 과 테스트라는 블록에 성립되어 있고, 이 Block 들 사이를 링으로 접속하고 있습니다. 액션과 테스트의 내용은 ST, IL, LD 중의 언어로 기술 가능합니다. Function 과 FunctionBlock 섹션에서 프로그램을 읽어 내는 것도 가능합니다. Flow Chart 는 다른 Flow Chart 읽어낼 수 있습니다. 이때 읽어낸 Flow Chart 를 FC 서브프로그램이라 합니다.

B.4.1 FC 기본요소

Flow Chart 언어는 이하의 기본요소로 성립되어 있습니다.

FC chart 시작

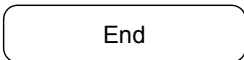
"Begin" symbol 은 Flow Chart 개시에 반드시 필요합니다. Flow Chart 가 작동된 때의 처음 상태 (처음의 사이클) 를 유지합니다. 이하에서는 "Begin" symbol 을 나타냅니다.



"Begin" symbol 은 반드시 하부보다 차트의 다른 오브젝트에 접속됩니다. 만약, 이 접속이 되지 않을 것 같은 Flow Chart 는 실행되지 않습니다.

FC chart 종료

"End" symbol 은 Flow Chart 에 반드시 필요하게 됩니다. "End" symbol 은 통상 차트의 다른 symbol 에서 상부로 접속됩니다만 다른 오브젝트에서 접속되지 않는 경우도 있습니다. 이 경우는 이 Flow Chart 는 항상 loop 처리를 하게 됩니다. 단, 이러한 경우에도 "End" symbol 은 차트 하부에 표시됩니다. "End" symbol 은 차트 처리의 종료를 나타냅니다. 이 symbol 이 처리되는 차트의 종료 상태를 유지하게 됩니다. 이하에서는 "End" symbol 을 나타냅니다.



"End" symbol 은 일반적으로 chart 의 다른 오브젝트에 연결 되 있다. 그러나 FC 는 "End" 오브젝트에 연결 되지 않을 수도 있는데 이때 "End" 오브젝트 는 chart 의 맨 밑에 여전히 나타난다.

links

Flow Link 는 차트 두개 포인트 사이의 처리 흐름을 의미합니다. Flow Link 는 반드시 화살표를 유지합니다. 이하에서는 Flow Link 를 나타냅니다.



두개의 Flow Link 는 동일 소스 오브젝트에서 꼬집어 낼 수 없습니다.

FC 액션

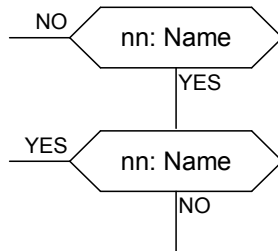
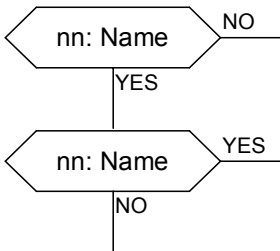
액션 심벌은 실행되는 액션을 나타냅니다. 액션은 번호와 이름으로 식별됩니다. 이하에서는 액션 심벌을 나타냅니다.

nn: Name

차트에서 두개의 다른 액션 심벌은 동일한 이름 또는 번호를 가질 수 없습니다. 액션 프로그램 기술에는 ST, LD, IL 중의 언어로 사용 가능합니다. 액션 심벌은 반드시 하나의 Link 이 들어가고, 하나의 Link 이 나갑니다.

FC 조건

테스트 (조건) 심벌은 Booleam 형의 테스트를 합니다. 테스트 심벌은 번호와 이름에 따라 식별됩니다. 테스트 프로그램은 ST, LD, IL 중의 언어로 기술됩니다. 테스트평가 내용에 따라 Flow 가 YES 또는 NO 패스로 흐릅니다. 이하에 테스트 심벌의 예를 나타냅니다.



두개의 다른 테스트 심벌이 동일 번호 또는 이름을 가질 수 없습니다. 테스트 내용은 아래 중의 하나의 언어로 기술됩니다.

- ST 언어에 따르는 표현식
- 하나의 rung 로 표시되는 LD (이때 코일에 변수 명을 할당할 필요는 없습니다.)
- 복수 행으로 된 IL 언어, IL 레지스터 표현치가 테스트 (조건) 결과에 사용됩니다.

ST 언어로 프로그램된 때는 표현식 마지막에 세미콜론(;) 찍을 수 있지만, 찍지 않아도 에러는 되지 않습니다. 평가 결과와 흐름은 아래와 같이 됩니다.

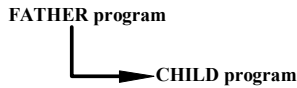
- 0 또는 FALSE : NO

- 1 또는 TRUE : YES

테스트 심벌은 반드시 하나의 Link 가 들어 오고, YES, NO 두개의 Link 가 나갑니다. 두개의 Link 은 반드시 다른 오브젝트에 접속되어야만 합니다.

FC 서브-프로그램

Flow Chart 프로그램에서는 계층적으로 프로그램을 구축할 수 있습니다. 각 Flow Chart 프로그램은 다른 Flow Chart 를 읽어낼 수 있습니다. 읽어낸 프로그램은 Child 프로그램, 또는 **FC 서버프로그램**라고 합니다. FC 서버프로그램을 읽어내는 프로그램을 Father 프로그램이라 합니다. FC 서버프로그램은 아무리 크더라도 하나의 Father 프로그램만을 가질 수 있습니다. FC 프로그램은 이하와 같이 계층구조를 가질 수 있습니다.



Flow Chart 에서의 **서버프로그램** 심벌은 FC 서버프로그램을 읽어 냅니다. FC 서버프로그램이 실행되고 있을 때는 이 처리가 종료할 때까지의 사이에 Father 프로그램의 실행은 support 됩니다. FC 서버프로그램은 번호와 이름에 따라 식별됩니다. 이하에서는 FC 서버프로그램을 읽어내는 서버프로그램 심벌을 나타냅니다.

nn: SpName

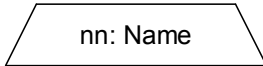
같은 차트에서 두개의 다른 서버프로그램은 동일 번호를 가질 수 없습니다. FC 프로그램 계층구조에선 이하의 룰이 이용됩니다.

- FC 프로그램에서 Father 프로그램을 갖고 있지 않는 FC 프로그램을 메인 FC 프로그램이라 합니다.
- 메인 FC 프로그램은 어플리케이션이 실행된 때에 작동됩니다. (Begin 섹션 실행 후에)
- 프로그램은 복수 FC 서버프로그램을 가질 수 없습니다.
- FC 서버프로그램은 하나의 Father 프로그램만을 가질 수 없습니다.
- FC 서버프로그램은 Father 프로그램으로만 읽어 냅니다.
- FC 서버프로그램끼리의 읽어내기는 불가능합니다.

동일 FC 서버프로그램이 Father 프로그램 차트 안에서 복수회에 걸쳐 읽어내는 일도 있습니다. 서버프로그램 심벌은 차트내의 액션 심벌 같이 Flow Link 접속 룰이 적용됩니다.

FC I/O 액션

FC I/O 조작 Block 은 액션 심벌과 같은 모양으로 다루어집니다. 단, 통상의 액션 심벌과 형태를 다르게 하고, 프로그램 내에서의 처리를 나중에 읽기 쉽게 하기 위해, 구별해서 사용하는 것도 가능합니다. 이하에서는 FC I/O 조작 Block 의 예를 나타냅니다.



I/O 조작 Block 프로그램은 액션 심벌과 같은 모양으로 ST, LD, IL 중의 언어로 서술됩니다.

주의 : I/O 조작이라고 하는 Block 에서는 실행 Platform 에 의존하는 기술을 하는 케이스가 있기 때문에 이러한 명칭이 붙어 있습니다. 특히 이 Block 으로 I/O 처리한다는 것을 의미하지는 않습니다.

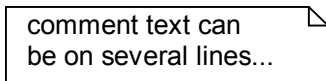
FC 연결

connector 는 차트 내의 두가지 포인트를 직접 follow link 에서 접속하는 대신에취급가능한 link 입니다. connector 는 원형으로 표시되어 흐름의 소스에서 접속됩니다. link 앞은 차트 오브젝트의 번호와 이름에 따라 지정됩니다. 이하에 connector 의 예를 나타냅니다.



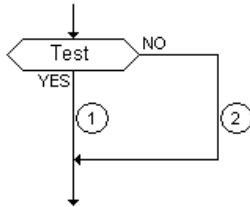
FC 주석

주석 심벌은 follow 차트 실행과는 전혀 관련이 없습니다. 차트 내의 공백 장소에 배치할 수 있습니다. 프로그램 해설에 사용할 수 있습니다. 이하에 커맨드 심벌의 예를 나타냅니다.



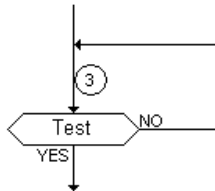
B.4.2 FC 복잡 구조

여기서는 follow 차트의 **편성 구조체**의 예를 나타냅니다. 이들은 기본 오브젝트 (테스트, 액션, link) 로 편성되어 있습니다.



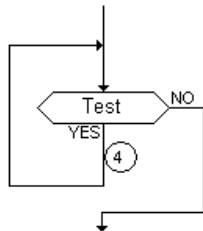
IF / THEN / ELSE

- (1) "THEN" 액션을 삽입합니다.
- (2) "ELSE" 액션을 삽입합니다.



REPEAT / UNTIL

- (3) repeat 할 액션을 삽입합니다.



WHILE / DO

- (4) repeat 할 액션을 삽입합니다.

B.4.3 FC 문법검사

레벨 2 프로그램을 기술하고 있는 ST, LD, IL 프로그램 문법 체크와는 별도로 이하와 같은 follow 차트 자체의 **문법 체크**가 있습니다.

메인 문법 체크 :

- 모든 심벌에 있어 접속 포인트는 반드시 follow link 에서 접속되어야만 합니다. (단, "End" 심벌만은 이 접속을 생략할 수 있습니다.)
- 기타 심벌과 독립한 심벌과 차트는 존재할 수 없습니다.
- Connector 심벌에서 접속 전에 있는 오브젝트가 바르게 지정되어야만 합니다.

기타 룰 :

- 공백 액션 (즉, 레벨 2 프로그램이 존재하지 않는 액션 심벌) 은 실행시에는 아무것도 처리되지 않습니다.

공백 테스트 (즉, 레벨 2 프로그램이 존재하지 않는 테스트 심벌) 는 실행시에는 항상 TRUE 로 간주됩니다.

follow 차트 타겟 상에서의 **실행 룰**은 이하와 같이 설명할 수 있습니다.

- Begin 심벌은 타겟 상에서 처음의 1 사이클을 사용합니다.
- End 심벌은 타겟 상에서 마지막 1 사이클을 사용하고, 차트 실행을 종료합니다. 이 심벌에는 도달한 다음 타겟 상에서의 사이클에서는 차트 상의 어떠한 액션도 실행되지 않습니다.
- Begin 와 End 심벌 중간 타겟 상에서 사이클에서는 테스트와 액션 심벌이 실행되지만, 1 사이클 내에 실행되어 처리되는 것은 SFC 과 달리 하나의 테스트와 액션 Block 에 한정된 것은 아닙니다. 처리 판단이 성립되어 있는 한 실행이 계속되고, 한번 실행한 처리를 다시 반복하려면 loop 를 검출하면 사이클을 뺏아 내게 됩니다. 구체적으로 SFC 로 한 스텝 내에서 기술되어야만 처리에서도 FC 는 복수 테스트와 액션 심벌로 기술 가능하게 됩니다.
- 실행되고 있는 테스트와 액션심벌은 시뮬레이션과 debug 중에는 SFC 경우와 같이 색깔이 있어 다른 심벌과 구별할 수 있습니다.

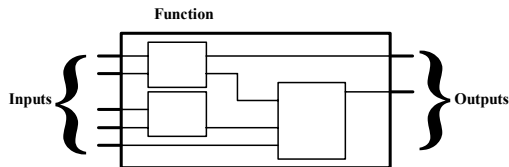
주의: SFC 가 상태 이동을 표현하는 것에 대해 FC 는 판단 follow 을 표현하게 됩니다. 프로그램시에 실행 룰 (algorithm) 을 고려한 후에 프로그램하는 것이 필요합니다. ISaGRAF 샘플 프로젝트로 SFC, FC 를 비교하고 각각의 특징을 살려서 프로그램을 행하는 것이 필요합니다.

B.5 FBD 언어

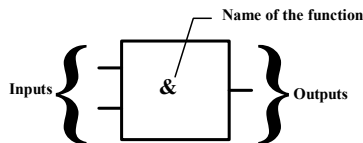
Functional Block Diagram (FBD)은 각종 타입의 장방형 (상자) 으로 나타낸 평선블록이 접속된 그래픽표현 방법의 하나입니다. 상자 좌측에는 평선의 입력이, 우측에는 평선의 출력이 접속됩니다. 변수가 **논리적인 링**을 가진 상자에 접속됩니다. 출력은 다른 평선의 입력에 접속되는 일도 있습니다.

B.5.1 FBD 형식

FBD 프로그램의 입력은 function block 의 입력 연결에 설정 되어야 합니다. 입력변수의 타입은 관련된 타입과 동일한 타입 이어야 합니다. FBD 입력타입은 다음과 같다.(상수, 내부, 입력, 출력)



FBD 프로그램의 출력은 function block 의 출력 연결에 설정 되어야 합니다. 입력변수의 타입은 관련된 타입과 동일한 타입 이어야 합니다. FBD 는 모든 **내부**, 출력 변수, 프로그래밍 (단,**sub-programs**)이 사용 가능 합니다.출력연결이 현재 편집중인 sub-program 인경우 ,sub-program (호출한 program 으로 값 반환)의 반환값을 할당 하게 됩니다.



function blocks 의 입력, 출력 변수는 **연결라인** 으로 연결 되어 있습니다.

- 입력변수는 function block 의 입력측

- 출력변수는 다른 function block 의 입력단 으로 연결됨
- 출력변수는 function block 의 출력측

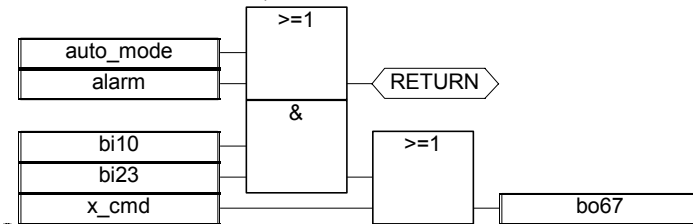
관련 데이터를 좌측 끝에서 우측 끝으로 이동한다는 의미로 좌/우의 타입은 동일하여야 합니다.
다중 우측 접속역시 마찬가지이다.

모든 연결의 타입은 동일 하여야 합니다

B.5.2 RETURN

"<리턴>" 은 출력측에 사용하게 됩니다. 반듯이 function block 의 이진 출력측에 연결되어야 합니다. "리턴" 명령은 조건부 마침인데, 예를 들어 **TRUE** 값이 설정 되어 있는 경우라면 나머지 부분은 실행이 되지 않습니다.

(* FBD 내에서 "리턴" 명령의 사용 예 *)



(* ST equivalence: *)

```

If auto_mode OR alarm Then
    Return;

```

```

End_if;

```

```

bo67 := (bi10 AND bi23) OR x_cmd;

```

B.5.3 점프 / 라벨

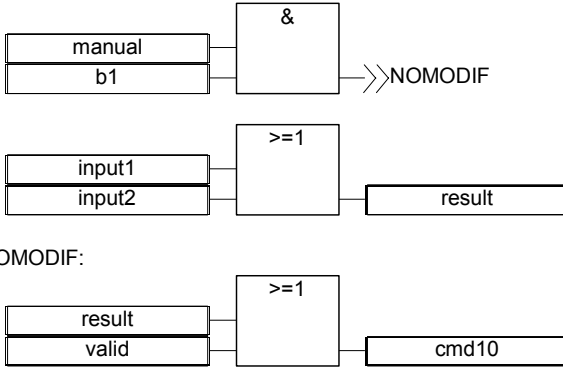
점프/라벨은 다이어그램의 실행을 제어 하는 수단으로 사용 됩니다. 다음명령어를 참고 하세요:

>>LAB.....라벨명으로 점프 (라벨명 "LAB")

LAB:.....라벨 정의 (라벨명 "LAB")

연결선 좌측의 이진 값이 TRUE 인경우 프로그램 실행은 상응 하는 라벨로 직접 점프 합니다

(* 라벨/점프 를 사용한 FBD 프로그램의 예 *)



(* IL Equivalence: *)

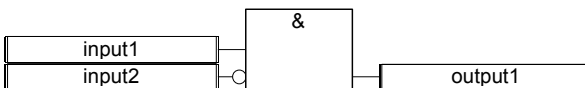
	ld	manual
	and	b1
	jmpc	NOMODIF
	ld	input1
	or	input2
	st	result
NOMODIF:	ld	result
	or	valid
	st	cmd10

B.5.4 이진 반전

입력측에 단일 연결은 function block 에서 이진 반전이 가능 합니다

작은 원 모양으로 표시되는데, 이진 반전이 사용된 경우 좌/우 타입은 이진 타입으로 설정 되어야 합니다.

(*FBD 에서 라벨/점프 사용 예 *)



(* ST equivalence: *)

output1 := input1 AND NOT (input2);

(* ST equivalence: *)

output1 := input1 AND NOT (input2);

B.5.5 FBD 에서 function / function block 호출

FBD 언어는 sub-프로그램, function, function block 의 호출이 가능 합니다. sub-프로그램, function, function block 은 function box 로 나타 나는데, Box 에 적힌 이름은 sub-프로그램, function, function block 의 이름 입니다. sub-프로그램, function 의 경우엔 반환값은 function box 의 출력 값 입니다.

function block 은 하나 이상의 출력을 갖습니다..

```
(* ST Equivalence *)  
net_weight := Weighing (mode, delta); (* call sub-program *)  
If (net_weight = 0) Then Return; End_if;  
weight := net_weight + tare_weight;
```

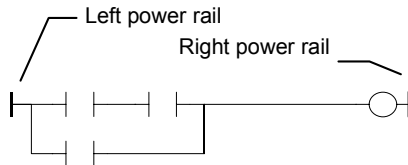
B.6 LD 언어

LD는 boolean 연산의 그래픽한 표현방식 입니다. 입력 (접점, 또는 **콘택트**) 과 출력 (**코일**) 의 편성으로 이루어져 있습니다. LD 다이어그램은 전기 relay 회선과 함께 다룰 수 있습니다. LD 다이어그램은 좌우 수직 **모선**으로 접속되어야 합니다.

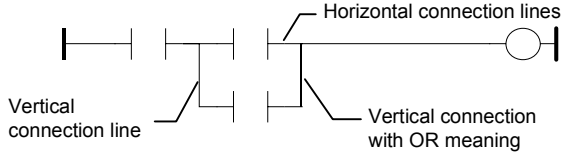


B.6.1 모선 / 접속선

LD 다이어그램은 좌우 수직 라인 즉, 좌/우 모선에 접속되어야 합니다



LD 다이어그램에서의 그래픽 심벌은 접속선 (connection 라인) 으로 모선 또는 다른 심벌에 접속됩니다. 접속선은 수직 또는 수평 라인입니다.



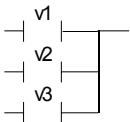
각 라인 Segment 는 boolean 상태의 **FALSE** 또는 **TRUE** 의 값을 가집니다. 이 상태는 직접 접속되어 있는 Segment 는 같은 값을 가집니다. 수직인 좌우 모선에 접속된 수평라인은 **TRUE** 상태를 가집니다.

B.6.2 다중 접속

싱글 수평라인은 좌우 하나의 라인으로 되어 있지만, 수평라인과 수평라인을 편성해서 복수라인을 모을 수 있습니다. (**multiple 라인**) multiple 라인에 따라 boolean 상태는 이하의 룰을 가집니다.

좌측에 multiple 라인을 가질 경우, 즉, 복수 수평 라인이 좌 모선에서 접속되어 심벌을 자유롭게 우측에서 하나의 수평라인으로 접속되는 경우는 우측의 boolean 상태는 좌측 라인의 **논리 OR**을 가지게 됩니다.

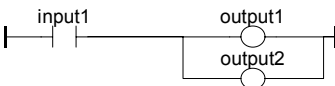
(* 좌측의 multiple 라인 *)



(* 우측의 상태는 (v1 OR v2 OR v3) *)

우측 multiple connction 의 경우는 즉, **좌측의 하나의 수평라인**이 하나 이상의 라인이 되어 **우측**에 접속되는 경우는 좌측 논리상태가 우측으로 전달됩니다.

(* 우측 multiple 라인 *)

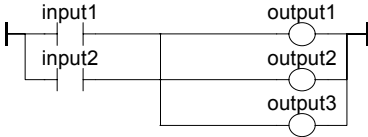


(* 등가한 S T 언어 *)

output1 := input1;
output2 := input1;

좌우 multiple 라인의 경우는 즉, 좌우측이 복수 라인으로 좌우 수직라인에 접속되는 경우, 우측 논리 상태는 좌측 복수 라인의 논리 OR 이 됩니다.

(* 좌우 multiple 라인 예 *)



(* 등가한 S T 언어 *)

output1 := input1 OR input2;

output2 := input1 OR input2;

output3 := input1 OR input2;

B.6.3 기본 접점과 코일

입력 접점 (콘택트) 에는 몇 개의 종류가 있습니다.

a 접점 (Direct contact)

b 접점 (Inverted contact)

오르기 접점

내리기 접점

출력 코일에는 몇 개의 종류가 있습니다.

코일

반전 코일

SET 코일

RESET 코일

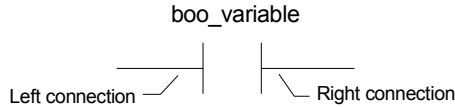
오르기 검출 코일

내리기 검출 코일

boolean 변수명은 이 그래픽 심벌 위에 적힙니다.

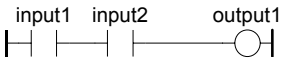
☐ a 접점

a 접점은 **boolean 형 변수와 접속선 사이**에서 boolean 을 조작합니다.



컨택트 우측 접속선의 상태는 컨택트 좌측 접속선과 컨택트에 할당된 변수와의 논리적 AND의 결과입니다.

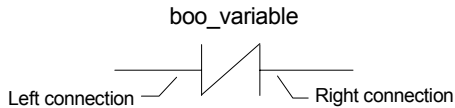
(* a 접점의 예 *)



(* ST Equivalence: *)
output1 := input1 AND input2;

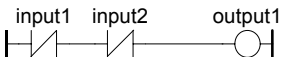
□ b 접점

b 접점은 접속선과 boolean 형 변수의 반전과 boolean 을 조작합니다.



컨택트의 우측 접속선의 상태는 컨택트 좌측 접속선과 컨택트에 할당된 변수의 반전과의 논리적인 AND의 결과입니다.

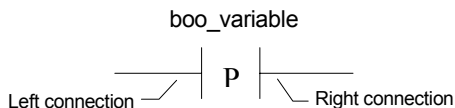
(* b 접점의 예 *)



(* ST Equivalence: *)
output1 := NOT (input1) AND NOT (input2);

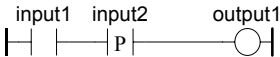
□ 오르기 접점

오르기 접점은 접속선과 논리 변수의 오르기와 boolean 을 조작합니다.



컨택트의 우측 접속선은 좌측 접속선이 **TRUE**인 동시에 boolean 형 변수가 **FALSE**에서 **TRUE**로 변화한 때만 **TURE**가 됩니다. 이외의 상태에서는 **FALSE**가 됩니다.

(*오르기 접점의 예 *)



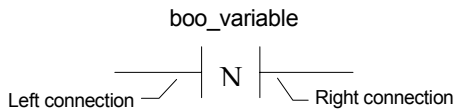
(* ST Equivalence: *)

output1 := input1 AND (input2 AND NOT (input2prev));

(* input2prev is the value of input2 at the previous cycle *)

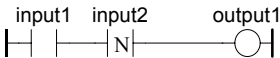
내리기 접점

내리기 접점은 접속선과 논리 변수의 내리기와 boolean 을 조작합니다.



컨택트의 우측 접속선은 좌측 접속선이 **TRUE**인 동시에 boolean 형 변수가 **TRUE**에서 **FALSE**로 변화한 때만 **TURE**가 됩니다. 이외의 상태에서는 **FALSE**가 됩니다.

(*내리기 접점의 예 *)



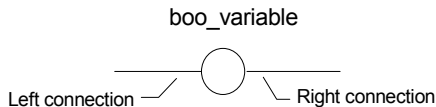
(* ST Equivalence: *)

output1 := input1 AND (NOT (input2) AND input2prev);

(* input2prev is the value of input2 at the previous cycle *)

코일

코일은 접속선 boolean 상태를 boolean 입력합니다.

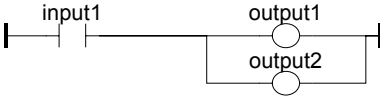


좌측 접속선 상태가 코일에 할당된 논리 출력변수에 할당됩니다. 좌측 접속선 상태는 그대로 우측 접속선으로 전달됩니다. 우측 접속선은 통상 우 모선에 접속됩니다.

코일에 할당된 논리 출력변수는 **내부변수** 또는 외부 **출력 변수**이어야 합니다.

변수명은 프로그램명 (단, 서버프로그램명만) 을 사용하는 것도 가능합니다. 이 경우 변수명은 서버프로그램의 출력 parameter 에 일치해야 합니다.

(* 코일의 예 *)



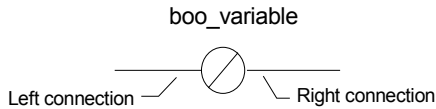
(* ST Equivalence: *)

output1 := input1;

output2 := input1;

반전 코일

반전 코일은 접속선 boolean 상태의 반전을 boolean 출력합니다.

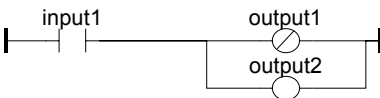


좌측 접속선의 반전 상태가 코일에 할당된 논리 출력 변수에 할당됩니다. 좌측 접속선의 반전 상태는 그대로 우측 접속선에 전달됩니다. 우측 접속선은 통상 우 모선에 접속됩니다.

코일에 할당되는 논리 출력 변수는 **내부 변수** 또는 외부 **출력 변수**이어야 합니다.

변수명은 프로그램명 (단, 서버프로그램명만) 을 사용하는 것도 가능합니다. 이 경우 변수명은 서버프로그램의 출력 parameter 에 일치해야 합니다.

(*반전 코일의 예 *)



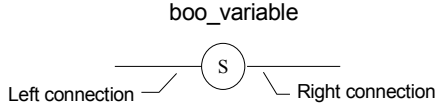
(* ST Equivalence: *)

output1 := NOT (input1);

output2 := input1;

SET 코일

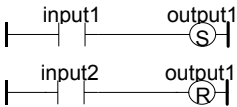
set 코일은 접속선 boolean 상태를 boolean 출력합니다.



할당된 출력변수는 좌측 접속선 상태가 TRUE 가 된때에 **TRUE**에 **set** 됩니다. 출력변수는 이 상태를 reset 코일에서 반전 될 때 까지 계속합니다. 좌측 접속선 상태는 그대로 우측 접속선에 전달됩니다. 우측 접속선은 통상 우 모선에 접속됩니다.

할당 되는 변수는 내부 변수 또는 외부 출력 변수이어야 합니다.

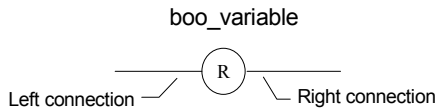
(* set, reset 의 예 *)



(* ST Equivalence: *)

```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

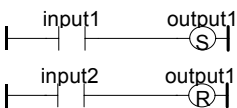
RESET 코일



할당된 출력변수는 좌측 접속선 상태가 TRUE 가 되었을때 **FALSE**에 **reset** 됩니다. 출력변수는 이 상태를 reset 코일에서 반전 될 때 까지 계속합니다. 좌측 접속선 상태는 그대로 우측 접속선에 전달됩니다. 우측 접속선은 통상 우 모선에 접속됩니다.

할당 되는 변수는 내부 변수 또는 외부 출력 변수이어야 합니다.

(* set, reset 의 예 *)

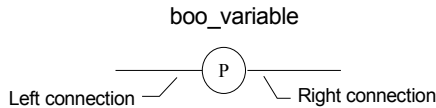


```
(* ST Equivalence: *)
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

☐ 오르기 검출 코일

오르기 검출 코일은 접속선 boolean 를 출력합니다.

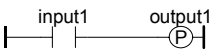
이 코일은 Quick LD 에디터를 사용한 경우에만 유효합니다.



할당된 변수는 좌측 접속선의 상태가 FALSE 에서 TRUE로 올라 갔을 때 TRUE 에 **set** 됩니다. 출력변수는 FALSE 에 reset 됩니다. 좌측 접속선 상태는 그대로 우측 접속선에 전달됩니다. 우측 접속선은 통상 우 모선에 접속됩니다.

할당 되는 변수는 내부 변수 또는 외부 출력 변수이어야 합니다.

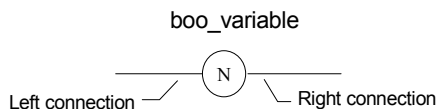
(* 오르기 검출 코일의 예 *)



```
(* ST Equivalence: *)
IF (input1 and NOT(input1prev)) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev is the value of input1 at the previous cycle *)
```

☐ 내리기 검출 코일

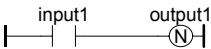
내리기 검출 코일은 접속선 boolean 를 출력합니다. 이 코일은 Quick LD 에디터를 사용한 경우에만 유효합니다.



할당된 변수는 좌측 접속선의 상태가 TRUE 에서 FALSE 로 내려 갔을 때에 TRUE 에 **set** 됩니다. 출력 변수는 FALSE 에 reset 됩니다. 좌측 접속선 상태는 그대로 우측 접속선에 전달됩니다. 우측 접속선은 통상 우 모선에 접속됩니다.

할당 되는 변수는 내부 변수 또는 외부 출력 변수이어야 합니다.

(*내리기 검출 코일*)



(* ST Equivalence: *)

IF (NOT(input1) and input1prev) THEN

output1 := TRUE;

ELSE

output1 := FALSE;

END_IF;

(* input1prev is the value of input1 at the previous cycle *)

B.6.4 RETURN 기술법

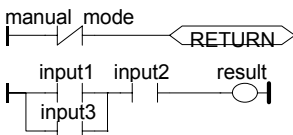
B.6.5 RETURN 라벨은 조건 종료에 사용됩니다. RETURN 의 우측에는 아무것도 접속되지 않습니다.



RETURN 라벨 좌측 라인의 논리 상태가 TRUE 일 때 프로그램은 이하의 프로그램을 실행하지 않고 RETURN 합니다.

주의 : L D 프로그램이 서버프로그램일 때는 프로그램명과 같은 이름을 RETURN 값에서 set 해서 Father 프로그램에 RETURN 해야 합니다.

(* RETURN 심벌의 예 *)



(* ST Equivalence: *)

```
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;
```

B.6.6 점프 / 라벨

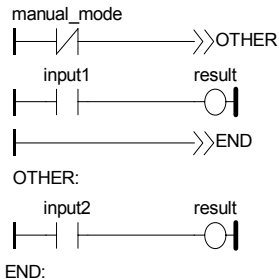
라벨과 조건 부치기 (조건 없음) 점프를 사용해서 다이어그램의 실행 제어가 가능합니다.
라벨과 점프 심볼의 우측에 라인 접속은 할 수 없습니다.

>>LAB.....라벨명 "LAB"로 점프

LAB:.....라벨명 "LAB"의 내용

점프 심벌 좌측의 논리 상태가 TRUE 일 때 프로그램은 대응한 라벨 심벌로 점프한 후에 실행됩니다.

(*점프와 라벨의 예*)



(* IL Equivalence: *)

	ldn	manual_mode
	jmpc	other
	ld	input1
	st	result
	jmp	END
OTHER:	ld	input2
	st	result
END:	(* end of program *)	

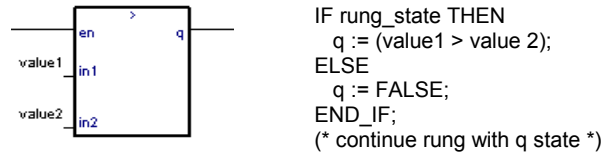
B.6.7 LD 내의 Blocks

QuickLD 에디터를 사용해서 Function (Block) 을 라인으로 접속할 수 있습니다. 여기서 Function 은 연산자, FunctionBlock, Function 을 가리킵니다. 모든 Block 이 반드시 boolean 형 입출력을 가지고 있는 것을 제한하지 않기 위해, 새로운 EN, ENOparameter 를 입출력

interface 에 추가합니다. 단, F B D / L D 에디터에서는 필요한 형태의 변수를 직접 접속 가능케 하기 위해, E N, E N O 는 표현되지 않습니다.

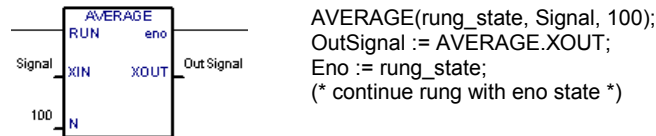
☐ "EN" 입력

연산자, FunctionBlock, Function 최초의 parameter 가 boolean 형이 아닐 경우는 자동적으로 **E N**이라는 입력 parameter 가 삽입됩니다. 이 **E N**가 TRUE 일때만 Block 이 실행됩니다. 이하에 비교 명령의 예와 등가한 S T 언어 표현을 나타냅니다.



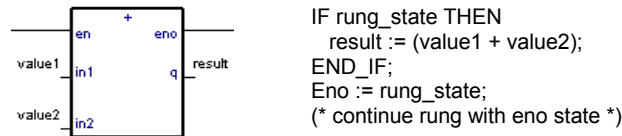
☐ "ENO" 출력

연산자, FunctionBlock, Function 최초의 parameter 가 boolean 형이 아닐 경우는 자동적으로 **E N O**라는 입력 parameter 가 삽입됩니다. 이 **E N O**는 항상 최초의 입력 parameter 값을 가집니다. 이하에 평균수명의 예와 등가한 S T 언어 표현을 나타냅니다.



☐ "EN" / "ENO" 파라미터

어떤 경우는 **E N**, **E N O** 둘 다 필요한 경우가 있습니다. 이하에 산술연산의 예와 등가한 S T 언어 표현을 나타냅니다.



B.7 ST 언어

ST (**Structured Text**)는 High Level 의 구조화 언어 입니다. 이 언어는 주로 그래픽 언어로 충분히 표현하기 어려운 수속을 나타내는데 적용됩니다. ST는 **S F C**언어의 transition 과 스텝 내의 Function 기술 디폴트 언어입니다.

B.7.1 ST 문법

ST 프로그램은 ST스테이트먼트 리스트입니다. 각 스테이트먼트의 마지막에 세미콜론 (;) separator 가 필요합니다. 소스 코드중의 변수는 inactive separator (스페이스, Tab 등) 와 active separator (: =, <, > 등) 으로 구분됩니다. 커맨드는 (* *)로 에워싸입니다.

기본 ST statement 는 이하의 것이 있습니다.

- 스테이트먼트 assign (variable := expression;)
- 서브프로그램, Function 호출
- C언어 FunctionBlock 호출
- 선택 스테이트먼트 (IF, THEN, ELSE, CASE...)
- 반복 스테이트먼트 (FOR, WHILE, REPEAT...)
- 컨트롤 스테이트먼트 (RETURN, EXIT...)
- S F C 등 기타 언어와 link 용의 특수 statement

Inactive separator 는 Active separator, 정수, 식별자 사이에 있으면 자유롭게 삽입 가능합니다. Inactive separator 는 스페이스, Tab, End of Line 캐릭터입니다. 이하의 룰을 지켜 Inactive separator 를 사용하면 읽기 쉬운 ST 프로그램을 만들 수 있습니다.

- 한 행에 하나 이상의 스테이트먼트는 적을 수 없습니다.
- 복잡한 스테이트먼트는 Tab 키를 사용해서 intend 합니다.
- 커맨드를 가능한 삽입합니다.

B.7.2 표현법과 괄호

S T 표현은 S T 조작자와 변수, 정수를 편성해서 행합니다. 하나의 표현에 대해서 operand 타입은 같아야 합니다. 이들의 편성에 따라 복잡한 표현을 기술합니다. 예를 들면,

(boo_var1 AND boo_var2)	논리형
not (boo_var1)	논리형
(sin (3.14) + 0.72)	실수형
(t#1s23 + 1.78)	부정환 표현값

괄호 표현은 전체 표현 가운데 어떤 부분을 독립시키는 것으로, 조작의 우선 순위를 높입니다. 괄호가 없는 표현에서의 조작은 S T 조작자 간의 암시적인 우선 순위에 따릅니다. 예를 들면,

$2 + 3 * 6$	equals $2+18=20$	승산은 가산보다 우선 순위가 높다.
$(2+3) * 6$	equals $5*6=30$	괄호에 따라 우선 순위를 변경

주의 : 하나의 S T 표현 내에 최대 8 레벨까지 괄호가 인정됩니다.

B.7.3 Function / function block 호출

표준적인 S T에 따라 Function 호출로 다음의 Function 을 호출할 수 있습니다.

- 프로그램
- I E C 언어로 적힌 library Function
- C 언어 Function, C 언어 FunctionBlock
- 형 변환 함수

≡ 서브프로그램(sub-programs) / function 호출

이름: 서브프로그램명 또는 IEC 와 C 언어로 적힌 library Function

의미: ST, IL, LD, FBD 로 적힌 서브프로그램과 Function 과 언어의 호출, 되돌리기 값을 얻는다.

문법: <variable> := <subprog> (<par1>, ... <parN>);

operand : 되돌리기 값과 인수 타입은 서브프로그램에서 정의된 파라미터정의에 따라야 합니다.

되돌리기 값

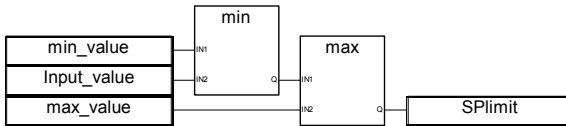
서브프로그램 호출은 각종 표현으로 호출됩니다. SFC 프로그램 transition 내부에도 사용됩니다.

예 1 : 서브프로그램 호출

Example1: Sub-program call

```
(* Main ST program *)
(* gets an analog value and converts it into a limited time value *)
ana_timeprog := SPLimit ( tprog_cmd );
appl_timer := tmr (ana_timeprog * 100);
```

(* Called FBD program 이름 d 'SPLimit' *)



Example2: Function call

```
(* functions used in complex expressions: min, max, right, mlen and left are standard "C"
functions *)
limited_value := min (16, max (0, input_value) );
rol_msg := right (message, mlen (message) - 1) + left (message, 1);
```

function blocks 호출

이름: FunctionBlock instance 명

의미: ISaGRAF library 에 등록되어 있는 FunctionBlock 을 호출해서 출력 파라미터를 사용한다.

문법: (* 호출 of the function block *)

<block 이름> (<p1>, <p2> ...);

(gets its return 파라미터 s *)

<result> := <block 이름>. <ret_param1>;

...

<result> := <block 이름>. <ret_paramN>;

operand: 파라미터타입은 FunctionBlock 에서 정의되고 있는 타입과 일치해야 합니다.

되돌리기 값: 문법 참조

Function 의 의미, 파라미터형, 되돌리기 값형 등은 I S a G R A F library 에 등록되어 있는 것을 참조.

FunctionBlock instance 는 반드시 모음으로 등록해 둡니다.

예 :

(*FunctionBlock 을 호출하는 S T 프로그램의 예 *)

(*FunctionBlock instance 를 모음으로 설정해 놓습니다. *)

(* trigb1 : block R_TRIG – 오르기 검출*)

(* S T 언어에서 읽어내기 *)

If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;

B.7.4 ST 특수 이진 명령어

다음의 boolean 명령은 특수한 ST 언어입니다.

REDGE 오르기 검출

FEDGE 내리기 검출

기타 기본 boolean 명령을 이하에 나타냅니다.

- **NOT** 논리 반전
- **AND (&)** AND (논리적)
- **OR** OR (논리화)
- **XOR** XOR (배타적 논리화)

☐ "REDGE" 명령

0 | REDGE**의미** boolean 형 변수의 오르기 검출

```
<edge> := REDGE ( <boo_expression>,<memo_variable> );
```

operand	<boo_expression>	오르기	검출	하는	boolean	형
	변수<memo_variable>	직전의	변수	상태를	스토어하기	위한
		내부변수				

되돌리기 값 TRUE / FALSE 에서 TRUE 로 변화한 때
FALSE 위와 다른 경우

REDGE() 을 사용한 오르기 검출은 1 회 실행 Scan 에 2 회 이상은 검출되지 않습니다. 오르기 검출이 S F C transition 조건에 사용되는 경우는 있습니다.

주의 : <memo_variable> 변수는 변수 직전의 상태를 보관하기 위해 사용되는 것으로, 다른 오르기 검출 등을 목적으로 동시 사용은 불가능합니다.

통상, 오르기 검출을 하고 싶은 변수명이 "xxx"일때, <memo_variable>로서는 "EDGE_xxx"을 사용합니다. 따라서 다른 REDGE 조작에 의해 표시되는 일은 없습니다.

예 :

(* ST program using REDGE operator *)

(* this program counts the rising edges of a boolean input *)

(* Bi120 is an input boolean variable *)

(* Edge_Bi120 is the memory of the Bi120 variable state *)

If REDGE (Bi120, Edge_Bi120) Then

Counter := Counter + 1;

End_if;

주의 : 이 명령은 IEC1131-3 에 포함되어 있지 않습니다. 호환성을 위해서는 R_TRIG 표준 Block 의 이용을 권합니다.

≡ "FEDGE" 명령

의미 boolean 형 변수의 오르기 검출

문법 <edge> := FEDGE (<boo_expression>, <memo_variable>);

operand <boo_expression> 오르기 검출하는 boolean 형 변수
<memo_variable> 직전의 변수 상태를 스토어하기 위한 내부 변수

되돌리기 TRUE - TRUE 에서 FALSE 로 변화한 때

FALSE - 위와 다른 경우

FEDGE() 을 사용해서 오르기 검출은 1 회 실행 Scan 에 2 회 이상은 검출되지 않습니다. 오르기 검출이 S F C transition 조건에 사용되는 경우는 있습니다.

주의 : <memo_variable> 변수는 변수 직전의 상태를 보관하기 위해 사용되는 것으로, 다른 오르기 검출 등을 목적으로 동시 사용은 불가능합니다.

통상, 오르기 검출을 하고 싶은 변수명이 "xxx"일때, <memo_variable>로서는 "EDGE_xxx"을 사용합니다. 따라서 다른 FEDGE 조작에 의해 표시되는 일은 없습니다.

예 :

```
(* ST program using FEDGE operator *)
```

```
(* this program counts the falling edges of a boolean input *)
```

```
(* Bi120 is an input boolean variable *)
```

```
(* Edge_Bi120 is the memory of the Bi120 variable state *)
```

```
If FEDGE (Bi120, Edge_Bi120) Then
```

```
    Counter := Counter + 1;
```

```
End_if;
```

주의 : 이 명령은 IEC1131-3 에 포함되어 있지 않습니다. 환성을 위해서는 F_TRIG 표준 Block 의 사용을 권합니다.

B.7.5 ST 기본

S T 언어의 기본 스테이트먼트에는 이하의 것이 있습니다.

Assignment

RETURN

IF-THEN-ELSIF-ELSE 구조

CASE

WHILE

REPEAT

FOR

EXIT

≡ Assignment

이름 :=

의미 표현을 변수에 Assign

문법 <variable> := <any_expression> ;

operand 변수는 내부, 또는 출력 변수이어야 합니다.

변수와 표현 타입이 일치해야 합니다.

<any_expression>는 서브프로그램과 I S a G R A F library C언어 Function 이라도 상관없습니다.

예 :

(*Assign 을 사용한 S T 프로그램 *)

(* variable <= variable *)

bo23 := bo10;

(* variable <= expression *)

bo56 := bx34 OR alm100 & (level >= over_value);

result := (100 * input_value) / scale;

(* 서브프로그램의 되돌리기 값을 Assign *)

rc := PSelect ();

(* Function 호출을 Assign *)

limited_value := min (16, max (0, input_value));

RETURN

이름 **RETURN**

의미 현재 실행 중인 프로그램을 종료한다.

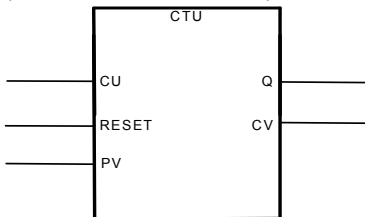
문법 **RETURN ;**

operand (none)

S F C 액션 Block 에서 RETURN 스테이트먼트는 그 액션 Block 만의 종료를 의미합니다.

예 :

(* FBD 프로그램 카운터 예*)



(* RETURN 스테이트먼트를 사용한 S T 프로그램 예 *)

If not (CU) then

 Q := false;

 CV := 0;

 RETURN; (*프로그램의 종료 *)

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

이름: IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF

의미 : S T 스테이트먼트를 boolean 형 표현의 값에 따라 선택해서 실행합니다.

문법:

```

IF <boolean_expression> THEN
    <스테이트먼트> ;
    <스테이트먼트> ;
    ...
ELSIF <boolean_expression> THEN
    < 스테이트먼트 > ;
    < 스테이트먼트 > ;
    ...
ELSE
    < 스테이트먼트 > ;
    < 스테이트먼트 > ;
    ...
END_IF;

```

ELSE 와 ELSIF 스테이트먼트는 옵션입니다. 만약, ELSE 스테이트먼트가 적혀 있지 않는 경우는 조건이 FALSE 일때 아무것도 실행되지 않습니다.

예 :

(* IF 스테이트먼트를 사용한 S T 프로그램 *)

```

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;

```

(* IF structure without ELSE *)

```

If overflow THEN
    alarm_level := true;

```

END_IF;



CASE

이름: CASE ... OF ... ELSE ... END_CASE

의미: 복수 S T 스테이트먼트 중에 정수표현 내용에 따라 스테이트먼트를 선택해 실행합니다.

문법:

```

CASE <integer_expression> OF
    <value> : <statements> ;
    <value> , <value> : <statements> ;
    ...
ELSE
    <statements> ;
END_CASE;
```

Case <value>값은 반드시 정수 이어야만 합니다. 복수<value>를 comma(.)로구분해 나열한 경우는 같은 스테이트먼트를 실행하는 것으로 의미합니다. ELSE 스테이트먼트는 옵션입니다.

예 :

(* CASE 스테이트먼트를 사용한 S T 프로그램 *)

```

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

이름값 WHILE ... DO ... END_WHILE

의미 : S T 스테이트먼트 그룹을 반복 실행합니다. 반복 전에 반복 조건을 평가합니다.

문법값

```

WHILE <boolean_expression> DO
    <statement> ;
    <statement> ;
    ...
END_WHILE ;
```

주의 : I S a G R A F 는 사이클 타임이 같은 기간인 시스템이기 때문에 예를 들면, 입력 변수는 W H I L E 의 반복 실행 중에는 변화하지 않습니다. 즉, 정수와 입력 변수를 직접 W H I L E 반복 조건에 사용할 수 없습니다. (무한 loop 가 되어버립니다.)

예 :

(* WHILE 스테이트먼트를 사용한 S T 프로그램 *)

(* C Function 을 Real 보드에서 문자 읽어내기에 사용하고 있습니다. *)

```
string := ""; (* empty string *)
nbchar := 0;
```

```
WHILE ((nbchar < 16) & ComIsReady ( )) DO
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
END_WHILE;
```

≡	REPEAT
이름:	REPEAT ... UNTIL ... END_REPEAT
의미:	S T 스테이트먼트 그룹을 반복 실행합니다. 반복 후에 반복 조건을 평가합니다.
문법:	REPEAT < 스테이트먼트 > ; < 스테이트먼트 > ; ... UNTIL <boolean_condition> END_REPEAT ;

주의 : I S a G R A F 는 사이클 타임이 같은 기간인 시스템이기 때문에 예를 들면, 입력 변수는 R E P E A T 의 반복 실행 중에는 변화하지 않습니다. 즉, 정수와 입력 변수를 직접 R E P E A T 반복 조건에 사용할 수 없습니다. (무한 loop 가 되어버립니다.)

예 :

(* REPEAT 스테이트먼트를 사용한 S T 프로그램 *)

(* C Function 을 Real Board 에서 문자 읽어내기에 사용하고 있습니다. *)

```
string := ""; (* empty string *)
nbchar := 0;
IF ComIsReady ( ) THEN
```

```

REPEAT
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
END_REPEAT;
END_IF;

```

이름: FOR ... TO ... BY ... DO ... END_FOR

의미 : 유한 회수분의 반복 조작을 정수형 변수를 index 취급법에 따릅니다.

문법: FOR <index> := <mini> TO <maxi> BY <step> DO
 <스테이트먼트> ;
 <스테이트먼트> ;
 END_FOR;

Operand: index: 내부 정수형 변수로 매회 증가
 mini: index 의 초기 값
 maxi: index 의 최대값
 step: index 의 매회 증가폭

[BY step] 스테이트먼트는 옵션입니다. 아무 지정이 없으면 STEP 은 1 입니다.

주의 : I S a G R A F 는 사이클 타임이 같은 기간인 시스템이기 때문에 예를 들면, 입력 변수는 FOR 의 반복 실행 중에는 변화하지 않습니다.

이하에서는 FOR 스테이트먼트와 등가한 WHILE 스테이트먼트를 나타냅니다.

```

index := mini;
while (step <= maxi) do
    <statement> ;
    <statement> ;
    index := index + step;
end_while;

```

예 :

(* FOR 스테이트먼트를 사용한 S T 프로그램 *)

(* 문자열에서 수치 캐릭터를 집어냅니다. *)

```

length := mlen (message);
target := ""; (* empty string *)
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;

```

	EXIT
이름:	EXIT
의미 :	FOR, WHILE, REPEAT 반복 스테이트먼트에서 EXIT
문법:	EXIT;
operand :	(none)

EXIT 는 통상 IF 스테이트먼트 내에서 사용됩니다.

예 :

(* EXIT 스테이트먼트를 사용한 S T 프로그램 *)
 (* 문자열에서 캐릭터를 검색합니다. *)

```

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 확장

이하에서는 S T 언어의 확장 부분에 대해서 설명합니다.

ISTART - ISTOP: timer 제어

이하의 스테이트먼트와 Function 은 child S F C 프로그램을 컨트롤할 때에 사용됩니다.
 ACTION(): ... END_ACTION; S F C 스텝 내에서 사용됩니다.

<u>GSTART</u>	SFC 프로그램 동작
<u>GKILL</u>	SFC 프로그램 정지
<u>GFREEZE</u>	SFC 프로그램 동결
<u>GRST</u>	동결된 SFC 프로그램 재동작
<u>GSTATUS</u>	SFC 프로그램 현재의 status 취득

주의 : 이 Function 은 IEC 1131-3 에는 포함되어 있지 않습니다. IEC1131-3 에 준거하는 경우는 GSTART 와 GKILL 대신에 다음의 문법으로 얻을 수 있습니다.

```
child_이름(S); (* GSTART(child_이름); 등가 *)
child_이름(R); (* GKILL(child_이름); 등가 *)
```

또한, S F C 스텝의 status 를 확인하는데 이하의 예가 있습니다.

GSnnn.x 스텝의 활성 / 비활성 상태를 boolean 값으로 얻는다.
GSnnn.t 스텝이 활성 상태된 후의 경과시간을 얻는다.
 ("nnn" 는 S F C 스텝 refresh 번호입니다.)

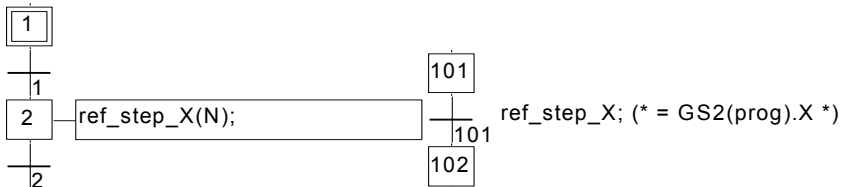
다른 S F C 프로그램 스텝의 활성 상태는 다음 표현으로 얻을 수 있습니다.

GSnnn(prog 이름).x

주의 : 다른 프로그램의 참조에 대해서 이 표현은 IEC1131-3 에는 포함되어 있지 않습니다. IEC1131-3 에 준거하는 경우는 글로벌 변수 (ref_step_X) 를 설정해서 이것에 참조하고 싶은 스텝 상태를 Assign 시킵니다. 즉, 스텝 안에서 N 액션 qualifier 를 사용해서 액션을 기술합니다. (ref_step_X(N);) 이러한 상태에서 스텝 상태를 테스트 (참조) 하고 싶은 장소에서 글로벌 변수 (ref_step_X) 를 사용합니다.

이하에 프로그램을 나타냅니다.

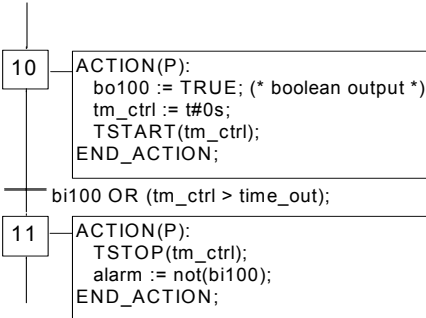
Prog 프로그램의 스텝상태를 사용하고 싶은 다른프로그램



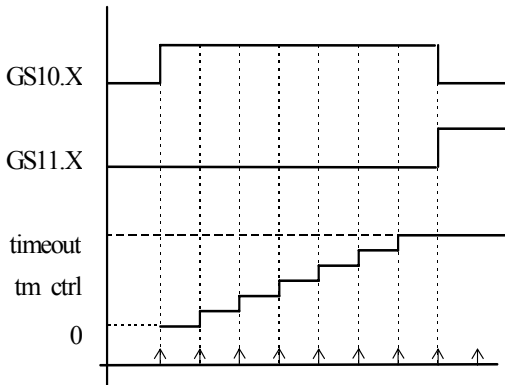
TSTART
이름: **TSTART**
의미: 타임형 변수의 카운터를 현재 값에서 갱신합니다.
 TSTART 커맨드로 타임형 변수는 변경되지 않습니다.
문법: **TSTART (<timer_variable>);**
operand: 비활성 상태의 타임 변수
되돌리기: (none)

예 :

(* TSTART, TSTOP 스테이트먼트를 사용한 S T 프로그램 *)



bi100 가 항상 FALSE 일때의 타임 차트:



타임 값은 사이클 중에는 같은 값을 유지합니다

	TSTOP
이름	TSTOP
의미	타임형 변수의 갱신을 정지합니다. TSTOP 커맨드로 타임형 변수는 변경되지 않습니다.
문법	TSTOP (<timer_variable>);
operand	활성 상태의 타임 변수
되돌리기	(none)

예 : 위의 TSTART 를 참조 바랍니다.

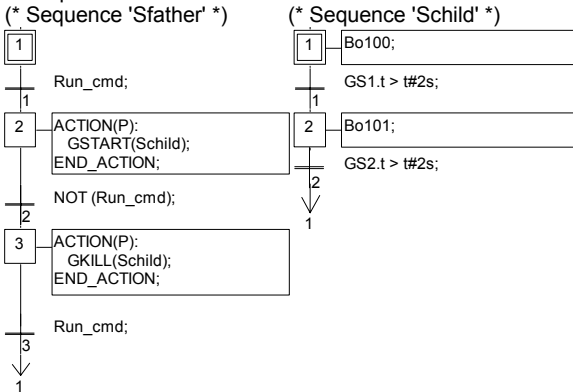
	GSTART
이름	GSTART
의미	child SFC 프로그램을 각 초기 스텝에 token 을 set 해서 작동시킵니다.
문법	GSTART (<child_program>);
operand	작동되는 child SFC 프로그램은 GSTART 스테이트먼트가 적혀 있는 SFC 프로그램의 child SFC 프로그램이어야 합니다.
되돌리기	(none)

child S F C 프로그램의 child (손자 S F C 프로그램) 는 자동으로 작동시킬 수 없습니다.

주의 : GSTART 는 IEC 1131-3 에는 포함되어 있지 않습니다. 이하와 같이 S qualifier 을 사용해서 child 프로그램을 작동합니다.

Child_이름(S);

Example: Use of GSTART and GKILL



	GKILL
이름	GKILL
의미	child SFC 프로그램에 있는 실행 token 을 없애면 child SFC 프로그램을 정지 합니다.
문법	GKILL (<child_program>);
operand	지정되는 SFC 프로그램은 GKILL 스테이트먼트가 적혀있는 끝편프로그램의 child SFC 프로그램이어야 합니다.
되돌리기	(none)

child S F C 프로그램의 child (손자 S F C 프로그램) 도 자동으로 정지됩니다.

주의 : GKILL 은 IEC 1131-3 norm 에 포함되어 있지 않습니다. 이하와 같이 R qualifier 을 사용해서 child 프로그램을 정지 시킵니다.

Child_이름(R);

Example: See GSTART (function described above)

⇐ GFREEZE

이름 GFREEZE

의미 child S F C 프로그램이 있는 실행 token 을 없애고, child S F C 프로그램을 정지합니다. 나중에 GRST 스테이트먼트로 재작동 가능합니다.

문법 GFREEZE (<child_program>);

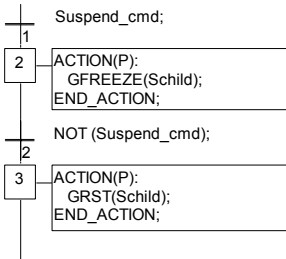
operand 지정되는 SFC 프로그램은 GFREEZE 스테이트먼트가 적혀있는 SFC 프로그램의 child-SFC 프로그램이어야 합니다.

되돌리기 (none)

child S F C 프로그램의 child (손자 S F C 프로그램) 도 자동적으로 동결됩니다.

주의 : GFREEZE 는 IEC 1131-3 에는 포함되지 않습니다.

Example:



⇐ GRST

이름: GRST

의미: GFREEZE 스테이트먼트로 동결되어 있던 child S F C 프로그램을 재작동합니다. GFREEZE 에서 없어진 실행 token 이 되돌아 옵니다.

문법: GRST (<child_program>);

operand: 지정되는 child-SFC 프로그램은 GRST 스테이트먼트가 적혀있는 SFC 프로그램의 child -SFC 프로그램이어야 합니다.

되돌리기: (none)

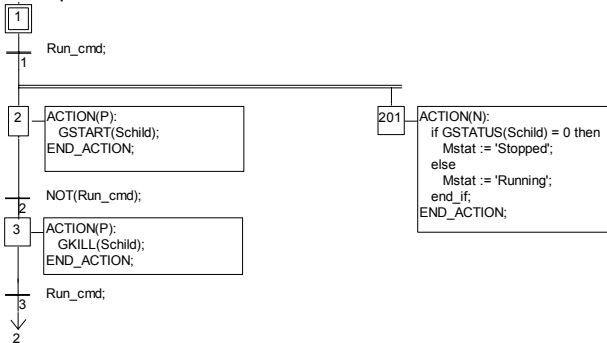
동결되어 있던 child SFC 프로그램의 child (손자 SFC 프로그램) 도 자동적으로 재작동 됩니다.

주의 : GRST 는 IEC 1131-3 에 포함되지 않습니다.

예 : 위 GFREEZE 스테이트먼트를 참조

	GSTATUS
이름	GSTATUS
의미	child SFC 프로그램의 현재상태를 되돌립니다.
문법	<ana_var> := GSTATUS (<child_program>);
operand	지정되는 child SFC 프로그램은 GSTATUS 스테이트먼트가 적혀있는 SFC 프로그램의 child SFC 프로그램이어야 합니다.
되돌리기	0 = 프로그램이 비활성 상태 (killed) 1 = 프로그램이 활성 상태 (started) 2 = 프로그램이 동결 상태 (frozen)

Example:



B.8 IL 언어

명령 리스트 (I L) 는 low 레벨 언어입니다. 적은 어플리케이션과 어플리케이션 특정부분에 가장 적합합니다. 인스트럭션은 항상 **현재 결과** (즉, **IL register**) 에 관련되어 있습니다. operation 은 현상 값과 operation 사이에서 반드시 행해집니다. 그 결과는 다시 현재 결과로 저장 됩니다.

B.8.1 IL 문법

IL 프로그램은 인스트럭션 리스트에도 있습니다. 각각의 인스트럭션은 새로운 라인에 적혀, **operator (명령)** 와 **수식자 (수식자)** 필요에 응해서 하나 이상의 **operator** 에 이루어져 있습니다. 이 요소들은 콤마 (,) 로 구분되어 있습니다. **라벨** : 을 인스트럭션 앞에 놓는 것도 가능합니다. 만약, 커맨드를 기입할 경우는 라인의 마지막에 추가 가능합니다. 단, (* 로 시작되고 *)로 끝나야 합니다. 공백 라인도 인정되고, 여기에 커맨드를 기입할 수도 있습니다.

라벨	명령	Operand	주석
Start:	LD	IX1	(* push button *)
	ANDN	MX5	(* command is not forbidden *)
	ST	QX2	(* start motor *)

라벨

라벨명 뒤에는 콤마 (:) 가 붙습니다. 인스트럭션은 이 라벨 뒤에서부터 적힙니다. 라벨은 공백행에 적힙니다. 라벨명은 다른 인스트럭션 안에서 점프 전에 operand 에서 사용되는 일이 있습니다. 라벨명은 이하의 룰에 따릅니다.

- 이름은 1 6 문자 이내
- 최초의 문자는 레터 (a - z)
- 그 이후는 레터, 숫자, '_'

같은 라벨명은 동일 I L 프로그램 안에서는 사용할 수 없습니다. 단, 변수명과 동일 라벨명을 정의할 수는 있습니다.

명령 수식자

취급가능한 명령 수식자를 이하에 나타냅니다. 수식자 문자는 명령을 완전한 형으로 합니다. 수식자 사이에 스페이스를 포함하면 안됩니다.

N operand 의 반전

- (지연 조작
- C 조건 조작

'N' 수식자는 boolean operand 의 반전을 의미합니다.

예를 들면,

ORN IX12 는 S T 에션 이하와 같이 됩니다.

result := result OR NOT (IX12)

'(' 수식자는 ')' 가 발견될 때까지 괄호로 둘러 싸인 인스트럭션의 실행을 먼저 보내게 됩니다.

'C' 수식자는 이하의 인스트럭션은 현재 결과가 T R U E (boolean 형 변수 이외의 경우는 0 이외의 값) 일때만 실행되도록 하는 조건 입니다. 'C' 수식자는 'N' 수식자와 합쳐서 사용할 수도 있습니다. 이 경우는 현재 결과가 F A L S E (boolean 형 변수 이외의 경우는 0) 일때만 이하의 인스트럭션이 실행됩니다.

지연 조작

하나의 I L register (즉, 현재 결과) 밖에 없기 때문에 어떤 처리는 지연 조작 (먼저 보내기) 되어야 합니다. 인스트럭션 실행 순서가 변경됩니다.

'(' 수식자	처리의 먼저 보내기를 의미합니다.
')' 명령	먼저 보내져 있던 처리를 실행합니다.

'(' 수식자는 ')' 가 발견될 때까지 괄호로 둘러 싸인 인스트럭션의 실행을 먼저 보내게 됩니다.

예를 들면,

**AND(IX12
OR IX35
)**

는 이하와 같이 해석됩니다.

result := result AND (IX12 OR IX35)

B.8.2 IL 명령

이하의 테이블은 I L 언어의 표준명령을 종합한 것입니다.

명령어	수식자	동작	설명
-----	-----	----	----

<u>LD</u>	N	변수, 상수	로드
<u>ST</u>	N	변수	현재결과저장
<u>S</u>		BOO 변수	TRUE 설정
<u>R</u>		BOO 변수	FALSE 재설정
<u>CAL</u>	C N	FB instance 명	function block 호출
<u>JMP</u>	C N	라벨	라벨 점프
<u>RET</u>	C N		리턴
<u>└</u>			지연조작 실행
<u>Function</u>			function / 서브프로그램 호출
AND	N (	BOO	boolean AND
&	N (	BOO	boolean AND
OR	N (	BOO	boolean OR
XOR	N (	BOO	exclusive OR
ADD	(	변수, 상수	Addition
SUB	(	변수, 상수	Subtraction
MUL	(	변수, 상수	Multiplication
DIV	(	변수, 상수	Division
GT	(	변수, 상수	Test: >
GE	(	변수, 상수	Test: >=
EQ	(	변수, 상수	Test: =
LE	(	변수, 상수	Test: <=
LT	(	변수, 상수	Test: <
NE	(	변수, 상수	Test: <>

이하에 I L 고유의 명령만 나타냅니다. 기타 표준 명령은 「표준 명령, Function Block, Function」으로 해설합니다.

LD 명령

operation : 현재 결과에 값을 로드합니다.

취급가능한 수식자 : N

operand : 정수 또는 내부, 입력, 출력 변수

Example:

```
(* EXAMPLES OF LD OPERATIONS *)
LDex:  LD      false      (* result := FALSE boolean 상수 *)
        LD      true       (* result := TRUE  boolean 상수 *)
        LD      123        (* result := integer 상수 *)
        LD      123.1      (* result := real  상수 *)
        LD      t#3ms      (* result := time  상수 *)
        LD      boo_var1   (* result := boolean 변수 *)
        LD      ana_var1   (* result := analog 변수 *)
        LD      tmr_var1   (* result := timer 변수 *)
        LDN     boo_var2   (* result := NOT ( boolean 변수 ) *)
```

== ST 명령

operation : 현재 결과를 저장 합니다. 현재 결과는 이 조건에서
변화하지 않습니다.

취급 가능한 수식자 : N

operand : 내부, 출력 변수

Example:

```
(* EXAMPLES OF ST OPERATIONS *)
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 명령

operation: 현재 결과가 TRUE 일때 boolean 형 변수에 꺾꺾꺾꺾를 저장 합니다.
만약, 현재 결과가 FALSE 일때는 아무것도 처리하지 않습니다. 이
조작에 따라 현재 결과는 변화하지 않습니다.

취급 가능한 수식자 : (none)

operand: 출력, 입력 변수 boolean 형 변수

Example:

```
(* EXAMPLES OF S OPERATIONS *)
SETex: LD      true       (* current result := TRUE *)
        S      boo_var1   (* boo_var1 := TRUE *)
                          (* current result is not modified *)
```

```
LD      false      (* current result := FALSE *)
S      boo_var1 (* nothing done - boo_var1 unchanged *)
```

≡ R 명령

operation: 현재 결과가 TRUE 일때 boolean 형 변수에 FALSE 일때는 스토어합니다. 만약, 현재 결과가 FALSE 일때는 아무것도 처리하지 않습니다. 이 조작에 따라 현재 결과는 변화하지 않습니다.

취급 가능한 수식자: (none)

operand : 출력, 입력 변수 boolean 형 변수

Example:

```
(* EXAMPLES OF R OPERATIONS *)
RESETex: LD      true      (* current result := TRUE *)
          R      boo_var1 (* boo_var1 := FALSE *)
                      (* current result is not modified *)
          ST      boo_var2 (* boo_var2 := TRUE *)
          LD      false     (* current result := FALSE *)
          R      boo_var1 (* nothing done - boo_var1 unchanged *)
```

≡ JMP 명령

operation : 특정 라벨로 점프

취급 가능한 수식자: C N

operand : 동일 IL 프로그램에서 정의된 라벨

예 :

(* 이하의 예는 아나로그 selector 을 테스트합니다. (0 or 1 or 2)

(* 세개의 출력 boolean 형에 1 을 set. Selector 내용이 0 일때 점프합니다. *)

```
JMPex: LD      selector (* selector is 0 or 1 or 2 *)
        BOO     (* conversion to boolean *)
        JMPC    test1   (* if selector = 0 then *)
        LD      true
        ST      bo0     (* bo0 := true *)
        JMP     JMPend  (* end of the program *)
test1:  LD      selector
        SUB     1       (* decrease selector: is now 0 or 1 *)
        BOO     (* conversion to boolean *)
        JMPC    test2   (* if selector = 0 then *)
        LD      true
        ST      bo1     (* bo1 := true *)
        JMP     JMPend  (* end of the program *)
test2:  LD      true     (* last possibility *)
        ST      bo2     (* bo2 := true *)
JMPend: (* end of the IL program *)
```

RET 명령

operation : 현 명령 리스트를 종료합니다. 만약,
IL sequence 가 서브 프로그램일때는 IL register(현재결과)가
Father 프로그램에 되돌리기 값으로 되돌아 옵니다.

취급 가능한 수식자 : C N

operand : (none)

예 :

(*이하의 예는 Analog selector 의 값을 테스트합니다. (0 or 1 or 2) *)

(*3 개의 출력 boolean 형에 1 을 set. Selector 내용이 0 일때 점프합니다.)

```
JMPex: LD      selector  (* selector is 0 or 1 or 2 *)
        BOO      (* conversion to boolean *)
        JMPCL test1    (* if selector = 0 then *)
        LD      true
        ST      bo0    (* bo0 := true *)
        RET          (* end - return 0 *)
        (* decrease selector *)

test1:  LD      selector
        SUB      1      (* selector is now 0 or 1 *)
        BOO      (* conversion to boolean *)
        JMPCL test2    (* if selector = 0 then *)
        LD      true
        ST      bo1    (* bo1 := true *)
        LD      1      (* load real selector value *)
        RET          (* end - return 1 *)
        (* last possibility *)

test2:  RETNC      (* returns if the selector has *)
        (* an invalid value *)

        LD      true
        ST      bo2    (* bo2 := true *)
        LD      2      (* load real selector value *)
        RET          (* end - return 2 *)
```

)" 명령

operation 지연(먼저 보내기)되어 있던 처리를 실행합니다. 먼저 보내져 있던
처리는 '(' 으로 시작되고 있습니다.

취급 가능한 수식자: (none)

operand: (none)

예 :

(* 지연 조작을 포함한 예 *)

```

Delayed: LD      a1      (* result := a1; *)
          ADD(    a2      (* delayed ADD - result := a2; *)
          MUL(    a3      (* delayed MUL - result := a3; *)
          SUB     a4      (* result := a3 - a4; *)
          )        (* execute delayed MUL - result := a2 * (a3-a4); *)
          MUL     a5      (* result := a2 * (a3 - a4) * a5; *)
          )        (* execute delayed ADD *)
                  (* result := a1 + (a2 * (a3 - a4) * a5); *)
          ADD     a6      (* result := a1 + (a2 * (a3 - a4) * a5) + a6; *)
          ST      res     (* store current result in variable res *)
    
```

서브-프로그램 / functions 호출

I L, S T, L D, F B D으로 적힌 서버프로그램과 Function 은 I L에서 호출 가능합니다. 서버프로그램명을 명령으로서 지정합니다.

operation : 서버프로그램과 Function 을 실행합니다. 서버프로그램에서 되돌리기 값은 꺾꺾 register(현재결과)에 저장 됩니다.

취급 가능한 수식자: (none)

operand : 최초의 parameter 는 꺾꺾 register 에 스토어 됩니다. 이하의 parameter 는 커맨드로 구분되고 operand 필드에 적힙니다.

예 :

(* call 프로그램 : analog 값을 time 값으로 변환 *)

```

Main:      LD      bi0
          SUBPRO   bi1,bi2  (* call sub-program to get analog value *)
          ST      result    (* result := value returned by sub-program *)
          GT      vmax      (* test value overflow *)
          RETC     (* return if overflow *)
          LD      result
          MUL      1000     (* converts seconds in milliseconds *)
          TMR      (* converts to a timer *)
          ST      tmval     (* stores converted value in a timer *)
    
```

(* 읽어 낼수 있는 서버프로그램 'SUBPRO' : Analog 값의 평가 *)

(* 3 개의 vinally boolean : in0, in1, in2 가 서버프로그램의 입력 parameter *)

```

          LD      in2
          ANA      (* result = ana (in2); *)
          MUL      2      (* result := 2*ana (in2); *)
          ST      temporary (* temporary := result *)
          LD      in1
          ANA
          ADD      temporary (* result := 2*ana (in2) + ana (in1); *)
          MUL      2      (* result := 4*ana (in2) + 2*ana (in1); *)
    
```

```

ST      temporary      (* temporary := result *)
LD      in0
ANA
ADD      temporary      (* result := 4*ana (in2) + 2*ana (in1)+ana (in0); *)
ST      SUBPRO (* return current result to calling program *)

```

== function blocks 호출 : CAL 명령

operation: FunctionBlock 의 호출

취급 가능한 수식자: C N

operand: FunctionBlock instance 명

FunctionBlock 입력 parameter 는 call 되기 전에 LD/ST 명령 으로 assign 해 둘 필요가 있습니다.

예 1 :

(* SR 의 call : SR1 은 SR 의 instance *)

```

LD      auto_mode
AND      start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL      SR1
LD      SR1.Q1
ST      command

```

—
예제 2

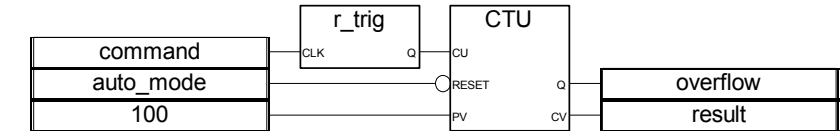
(*We suppose R_TRIG1 is an instance of R_TRIG block and CTU1 is an instance of CTU block*)

```

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

```

(* FBD 등가: *)



B.9 표준명령, FunctionBlock 과 Function

B.9.1 표준명령

이하에 IEC 1131 에 정의 되어 있는 표준 명령을 나타냅니다.

데이터 조작 : Assignment (Assign) , Analog negation (부호 반전)

boolean 조작 : AND (논리곱)

OR (논리합)

XOR (배타적 논리화)

연산 연산 : ADD (더하기)

SUB (빼기)

MUL (곱하기)

DIV (나누기)

논리 조작 : AND_MASK (bit 마다 AND 마스크)

OR_MASK (bit 마다 OR 마스크)

XOR_MASK (bit 마다 XOR 마스크)

NOT_MASK (bit 의 반전)

비교 테스트 : LT

LE

GT

GE

EQ

NE

데이터 변수 : BOO (boolean 형 변환)

ANA (정수형 변환)

REAL (실수형 변환)

TMR (타임형 변환)

MSG (문자열형 변환)

기타 : CAT

SYSTEM

OPERATE



인수:

IN 모든 타입
Q 모든 타입

설명:

어떤 변수를 다른 변수에 assign 합니다.

이 블록은 입력과 출력을 직접 접속하는 블록입니다. 논리의 반전 (◦ 심벌) 도 합쳐서 사용 가능합니다.

(* Assignment 블록을 이용한 F B D 예 *)

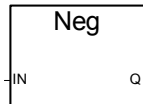
—
(* 등가한 S T 언어: *)

ao23 := ai10;
bo100 := NOT (bi1 AND bi2);

(* 등가한 IL 언어: *)

LD ai10
ST ao23
LD bi1
AND bi2
LDN bo100

NEG (부호 반전)



인수:

IN 정수· 실수형 입력, 출력 parameter 는 같은 타입
Q 정수· 실수형

설명:

변수의 반전 (부호 반전) assign

(* Negation 블록을 이용한 F B D 예 *)

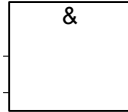
—
(* 등가한 S T 언어: *)

```
ao23 := - (ai10);
ro100 := - (ri1 + ri2);
```

(* 등가한 IL 언어: *)

```
LD      ai10
MUL     -1
ST      ao23
LD      ri1
ADD     ri2
MUL     -1.0
LD      ro100
```

& AND (논리곱)



주의: 이 명령은 확장 명령이라 합니다. 입력수의 변경은 가능합니다.

인수:

(inputs)	boolean 형	
output	boolean 형	입력 parameter 의 논리적

설명:

두개 이상의 입력 parameter 의 논리곱

(* "AND" 블록을 이용한 F B D 예*)

—
(* 등가한 S T 언어: *)

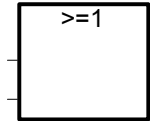
```
bo10 := bi101 AND NOT (bi102);
bo5 := (bi51 AND bi52) AND bi53;
```

(* 등가한 I L 언어: *)

LD	bi101	(* current result := bi101 *)
ANDN	bi102	(* current result := bi101 AND not(bi102) *)
ST	bo10	(* bo := current result *)
LD	bi51	(* current result := bi51;

&	bi52	(* current result := bi51 AND bi52 *)
&	bi53	(* current result := (bi51 AND bi52) AND bi53 *)
ST	bo5	(* bo := current result *)

>=1 OR (논리합)



주의: 이 명령은 Enhanced operator 라고 합니다. 입력 변경은 가능합니다.

인수:

(inputs)	boolean 형	
output	boolean 형	입력 parameter 의 논리합

설명:

두개 이상의 입력 parameter 의 논리합

(* "OR" 블록을 이용한 F B D 예 *)

—

(* 등가한 S T 언어: *)

bo10 := bi101 OR NOT (bi102);

bo5 := (bi51 OR bi52) OR bi53;

(* 등가한 IL 언어: *)

LD bi101

ORN bi102

ST bo10

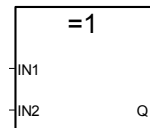
LD bi51

OR bi52

OR bi53

ST bo5

=1 XOR (배타적 논리합)



인수:

IN1	boolean 형	
IN2	boolean 형	
Q	boolean 형	두개의 boolean 입력의 배타적 논리합

설명:

두개의 boolean 입력의 배타적 논리합

(* "XOR" 블록을 이용한 F B D 예 *)

—
(* 등가한 S T 언어: *)

bo10 := bi101 XOR NOT (bi102);

bo5 := (bi51 XOR bi52) XOR bi53;

(* 등가한 I L 언어: *)

LD bi101

XORN bi102

ST bo10

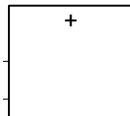
LD bi51

XOR bi52

XOR bi53

ST bo5

+ (더하기)



주의: 이 명령은 Enhanced operator 라고 합니다. 입력수의 변경은 가능합니다.

인수:

(inputs)	정수· 실수형 (모든 입력은 동일 타입)
output	정수· 실수형 모든 입력 부호 더하기

설명:

정수· 실수형 변수의 부호 더하기

(* 가산 블록을 이용한 F B D 예*)

—

(* 등가한 S T 언어: *)

ao10 := ai101 + ai102;

ao5 := (ai51 + ai52) + ai53;

(* 등가한 I L 언어: *)

LD ai101

ADD ai102

ST ao10

LD ai51

ADD ai52

ADD ai53

ST ao5

- (빼기)



인수:

IN1	정수· 실수형	
IN2	정수· 실수형	(IN1 과 IN2 는 같은 타입)
Q	정수· 실수형	감산 결과 (IN1 - IN2)

설명:

두개의 정수· 실수형 변수의 빼기 (1 번째 - 2 번째)

(* 감산 블록을 이용한 F B D 예 *)

(* 등가한 S T 언어: *)

ao10 := ai101 - ai102;

ao5 := (ai51 - 1) - ai53;

(* 등가한 I L 언어: *)

LD ai101

SUB ai102

ST ao10

LD ai51

SUB 1

SUB ai53

ST ao5

* (곱하기)



주의: 이 명령은 확장형 명령이라고 합니다. 입력수의 변경은 가능합니다.

인수:

(inputs)	정수· 실수형	(모든 입력은 같은 타입)
output	정수· 실수형	입력부호 곱하기

설명:

두개 이상의 정수· 실수형 변수의 곱하기

(* 승산 블록을 이용한 F B D예*)

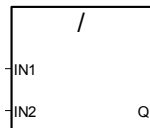
(* 등가한 S T 언어: *)

```
ao10 := ai101 * ai102;
ao5 := (ai51 * ai52) * ai53;
```

(* 등가한 IL 언어: *)

```
LD      ai101
MUL     ai102
ST      ao10
LD      ai51
MUL     ai52
MUL     ai53
ST      ao5
```

/ (나누기)



인수:

IN1	정수· 실수형	
IN2	정수· 실수형	0 이 아닐 것

(IN1 과 IN2 는 같은 타입)

Q 정수· 실수형 부호 나누기 (IN1 / IN2)

설명:

두개의 정수· 실수형 변수의 나누기 (1 번째 / 2 번째)

(* 계산 블록을 이용한 F B D 예*)

—

(* 등가한 S T 언어: *)

ao10 := ai101 / ai102;

ao5 := (ai5 / 2) / ai53;

(* 등가한 IL 언어: *)

LD ai101

DIV ai102

ST ao10

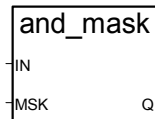
LD ai51

DIV 2

DIV ai53

ST ao5

AND_MASK



인수:

IN 정수형

MSK 정수형

Q 정수형 IN 과 MSK 의 bit 마다 논리곱

설명:

정수형 변수의 논리적으로 IN 의 각 bit 과 MSK 의 각 bit 와의 논리곱

(*Analog AND_MASK 블록을 이용한 F B D 예*)

—

(* 등가한 S T 언어: *)

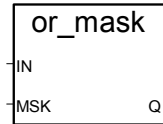
parity := AND_MASK (xvalue, 1); (* 1 if xvalue is odd *)

result := AND_MASK (16#abc, 16#f0f); (* equals 16#a0c *)

(* 등가한 I L 언어: *)

```
LD      xvalue
AND_MASK 1
ST      parity
LD      16#abc
AND_MASK 16#f0f
ST      result
```

OR_MASK



인수:

IN	정수형	
MSK	정수형	
Q	정수형	IN 과 MSK 의 bit 마다 논리합

설명:

정수형 변수의 논리화에서 IN 의 각 bit 과 MSK 의 각 bit 와의 논리합

(*Analog OR_MASK 블록을 이용한 F B D 예*)

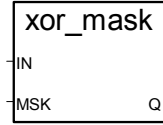
—
(* 등가한 S T 언어: *)

```
is_odd := OR_MASK (xvalue, 1); (* makes value always odd *)
result := OR_MASK (16#abc, 16#f0f); (* equals 16#fbf *)
```

(* 등가한 I L 언어: *)

```
LD      xvalue
OR_MASK 1
ST      is_odd
LD      16#abc
OR_MASK 16#f0f
ST      result
```

XOR_MASK



인수:

IN	정수형	
MSK	정수형	
Q	정수형	IN 과 MSK 의 bit 마다 배타적 논리합

설명:

정수형 변수의 배타적 논리화로 IN 의 각 bit 과 MSK 의 각 bit 과의 배타적 논리합

(*XOR_MASK 블록을 이용한 F B D 예 *)

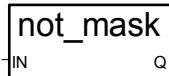
(* 등가한 S T 언어: *)

```
crc32 := XOR_MASK (prevcrc, nextc);
result := XOR_MASK (16#012, 16#011); (* equals 16#003 *)
```

(* 등가한 IL 언어: *)

```
LD      prevcrc
XOR_MASK nextc
ST      crc32
LD      16#012
XOR_MASK 16#011
ST      result
```

NOT_MASK



인수:

IN	정수형	
Q	정수형	3 2 bit 표현 IN 의 각 bit 의 반전

설명:

정수형 변수의 각 bit 의 반전

(*NOT_MASK 블록을 이용한 F B D 예*)

(* 등가한 S T 언어: *)

result := NOT_MASK(16#1234);

(* result is 16#FFFF_EDCB *)

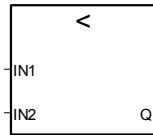
(* 등가한 IL 언어: *)

LD 16#1234

NOT_MASK

ST result

< (LT)



인수:

IN1 정수형· 실수형-

타입형 문자열형

IN2 정수· 실수형-

타입형 문자열형 두개의 입력은 동일 타입

Q boolean 형 만약, IN1 < IN2 이면 TRUE

설명:

모든 타입의 두개의 변수 사이에서의 비교 (LESS THAN)

(* "Less than" 블록을 이용한 F B D 예*)

(* 등가한 S T 언어: *)

areult := (10 < 25); (* areult is TRUE *)

mresult := ('z' < 'B'); (* mresult is FALSE *)

(* 등가한 IL 언어: *)

LD 10

LT 25

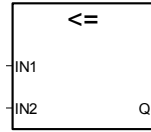
ST areult

LD 'z'

LT 'B'

ST mresult

<= (LE)



인수:

IN1	정수· 실수형 문자열형	
IN2	정수· 실수형 문자열형	두개의 입력은 동일 타입
Q	boolean 형	만약, IN1 <= IN2 이면 TRUE

설명:

두개의 정수, 실수, 문자열형 변수 사이에서의 비교
(LESS THAN or EQUAL)

("Less or equal to" 블록을 이용한 F B D 예*)

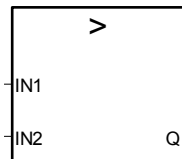
(* 등가한 S T 언어: *)

```
aresult := (10 <= 25); (* aresult is TRUE *)
mresult := ('ab' <= 'ab'); (* mresult is TRUE *)
```

(* 등가한 IL 언어: *)

```
LD      10
LE      25
ST      aresult
LD      'ab'
LE      'ab'
ST      mresult
```

> (GT)



인수:

IN1	정수· 실수형- 타임형 문자열형
------------	----------------------

IN2 정수· 실수형-
타입형 문자열형 두개는 동일 타입

Q boolean 형 (IN1 > IN2 에서 TRUE)

설명:

두개의 변수 사이에서의 비교 (GREATER THAN)

("Greater than" 블록을 이용한 F B D 예*)

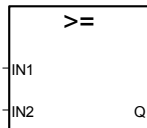
(* 등가한 S T 언어: *)

result := (10 > 25); (* result is FALSE *)
mresult := ('ab' > 'a'); (* mresult is TRUE *)

(* 등가한 IL 언어: *)

LD 10
GT 25
ST result
LD 'ab'
GT 'a'
ST mresult

>= (GE)



인수:

IN1 정수· 실수형 문자열형

IN2 정수· 실수형 문자열형 두개의 입력은 동일 타입

Q boolean 형 IN1 >= IN2 에서 TRUE

설명:

두개의 변수 사이에서의 비교 (GREATER THAN or EQUAL TO)

("Greater or Equal to"블록을 이용한 F B D 예 *)

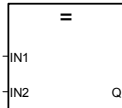
(* 등가한 ST 언어: *)

result := (10 >= 25); (* result is FALSE *)
mresult := ('ab' >= 'ab'); (* mresult is TRUE *)

(* 등가한 IL 언어: *)

```
LD      10
GE      25
ST      arestult
LD      'ab'
GE      'ab'
ST      mresult
```

= (EQ)



인수:

IN1 정수· 실수형 문자열형
IN2 정수· 실수형 문자열형 두개의 입력은 동일 타입
Q boolean 형 (IN1 = IN2 에서 TRUE)

설명:

두개의 변수 사이에서의 비교 (EQUAL TO)

(* "Is Equal to" 블록을 이용한 F B D 예*)

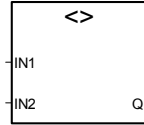
—
 (* 등가한 ST 언어: *)

```
arestult := (10 = 25); (* arestult is FALSE *)
mresult := ('ab' = 'ab'); (* mresult is TRUE *)
```

(* 등가한 IL 언어: *)

```
LD      10
EQ      25
ST      arestult
LD      'ab'
EQ      'ab'
ST      mresult
```

<> (NE)



인수:

IN1	정수· 실수형 문자열형	
IN2	정수· 실수형 문자열형	두개의 입력은 동일 타입
Q	boolean 형	(IN1 <> IN2 에서 TRUE)

설명:

두개의 변수 사이에서의 비교 (NOT EQUAL TO)

(* "Is Not Equal to" 블록을 이용한 F B D 예 *)

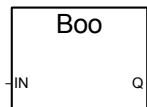
—
(* 등가한 ST 언어: *)

```
aresult := (10 <> 25); (* aresult is TRUE *)
mresult := ('ab' <> 'ab'); (* mresult is FALSE *)
```

(* 등가한 IL 언어: *)

```
LD      10
NE      25
ST      aresult
LD      'ab'
NE      'ab'
ST      mresult
```

BOO (이진 [boolean 형] 변환)



인수:

IN	모든 타입	(boolean 형 이외)
Q	boolean 형	TRUE : 0 이외 FALSE : 0 TRUE : 'TRUE' 문자열 FALSE : 'FALSE' 문자열

설명:

변수의 boolean 형으로의 변환

(* "boolean 형 변환 블록을 이용한 FBD 예 *)

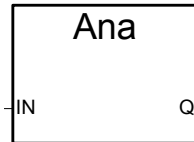
```

—
(* 등가한 ST 언어: *)
ares := BOO (10);           (* ares is TRUE *)
tres := BOO (t#0s);         (* tres is FALSE *)
mres := BOO ('false');      (* mres is FALSE *)
    
```

```

(* 등가한 IL 언어: *)
LD      10
BOO
ST      ares
LD      t#0s
BOO
ST      tres
LD      'false'
BOO
ST      mres
    
```

ANA (정수형 변환)



인수:

IN	모든 타입 (정수형 이외)
Q	정수형
	0 : IN 이 FALSE 일 때
	1 : IN 이 TRUE 일 때
	타임 m s 수치
	실수 변수의 정수부분
	문자열의 1 0 진수 표현

설명:

변수의 정수형으로의 변환

(* "Convert to Analog" 블록을 이용한 FBD 예 *)

—

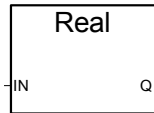
(* 등가한 ST 언어: *)

bres := ANA (true);	(* bres is 1 *)
tres := ANA (t#1s46ms);	(* tres is 1046 *)
mres := ANA ('0198');	(* mres is 198 *)

(* 등가한 IL 언어: *)

LD	true
ANA	
ST	bres
LD	t#1s46ms
ANA	
ST	tres
LD	'0198'
ANA	
ST	mres

REAL (실수형 변환)



인수:

IN	boolean 형 정수형-	
	타입형	실수, 문자열형 이외
Q	실수형	0.0 : IN 이 FALSE / 1.0 : IN 이 TRUE
		타임 m s 수치
		정수형과 같은 수치

설명:

변수의 실수형으로의 변환

(* "Convert to Real" 블록을 이용한 FBD 예 *)

(* 등가한 ST 언어: *)

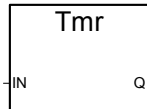
bres := REAL (true);	(* bres is 1.0 *)
tres := REAL (t#1s46ms);	(* tres is 1046.0 *)
ares := REAL (198);	(* ares is 198.0 *)

(* 등가한 IL 언어: *)

LD	true
REAL	

ST	bres
LD	t#1s46ms
REAL	
ST	tres
LD	198
REAL	
ST	ares

TMR (타임형 변환)



인수:

IN	정수·실수형 (타임 이외)
Q	타임형 IN으로 표현되는 타임 값 m s 표시 IN이 실수일때는 그 정수부분

설명:

정수·실수형 변수의 타임 값으로의 변환

(* "Convert to Timer"블록을 이용한 FBD 예 *)

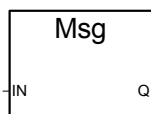
(* 등가한 ST 언어: *)

ares := TMR (1256);	(* ares := t#1s256ms *)
rres := TMR (1256.3);	(* rres := t#1s256ms *)

(* 등가한 IL 언어: *)

LD	1256
TMR	
ST	ares
LD	1256.3
TMR	
ST	rres

MSG (문자열형 변수)



인수:

IN boolean 형-

실수형 - 실수형 (문자열형 이외)

Q 문자열형 'false' 또는 'true' : IN 이 boolean 형 일 때
1 0 진수 표현 IN 이 정수, 실수형 일 때

설명:

변수의 문자열형으로의 변환

(* "Convert to Message" 블록을 이용한 FBD 예 *)

—
(* 등가한 ST 언어: *)

bres := MSG (true); (* bres is 'TRUE' *)

ares := MSG (125); (* ares is '125' *)

(* 등가한 IL 언어: *)

LD true

MSG

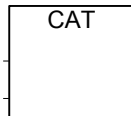
ST bres

LD 125

MSG

ST ares

CAT



주의: 이 명령은 확장형 명령이라 합니다. 입력수의 변경은 가능합니다.

인수:

(inputs) 문자열형 (모든 문자열형 변수의 결합, 출력 문자열의 최대 치를
 넘지 않을 것)

output 문자열형 입력 문자열의 결합

설명:

복수 문자열을 하나로 결합

(* "Message Concatenation" 블록을 이용한 FBD 예 *)

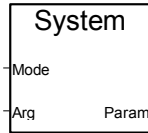
—

(*등가한 ST 언어: use the + operator *)
 myname := ('Mr' + ' ') + 'Jones';
 (* means: myname := 'Mr Jones' *)

(* 등가한 IL 언어: *)

```
LD      'Mr'
ADD     ''
ADD     'Jones'
ST      myname
```

SYSTEM



인수:

Mode	정수형	시스템 parameter 의 식별자, 또는 access 모드
Arg	정수형	타입형 적어 넣기 모드일 때 적어넣을 값
Param	정수형	access 결과 값

설명:

시스템 parameter 의 access

이하에 다룰 수 있는 SYSTEM Function 커맨드 (모드) 예를 나타냅니다. 이들 커맨드는 정의 종료 워드입니다.

Command	Meaning
SYS_TALLOWED	허용 사이클 타임 읽어 내기
SYS_TCURRENT	현재 사이클 타임 읽어 내기
SYS_TMAXIMUM	최대 사이클 타임 읽어 내기
SYS_TOVERFLOW	사이클 타임 over follow 회수 읽어 내기
SYS_TRESET	다이내믹 카운터 reset
SYS_TWRITE	사이클 타임의 변경
SYS_ERR_TEST	런타임 에러 체크

SYS_ERR_READ	최신 런타임 에러 읽어 내기
---------------------	-----------------

이하에 S Y S T E M Function 각각의 커맨드에 대한 인수를 나타냅니다.

<i>Command</i>	<i>인수</i>	<i>되돌리기 값</i>
SYS_TALLOWED	0	허용 사이클 타임
SYS_TCURRENT	0	현재 사이클 타임
SYS_TMAXIMUM	0	최대 사이클 타임
SYS_TOVERFLOW	0	Over follow 회수
SYS_TRESET	0	0
SYS_TWRITE	새로운 사이클 타임	적어 넣은 사이클 타임
SYS_ERR_TEST	0	에러가 없으면 0
SYS_ERR_READ	0	최신 에러 코드

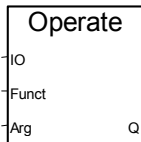
(* "System" 블록을 이용한 F B D 예*)

```

(* 등가한 S T 언어: *)
alarm := (SYSTEM (SYS_TOVERFLOW, 0) <> 0);
If (alarm) Then
    nb_err := nb_err + 1;
    rc := SYSTEM (SYS_TRESET, 0);
End_If;

```

OPERATE



인수:

IO	모든 타입	입출력 변수
Funct	정수형	지정 Function
Arg	정수형	I/O 액션에 대한 인수
Q	정수형	되돌리기 값

설명:

I/O 채널로 access

OPERATE 처리 내용은 I/O 드라이브에 따라 결정됩니다. 상세한 것은 대응하는 I/O 드라이브의 기술 메모 등을 참조 바랍니다.

B.9.2 표준 function blocks

ISaGRAF 에 따라 support 되고 있는 표준 FunctionBlock 을 이하에 나타냅니다. 이들은 표준으로 구비되어 있기 때문에 ISaGRAF library 에 등록할 필요가 없습니다.

boolean 조작 :SR : (set 우선 쌍안정)
 RS : (reset 우선 쌍안정)
 R_Trig : (오르기 검출)
 F_Trig : (내리기 검출)
 SEMA : (semafo)

카운터 :CTU: (up 카운터)
CTD: (down 카운터)
CTUD: (up down 카운터)

타임 :TON: (on delay time)
TOF: (off delay time)
TP : (pulse time)

정수 아나로그 :CMP: (완전 비교)
StackInt: (정수 analog Stack)

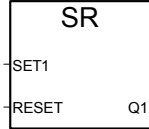
정수 아나로그 :AVERAGE: (평균치)
HYSTER: (hysteresis)
LIM_ALARM: (hysteresis 를 가진 상한 / 하한 limit)
INTEGRAL: (적분)
DERIVATE: (미분)

신호 발신:BLINK: (boolean형 신호 BLINK)
SIG GEN: (신호 발생기)

주의 : 새로운 C 언어 FunctionBlock 이 만들어진 경우에도 FBD 에서

_____ call 가능합니다.

SR (set 우선 쌍안정)



인수:

SET1	boolean 형	TRUE 이면 Q1 은 TRUE (set 우선함)
RESET	boolean 형	TRUE 이면 Q1 을 FALSE 에 reset
Q1	boolean 형	boolean 형 메모리 상태

설명:

set 우선 쌍안정 : 이하의 테이블 참조

Set1	Reset	Q1	Result Q1
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

(*SR"블록을 이용한 F B D 프로그램*)

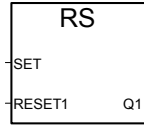
(*등가한 S T 언어 : SR1 은 SR 블록 instance *)

```
SR1((auto_mode & start_cmd), stop_cmd);
command := SR1.Q1;
```

(* 등가한 I L 언어 : *)

```
LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
LD      SR1.Q1
ST      command
```

RS (reset 우선 쌍안정)



인수:

SET boolean 형 TRUE 로 Q1 을 TRUE 에 set
RESET1 boolean 형 TRUE 로 Q1 을 FALSE 에 reset (reset 우선함)
Q1 boolean 형 boolean 형 메모리 상태

설명:

reset 우선 쌍안정 : 이하의 테이블 참조

Set	Reset1	Q1	Result Q1
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	0
1	1	1	0

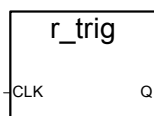
(*등가한 S T 언어: RS1 은 RS 블록 instance *)

```
RS1(start_cmd, (stop_cmd OR alarm));
command := RS1.Q1;
```

(* 등가한 I L 언어: *)

```
LD                    start_cmd
ST                    RS1.set
LD                    stop_cmd
OR                    alarm
ST                    RS1.reset1
CAL                   RS1
LD                    RS1.Q1
ST                    command
```

R_TRIG (오르기 검출)



인수:

CLK	boolean 형	
Q	boolean 형	TRUE : CLK 이 FALSE 가 TRUE 로 올랐을 때 FALSE : 이외의 경우

설명:

boolean 형 변수의 오르기 검출 (1 사이클 전 값과 비교)

(*R_TRIG"블록을 이용한 F B D 프로그램*)

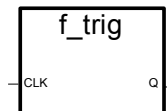
— (*등가한 S T 언어: R_TRIG1 は R_TRIG 블록 instance *)

```
R_TRIG1(cmd);
nb_edge := ANA(R_TRIG1.Q) + nb_edge;
```

(*등가한 I L 언어: *)

```
LD      cmd
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ANA
ADD     nb_edge
ST      nb_edge
```

F_TRIG (내리기 검출)



인수:

CLK	boolean 형	
Q	boolean 형	TRUE : CLK 이 TRUE 에서 FALSE 로 내렸을 때 FALSE : 이외의 경우

설명:

boolean 형 변수의 내리기 검출은 (1 사이클 전 값과 비교해서)

(*F_TRIG"블록을 이용한 F B D 프로그램*)

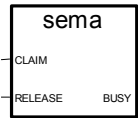
— (*등가한 S T 언어: F_TRIG1 은 F_TRIG 블록 instance *)

```
F_TRIG1(cmd);
nb_edge := ANA(F_TRIG1.Q) + nb_edge;
```

(* 증가한 I L 언어: *)

```
LD      cmd
ST      F_TRIG1.clk
CAL     F_TRIG1
LD      F_TRIG1.Q
ANA
ADD     nb_edge
ST      nb_edge
```

SEMA (SEMA fo)



인수:

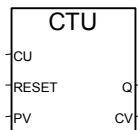
CLAIM	boolean 형	"테스트와 set" 커맨드
RELEASE	boolean 형	SEMA fo release
BUSY	boolean 형	SEMA fo 상태

설명:

(* "x" 는 boolean 형 변수로 FALSE 에 초기화*)

```
busy := x;
If claim Then
    x := True;
Else
    If release Then
        busy := False;
        x := False;
    End_if;
End_if;
```

CTU (up 카운터)



인수:

CU	boolean 형	카운터 입력 (CU 가 TRUE 에서 카운터)
RESET	boolean 형	reset 커맨드 (우선)
PV	정수형	최대 설정치
Q	boolean 형	over follow : CV = PV 에서 T R U E
CV	정수형	카운터 결과

주의 : C T U는 입력C U의 오르기, 내리기를 검출할 수 없습니다. 반드시 "R_TRIG" 또는 "F_TRIG" 을 사용해서 pulse 카운터를 만들 필요가 있습니다.

설명:

카운터 수 (정수형) 는 0 에서 1 씩 increment

(* "CTU"블록을 이용한 F B D 프로그램*)

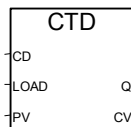
—
(*등가한 S T 언어: R_TRIG1 은 R_TRIG 블록 instance CTU1 은 CTU 블록 instance *)

```
CTU1(R_TRIG1(command),NOT(auto_mode),100);
overflow := CTU1.Q;
result := CTU1.CV;
```

(* 등가한 I L 언어: *)

```
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 (down 카운터)



인수:

CD	boolean 형	카운터 입력 (CD 가 TRUE 일 때 카운터 down)
LOAD	boolean 형	로드 커맨드 (우선) (CV = PV 에서 LOAD 가 TRUE)
PV	정수형	초기 카운터 값
Q	boolean 형	종료 신호 : CV = 0 에서 TRUE
CV	정수형	카운터 결과

주의 : C T D는 입력 C D의 오르기, 내리기를 검출할 수 없습니다. 반드시 "R_TRIG" 또는 "F_TRIG"을 사용해서 pulse 카운터를 만들 필요가 있습니다.

설명:

카운터 수 (정수형) 는 PV 값에서 1 씩 감소

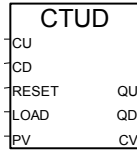
(* "CTD" 블록을 이용한 F B D 프로그램 *)

—
(* 등가한 S T 언어: F_TRIG1 은 F_TRIG 블록 instance CTD1 은 CTD 블록 instance *)
CTD1(F_TRIG1(command),load_cmd,100);
underflow := CTD1.Q;
result := CTD1.CV;

(* 등가한 I L 언어: *)

```
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 (updown 카운터)



인수:

CU	boolean 형	updown 카운터 입력 (CU 가 TRUE 에서 카운터)
CD	boolean 형	down 카운터 입력 (CD 가 TRUE 에서 카운터)
RESET	boolean 형	reset 커맨드 (우선) (R 이 TRUE 일 때 CV = 0)
LOAD	boolean 형	LOAD 커맨드(LD 가 TRUE 일때 CV = PV)
PV	정수형	최대 카운터 값
QU	boolean 형	over follow : TRUE when CV = PV
QD	boolean 형	under follow : TRUE when CV = 0
CV	정수형	카운터 결과

주의 : C T U D는 입력C U, C D의 오르기, 내리기, 내리기를 검출할 수 없습니다. 반드시 "R_TRIG" 또는 "F_TRIG"을 사용해서 pulse 카운터를 만들 필요가 있습니다.

설명:

카운터 수 (정수형) 는 0 에서 PV 값까지 1 씩 increment, 또는
카운터 수 (정수형) 는 현재 값에서 1 씩 감소

(*CTUD"블록을 이용한 F B D 프로그램*)

(*등가한 S T 언어: R_TRIG1 과 R_TRIG2 는 R_TRIG 블록 instance, CTUD1 은 CTUD 블록 instance*)

```

CTUD1(R_TRIG1(add_elt), R_TRIG2(sub_elt), reset_cmd, load_cmd,100);
full := CTUD1.QU;
empty := CTUD1.QD;
nb_elt := CTUD1.CV;
  
```

(* 등가한 I L 언어: *)

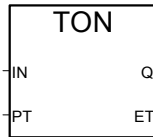
```

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       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 (on derail time)



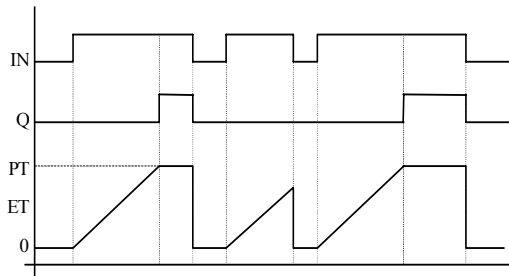
인수:

IN	boolean 형	오르기 검출로 타임값 increment 내리기 검출로 타임 값 정지, reset
PT	타임형	타임 up 설정치
Q	boolean 형	TRUE : 타임 up 설정치가 경과
ET	타임형	현재 경과 타임값

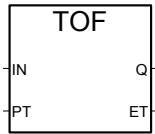
설명:

내부 값을 지정된 값 (PT) 까지 increment

타임 차트:



TOF (off derail time)



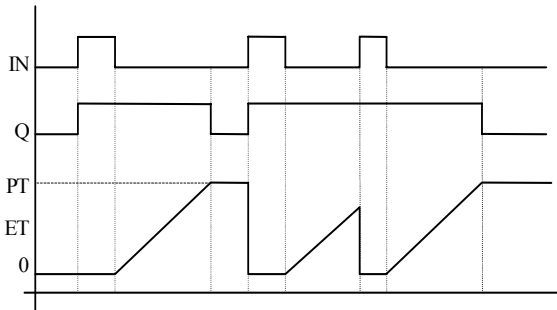
인수:

IN	boolean 형	내리기 검출로 타임값 increment 오르기 검출로 타임 값 정지, reset
PT	타임형	설정 타임 값
Q	boolean 형	TRUE : 설정 타임 값 까지 경과하지 않음
ET	타임형	현재 경과 타임값

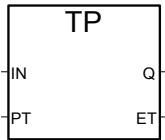
설명:

내부 타임 값을 지정된 값 (PT) 까지 increment

타임 차트:



TP (pulse time)



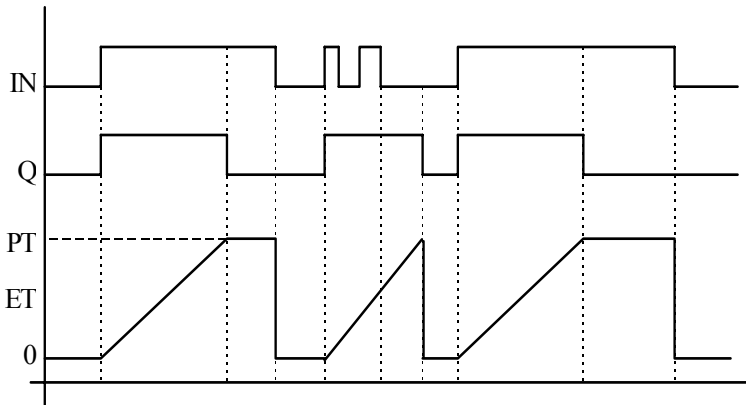
인수:

IN	boolean 형	오르기 검출에서 타임 값의 increment (increment 중이 아니면) 。 만약, FALSE 상태에서 타임 값이 경과한 때는 타임 값을 reset. 타임 increment 중은 IN 의 변화는 무시됩니다.
PT	타임형	설정 타임값
Q	boolean 형	TRUE : 타임이 increment 중
ET	타임형	현재 경과 타임값

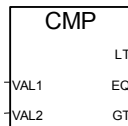
설명:

내부 타임 값을 지정된 값 (PT) 까지 increment

타임 차트:



CMP (비교)



인수:

VAL1	정수형	부호부치기 정수
VAL2	정수형	부호부치기 정수
LT	boolean 형	TRUE : VAL1 < VAL2
EQ	boolean 형	TRUE : VAL1 = VAL2

GT boolean 형 TRUE : VAL1 > VAL2

설명:

두개의 정수형 값 비교 : 3 개의 결과 중에 택합니다. (< , = , >)

(*CMP"블록을 이용한 F B D 프로그램*)

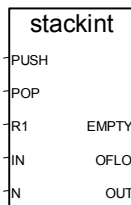
—
(*등가한 S T 언어: CMP1 은 CMP 블록 instance 명 *)

```
CMP1(level, max_level);
pump_cmd := CMP1.LT OR CMP1.EQ;
alarm := CMP1.GT AND NOT(manual_mode);
```

(* 등가한 I L 언어: *)

```
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
ANDN                  manual_mode
ST                    alarm
```

STACKINT



인수:

PUSH	boolean 형	push 커맨드 : 오르기 검출에 따라 IN 값을 stack top 에 추가
POP	boolean 형	pop 커맨드 : 오르기 검출에 따라 stack top (마지막에 push 된 값) 을 삭제
R1	boolean 형	stack 을 공백 상태에 reset 한다.
IN	정수형	push 되는 값

N	정수형	stack 사이즈
EMPTY	boolean 형	TRUE : stack 이 공백일 때
OFLO	boolean 형	TRUE : stack 이 가득참
OUT	정수형	stack top 의 값

설명:

정수형 값 stack 의 관리

STACKINT FunctionBlock 은 P U S H , P O P 의 입력에 대해 오르기 검출 기능도 포함하고 있습니다. **최대 stack 사이즈는 1 2 8** 입니다. Stack 사이즈 **N 은 1 에서 1 2 8** 의 값이어야만 합니다.

OFLO 값은 일단 reset 된 후에만 유효. (즉, R1 는 한번 TRUE 에 set 된 후에 FALSE 되돌아가야 합니다.)

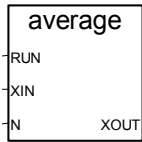
(* "STACKINT" 블록을 이용한 F B D 프로그램 : error management *)

—
(* 등가한 S T 언어: STACKINT1 は STACKINT 블록 instance *)

```
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;
```

(* 등가한 I L 언어: *)

```
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 (평균값)

인수:

RUN	boolean 형	TRUE : run / FALSE : reset
XIN	실수형	실수치 입력
N	정수형	평균을 내야할 샘플수 (매 사이클 1 입력)
XOUT	실수형	XIN 값의 사이클 개수 (N 개) 의 평균값

설명:

매 사이클 XIN 을 집어 넣고, 과거 샘플값 (샘플수 N) 의 평균치를 구합니다. 최신의 N 개 데이터가 축적되어 있습니다.

센플 N 은 최대 1 2 8 입니다. "RUN" 커맨드가 FALSE 일때 (reset 모드) , 출력치는 입력치와 같은 값이 됩니다.

축적된 센플이 N 이 되면, 처음에 축적된 값은 삭제됩니다.

(* "AVERAGE" 블록을 이용한 F B D 프로그램 *)

—
(* 등가한 S T 언어 : AVERAGE1 은 AVERAGE 블록 instance *)

```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
ave_value := AVERAGE1.XOUT;
```

(* 등가한 I L 언어: *)

```
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 (hysteresis)



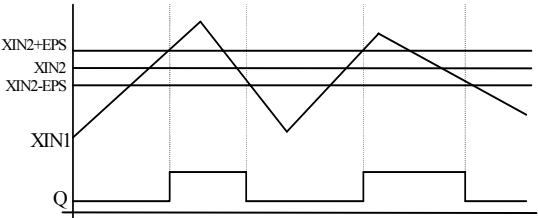
인수:

XIN1	실수형	실수
XIN2	실수형	실수 (XIN1 이 XIN2+EPS 을 넘든지 테스트된다.)
EPS	실수형	hysteresis 값 (> 0)
Q	boolean 형	TRUE : XIN1, XIN2+EPS 을 한번은 넘고, 아직 XIN2- EPS 를 하회하고 있지 않은 때

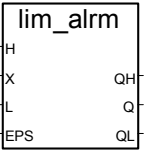
설명:

실수치의 상한 값을 위한 hysteresis

타임 차트 예:



LIM_ALARM



인수:

H	실수형	상한 설정치
X	실수형	입력치
L	실수형	하한 설정치
EPS	실수형	hysteresis 값 (> 0)

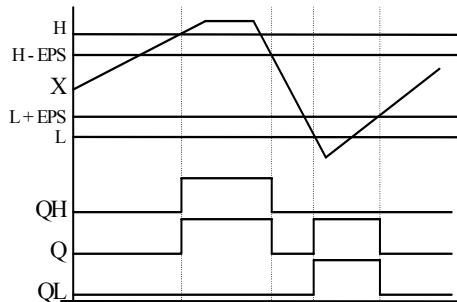
QH	boolean 형	상한 alarm, TRUE : X 가 상한값 H 를 넘은 다음에 H-EPS 를 하회 할 때 까지
Q	boolean 형	alarm, TRUE : QH 또는 QL
QL	boolean 형	하한 alarm, TRUE : X 가 하한값 L 을 하회한 다음에, L+EPS 를 넘을 때까지

설명:

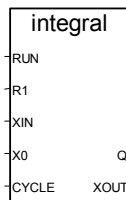
실수치에 대해 상하한에 대한 hysteresis

hysteresis 가 상한, 하한 리미터에 주어집니다. 상하한에 사용되는 hysteresis 데이터 양은 E P S parameter 의 절반이 됩니다.

타임 차트 :



INTEGRAL (적분)



인수:

RUN	boolean 형	모드, TRUE=적분 / FALSE=홀드
R1	boolean 형	표서 reset
XIN	실수형 입력치	
X0	실수형	초기값

CYCLE	타임형	샘플링 주기
Q	boolean 형	R1 의 반전
XOUT	실수형	적분 결과

설명:

실수치의 적분

"**CYCLE**" parameter 값이 I S a G R A F 사이클 타임보다도 짧은 경우는 샘플링 간격은 사이클 타임에 맞춰집니다.

(* "INTEGRAL" 블록을 이용한 F B D 프로그램 *)

~

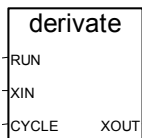
(* 등가한 S T 언어: INTEGRAL1 은 INTEGRAL 블록 instance *)

```
INTEGRAL1(manual_mode, NOT(manual_mode), sensor_value, init_value, t#100ms);
controlled_value := INTEGRAL1.XOUT;
```

(* 등가한 I L 언어: *)

```
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 (미분)



인수:

RUN	boolean 형	모드 , TRUE=run / FALSE=reset
XIN	실수형	입력치
CYCLE	타임형	샘플링 주기

XOUT 실수형 미분 결과

설명:

실수치의 미분

"CYCLE" parameter 값이 I S a G R A F 사이클 타임보다도 짧은 경우는 샘플링간격은 사이클 타임에 맞춰집니다.

(* "DERIVATE"블록을 이용한 F B D 프로그램 *)

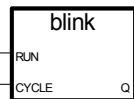
—
(* 등가한 S T 언어: DERIVATE1 은 DERIVATE 블록 instance *)

```
DERIVATE1(manual_mode, sensor_value, t#100ms);
derivated_value := DERIVATE1.XOUT;
```

(* 등가한 I L 언어: *)

```
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 (blink)



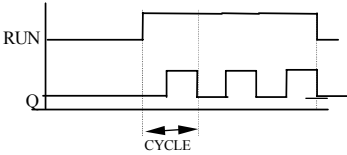
인수:

RUN	boolean 형	모드, TRUE= blink / FALSE=출력을 reset
CYCLE	타임형	blink 주기
Q	boolean 형	blink 출력 번호

설명:

blink 신호 생성

타임 차트:



SIG_GEN



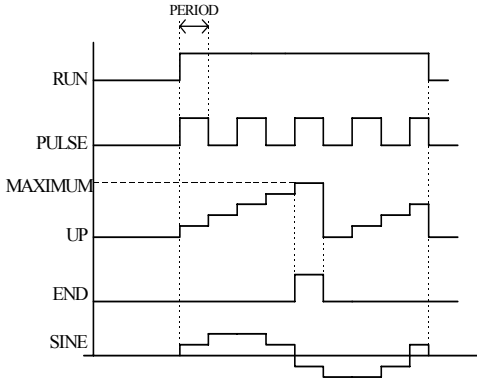
인수:

RUN	boolean 형	모드 TRUE=run / FALSE=reset
PERIOD	타입형	샘플시간 (pulse 폭)
MAXIMUM	정수형	최대 카운터 값
PULSE	boolean 형	각 샘플시간마다 반전
UP	정수형	각 샘플시간마다 up 카운터
END	boolean 형	TRUE : up 카운터 종료때
SINE	정수형	사인파형 신호 (파형 반주기는 up 카운터 시간)

설명:

각종 신호의 동시 발생 (pulse 파형, up 카운터 , 사인파형) 카운터가 최대값이 되면, 0 에서 restart 합니다. END 신호는 1 PERIOD 분만 TRUE 가 됩니다.

타임 차트:



B.9.3 표준 functions

I S a G R A F 에서 support 되고 있는 표준 Function 을 이하에 나타냅니다. 이들은 표준으로 준비되어 있기 때문에 I S a G R A F library 에 등록할 필요가 없습니다.

- 계산식 :ABS (절대값)
EXPT (지수관수) , POW
LOG (상용대수)
SQRT (평방근)
TRUNC (소수점이하 잘라 버리기)
- 삼각함수 :ACOS (어코사인) , ASIN (어사인)
ATAN (어탄젠트)
COS (코사인) , SIN (사인)
TAN (탄젠트)
- 레지스터 콘트롤 :ROL (좌회전) , ROR (우회전)
SHL (좌 Shift) , SHR (우 Shift)
- :MIN (최소값) , MAX (최대값)
LIMIT (상하근)
MOD (잉여산)
MUX4 (multiplexer (4 입력))
MUX8 (multiplexer (8 입력))
SEL (binary 선택)
ODD (홀수 parity)

RAND (random 값)

데이터 변환 :ASCII (ascii 변수)

CHAR (캐릭터변수)

문자열 조작 :MLEN (문자열장) = LEN

DELETE (문자열 삭제)

INSERT (문자열 삽입)

FIND (문자열 검색)

REPLACE (문자열 치환)

LEFT (좌측 문자열 추출) , MID (문자열 추출)

RIGHT (우측 문자열 추출)

DAY_TIME (일시)

배열 조작 :ARCREATE (배열 작성)

ARREAD (배열 요소 읽어 내기)

ARWRITE (배열 요소 적어 내기)

Binary 파일 관리 :F_ROPEN (읽어 내기 모드로 파일 오픈)

F_WOPEN (써넣기 모드로 파일 오픈)

F_CLOSE (파일 close)

F_EOF (파일 최종 검출)

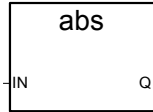
FA_READ (파일에서 정수값 읽어 내기)

FA_WRITE (파일로 정수값 적어 넣기)

FM_READ (파일에서 문자열 읽어 내기)

FM_WRITE (파일로 문자열 적어 넣기)

ABS (절대값)



인수:

IN	실수형	실수치 입력
Q	실수형	절대값

설명:

실수의 절대값 계산

(* "ABS" 블록을 이용한 F B D 프로그램 *)

—

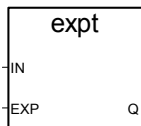
(* 등가한 S T 언어: *)

over := (ABS (delta) > range);

(* 등가한 I L 언어: *)

```
LD      delta
ABS
GT      range
ST      over
```

EXPT (지수관수)



인수:

IN	실수형	부호 부치기 실수 (base 부)
EXP	정수형	정수 (exponent 부)
Q	실수형	(IN EXP)

설명:

실수의 지수관수 : (base exponent) 'base' 가 최초의 인수, 'exponent' 가 두번째 인수로
정수

```
(**EXPT**)
```

...

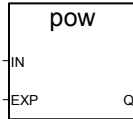
(*증가한 S T 언어:*)

```
tb_size := ANA (EXPT (2.0, range) );
```

(* 증가한 I L 언어: *)

```
LD          2.0
EXPT        range
ANA
ST          tb_size
```

POW



인수:

IN	실수형	실수치 (base 부)
EXP	실수형	실수치 (exponent 부)
Q	실수형	(IN EXP)

1.0 : IN<> 0.0, EXP= 0.0
 0.0 : IN= 0.0 , EXP < 0.0
 0.0 : IN =0.0 , EXP = 0.0
 0.0 : IN < 0

설명:

실수의 지수관수 : (base exponent) 'base' 가 최초의 인수, 'exponent' 두번째 인수로 정수

```
(**"POW" 블록을 이용한 F B D 프로그램 *)
```

...

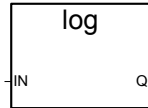
(*증가한 S T 언어:*)

```
result := POW (xval, power);
```

(* 등가한 I L 언어: *)

LD	xval
POW	power
ST	result

LOG (상용대수)



인용:

IN	실수형	실수치 (> 0)
Q	실수형	입력 값의 상용대수(base 10)

tjfaud:

실수 입력의 상용대수(base 10)

(* "LOG"블록을 이용한 F B D 프로그램 *)

—

(* 등가한 S T 언어: *)

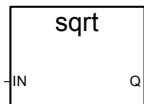
```

xpos := ABS (xval);
xlog := LOG (xpos);
  
```

(* 등가한 I L 언어: *)

LD	xval
ABS	
ST	xpos
LOG	
ST	xlog

SQRT (평방근)



인수:

IN	실수형	실수치 (>= 0)
-----------	-----	--------------

Q 실수형 입력치의 평방근

설명:

실수 입력치 평방근

(*SQRT"블록을 이용한 F B D 프로그램*)

—

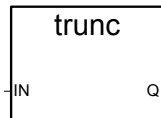
(*등가한 S T 언어:*)

xpos := ABS (xval);
xroot := SQRT (xpos);

(* 등가한 I L 언어: *)

LD xval
ABS
ST xpos
SQRT
ST xroot

TRUNC (소수점 이하 버리기)



인수:

IN	실수형	실수치
Q	실수형	IN>0, 정수부분 IN<0, 정수부분

설명:

실수치의 정수부분

(*TRUNC"블록을 이용한 F B D 프로그램*)

—

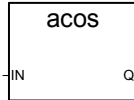
(*등가한 S T 언어:*)

result := TRUNC (+2.67) + TRUNC (-2.0891);
(* means: result := 2.0 + (-2.0) := 0.0; *)

(* 등가한 I L 언어: *)

LD	2.67
TRUNC	
ST	temporary (* temporary result of first TRUNC *)
LD	-2.0891
TRUNC	
ADD	temporary
ST	result

ACOS (A 코사인)



인수:

IN	실수형	[-1.0 .. +1.0]
Q	실수형	입력치 어코사인[0.0 .. PI] = 0.0 : 부정함 입력치일 때

설명:

실수치 어코사인의 계산

(* "COS" 과 "ACOS" 블록을 이용한 F B D 프로그램 *)

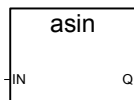
(* 등가한 S T 언어: *)

cosine := COS (angle);
result := ACOS (cosine); (* result is equal to angle *)

(* 등가한 I L 언어: *)

LD	angle
COS	
ST	cosine
ACOS	
ST	result

ASIN (A 사인)



인수:

IN	실수형	$[-1.0 \dots +1.0]$
Q	실수형	입력치 어사인 $[-\pi/2 \dots +\pi/2]$ = 0.0 : 부정할 입력일 때

설명:

실수 입력 어사인의 계산

(*SIN" 과 "ASIN"블록을 이용한 F B D 프로그램*)

—

(*등가한 S T 언어:*)

sine := SIN (angle);

result := ASIN (sine); (* result is equal to angle *)

(* 등가한 I L 언어: *)

LD angle

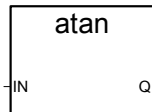
SIN

ST sine

ASIN

ST result

ATAN (A 탄젠트)



인수:

IN	실수형	실수치
Q	실수형	입력치 어탄젠트 $[-\pi/2 \dots +\pi/2]$ = 0.0 : 부정할 입력치일 때

설명:

실수치 어탄젠트의 계산

(*TAN" 과 "ATAN" 블록을 이용한 F B D 프로그램 *)

—

(*등가한 S T 언어:*)

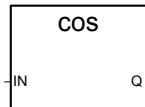
tangent := TAN (angle);

result := ATAN (tangent); (* result is equal to angle*)

(* 등가한 I L 언어: *)

```
LD      angle
TAN
ST      tangent
ATAN
ST      result
```

COS (코사인)



인수:

IN	실수형	실수치
Q	실수형	실수치 코사인 [-1.0 .. +1.0]

설명:

실수치 코사인의 계산

(* "COS" 와 "ACOS" 블록을 이용한 F B D 프로그램*)

—

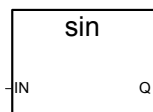
(* 등가한 S T 언어: *)

```
cosine := COS (angle);
result := ACOS (cosine); (* result is equal to angle *)
```

(* 등가한 I L 언어: *)

```
LD      angle
COS
ST      cosine
ACOS
ST      result
```

SIN (사인)



인수:

IN	실수형	실수치
Q	실수형	입력치 사인[-1.0 .. +1.0]

설명:

실수치 사인의 계산

(* "SIN" 과 "ASIN" 블록을 이용한 F B D 프로그램 *)

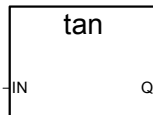
(* 등가한 S T 언어: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```

(* 등가한 I L 언어: *)

```
LD      angle
SIN
ST      sine
ASIN
ST      result
```

TAN (탄젠트)



인수:

IN	실수형	PI/2 이외
Q	실수형	입력치 탄젠트 = 1E+38 : 부정함 입력에 대해서

설명:

실수치 탄젠트의 계산

(* "TAN" 과 "ATAN" 블록을 이용한 F B D 프로그램 *)

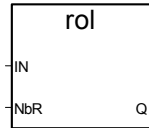
(* 등가한 S T 언어: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle *)
```

(* 등가한 I L 언어: *)

```
LD      angle
TAN
ST      tangent
ATAN
ST      result
```

ROL (좌회전)



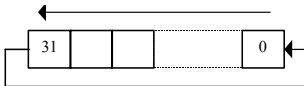
인수:

IN	정수형	정수치
NbR	정수형	회전 시키는 bit 수 [1..31]
Q	정수형	좌회전한 결과

변화 없음 : nb_rotation <= 0 일때

설명:

정수치 bit 좌회전. 회전은 32 bit 에서 됩니다.



(* "ROL" 블록을 이용한 F B D 프로그램 *)

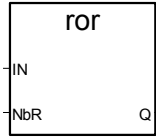
(* 등가한 S T 언어: *)

```
result := ROL (register, 1);
(* register = 2#0100_1101_0011_0101*)
(* result  = 2#1001_1010_0110_1010*)
```

(* 등가한 I L 언어: *)

```
LD      register
ROL     1
ST      result
```

ROR (우회전)

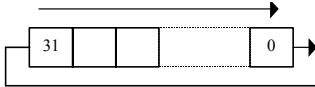


인수:

IN	정수형	정수치
NbR	정수형	회전시키는 bit 수 [1..31]
Q	정수형	회전한 결과
변화 없음 : nb_rotation <= 0 일때		

설명:

정수치 bit 우회전. 회전은 32 bit 으로 됩니다.



(*"ROR" 블록을 이용한 F B D 프로그램*)

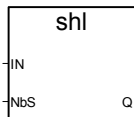
(*등가한 S T 언어:*)

```
result := ROR (register, 1);
(* register = 2#0100_1101_0011_0101 *)
(* result   = 2#1010_0110_1001_1010 *)
```

(* 등가한 I L 언어: *)

```
LD      register
ROR     1
ST      result
```

SHL (좌 Shift)



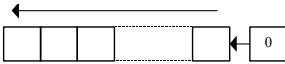
인수:

IN	정수형	정수치
NbS	정수형	bit 시프트 수 [1..31]

Q	정수형	좌 시프트한 결과
		변화 없음 : nb_shift <= 0 일때
		(LSB 에는 0 이 삽입됩니다.)

설명:

정수치 bit 표현으로 좌 Shift 합니다. 3 2 bit 단위로 합니다.



(* "SHL" 블록을 이용한 F B D 프로그램 *)

-

(* 등가한 S T 언어: *)

result := SHL (register, 1);

(* register = 2#0100_1101_0011_0101 *)

(* result = 2#1001_1010_0110_1010 *)

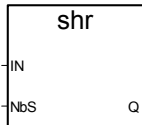
(* 등가한 I L 언어: *)

LD register

SHL 1

ST result

SHR (우 Shift)

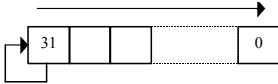


인수:

IN	정수형	정수치
NbS	정수형	bit 시프트 수 [1..31]
Q	정수형	우 시프트한 결과
		변화 없음 : nb_shift <= 0 일때
		(MSB 에는 같은 값이 copy 됩니다.)

설명:

정수치 bit 표현으로 좌 시프트합니다. 3 2 bit 단위로 합니다.



(* "SHR" 블록을 이용한 F B D 프로그램 *)

(* 등가한 S T 언어: *)

result := SHR (register, 1);

(* register = 2#1100_1101_0011_0101 *)

(* result = 2#1110_0110_1001_1010 *)

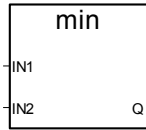
(* 등가한 I L 언어: *)

LD register

SHR 1

ST result

MIN (최소값)



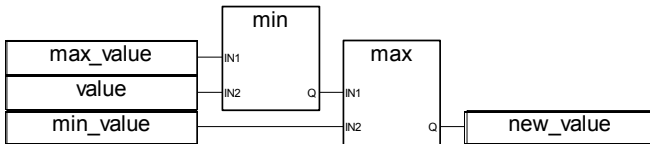
인수:

IN1	정수형	정수치
IN2	정수형	정수치
Q	정수형	두개의 입력 최소값

설명:

두개의 정수치의 최소값

(* "MIN" 과 "MAX" 블록을 이용한 F B D 프로그램 *)



(* 등가한 S T 언어: *)

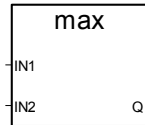
new_value := MAX (MIN (max_value, value), min_value);

(* bounds the value to the [min_value..max_value] set *)

(*등가한 I L 언어: *)

LD	max_value
MIN	value
MAX	min_value
ST	new_value

MAX (최대값)



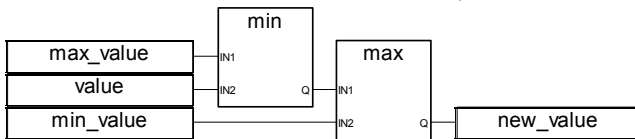
인수:

IN1	정수형	정수치
IN2	정수형	정수치
Q	정수형	두개의 입력 최대값

설명:

2 두개의 정수치 최대값

(* "MIN" 과 "MAX" 블록을 이용한 F B D 프로그램 *)



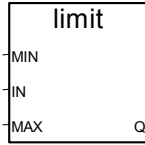
(*등가한 S T 언어:*)

new_value := MAX (MIN (max_value, value), min_value);
 (* bounds the value to the [min_value..max_value] set *)

(* 등가한 I L 언어: *)

LD	max_value
MIN	value
MAX	min_value
ST	new_value

LIMIT (상하근)



인수:

MIN	정수형	최소값 설정치
IN	정수형	정수 입력치
MAX	정수형	최대값 설정치
Q	정수형	입력치 : 입력치가 범위 내 (최대값 - 최소값) 일 때 최대값 : 입력치가 최대값을 넘었을 때 최소값 : 입력치가 최소값을 하회한 때

설명:

입력치에 상하근를 설치합니다. 입력치가 최대값과 최소값 사이일때는 입력치 그대로.
최대값을 넘었을 때는 최대값에, 최소값을 하회한 때는 최소값이 됩니다.

(* "LIMIT"블록을 이용한 F B D 프로그램 *)

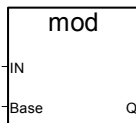
—
(* 등가한 S T 언어: *)

```
new_value := LIMIT (min_value, value, max_value);
(* bounds the value to the [min_value..max_value] set *)
```

(* 등가한 I L 언어: *)

```
LD      min_value
LIMIT   value, max_value
ST      new_value
```

MOD (나머지)



인수:

IN	정수형	입력치
Base	정수형	입력치 (> 0)
Q	정수형	잉여산의 계산

-1 : Base <= 0 일때

설명:

입력치 (IN)를 Base 로 나눈때의 잉여

(*MOD" 블록을 이용한 F B D 프로그램*)

(*등가한 S T 언어:*)

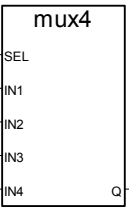
division_result := (value / divider); (* integer division *)

rest_of_division := MOD (value, divider); (* rest of the division *)

(* 등가한 I L 언어: *)

```
LD      value
DIV     divider
ST      division_result
LD      value
MOD     divider
ST      rest_of_division
```

MUX4 (multiplexer)



인수:

SEL	정수형	select 입력치 [0..3]
IN1..IN4	정수형	입력치
Q	정수형	= IN1 : SEL = 0 일 때 = IN2 : SEL = 1 일 때 = IN3 : SEL = 2 일 때 = IN4 : SEL = 3 일 때 = 0 : SEL 이 상기 이외일 때

설명:

4 점 입력용 multiplexer : 4 개의 정수 입력치 중에서 하나를 선택

(* "MUX4" 블록을 이용한 FBD 프로그램 *)

-

(*등가한 S T 언어:*)

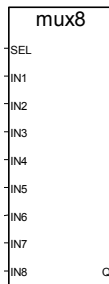
range := MUX4 (choice, 1, 10, 100, 1000);

(* 네개의 입력에서 하나를 선택한다. choice 가 1 일때는 range 는 10 *)

(* 등가한 I L 언어: *)

LD choice
MUX4 1,10,100,1000
ST range

MUX8 (multiplexer)



인수:

SEL	정수형	select 입력치 [0..7]
IN1..IN4	정수형	입력치
Q	정수형	= IN1 : SEL = 0 일때 = IN2 : SEL = 1 일때 = IN3 : SEL = 2 일때 = IN8 : SEL = 7 일때 = 0 : SEL 가 상기 이하일때

설명:

8 점 입력용 multiplexer : 여덟개 정수입력치 중에서 하나를 선택

(* "MUX8" 블록을 이용한 F B D 프로그램 *)

-

(*등가한 S T 언어:*)

range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);

(* 여덟개 입력에서 하나를 선택한다. choice 가 3 일때는 range 는 50 *)

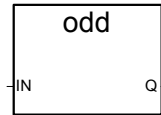
(* 등가한 I L 언어: *)

LD choice

MUX8 1,5,10,50,100,500,1000,5000

ST range

ODD (홀수 verity)



인수:

IN	정수형	정수 입력
Q	boolean 형	TRUE : 입력치가 홀수일 때 FALSE : 입력치가 짝수일 때

설명:

정수 입력치 verity 체크 : 홀수인지 짝수인지 판단

(*"ODD"블록을 이용한 F B D 프로그램*)

-
(*등가한 S T 언어:*)

If Not (ODD (value)) Then Return; End_if;

value := value + 1;

(* makes value always even *)

(* 등가한 I L 언어: *)

LD value

ODD

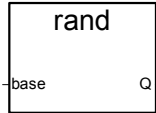
RETNC

LD value

ADD 1

ST value

RAND (random 값)



인수:

base	정수형	random 값의 최대값
Q	정수형	random 값 발생 [0..base-1]

설명:

정수값에서 주어진 범위에서 random 값 발생

(* "RAND" 블록을 이용한 F B D 프로그램 *)

```

--
(* 등가한 S T 언어: *)
selected := MUX4 ( RAND (4), 1, 4, 8, 16 );
(* 발생하는 random 값에 따라 네개의 입력에서 하나를 임의로 선택한다.
RAND 에서 발생하는 수치는 [0..3], 결과로서 MUX4 의 'selected' 에는 이하의 출력이 나옵니다.
1 : RAND 출력이 0,
4 : RAND 출력이 1
8 : RAND 출력이 2
16 : RAND 출력이 3
*)

```

(* 등가한 I L 언어: *)

```

LD      4
RAND
MUX4    1,4,8,16
ST      selected

```

SEL (vinally 선택)



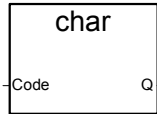
인수:

SEL	boolean 형	선택용
IN1, IN2	정수형	정수치

(* 등가한 I L 언어: *)

LD message
ASCII 1
ST FirstChr

CHAR (캐릭터 변환)



인수:

Code	정수형	캐릭터 ASCII 코드 [0 .. 255]
Q	문자열형	캐릭터 문자

ASCII 코드 (Code) 에 대한 캐릭터
Code 는 256 잉여산을 사용합니다.

설명:

주어진 ASCII 코드에 대한 캐릭터

(* "CHAR" 블록을 이용한 F B D 프로그램 *)

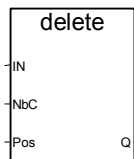
—
(* 등가한 S T 언어: *)

```
Display := CHAR ( value + 48 );
(* value is in set [0..9] *)
(* 48 is the ascii code of '0' *)
(* result is one character string from '0' to '9' *)
```

(* 등가한 I L 언어: *)

LD value
ADD 48
CHAR
ST Display

DELETE (문자열 삭제)



인수:

IN	문자열	NULL 이외의 문자열
NbC	정수형	문자열에서 삭제하는 문자수
Pos	정수형	삭제하는 최초의 문자열 장소를 나타내는 번호 (최초의 문자는 Pos = 1)
Q	문자열형	지정 문자열이 삭제되고 수정된 문자열 =빈 문자열 : Pos < 1 =최초의 문자 : Pos > IN 의 문자열장 =최초의 문자 : NbC <= 0

설명:

입력 문자열에서 지정되어 연속적인 문자열을 삭제

(* "DELETE" 블록을 이용한 F B D 프로그램 *)

(* 등가한 S T 언어: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH' *)
sub_string := DELETE (complete_string, 4, 3); (* sub_string is 'ABGH' *)
```

(* 등가한 I L 언어: *)

```
LD      'ABCD'
ADD     'EFGH'
ST      complete_string
DELETE  4,3
ST      sub_string
```

FIND (문자열 검색)



인수:

In	문자열형	입력 문자열
Pat	문자열형	검색하는 문자열 버튼
Pos	정수형	= 0 : 문자열 버튼이 발견되지 않을 때 = 검색하는 문자열이 최초로 발견된 위치 (첫째 문자가 있으면 1)

대소문자 구별 있음

설명:

문자열 안에서 지정한 문자열을 검색해서, 발견된 경우는 그 장소를 되돌립니다.

(* "FIND" 블록을 이용한 F B D 프로그램 *)

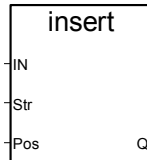
—
(* 등가한 S T 언어: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH' *)
found := FIND (complete_string, 'CDEF'); (* found is 3 *)
```

(* 등가한 I L 언어: *)

```
LD      'ABCD'
ADD     'EFGH'
ST      complete_string
FIND    'CDEF'
ST      found
```

INSERT (문자열 삽입)



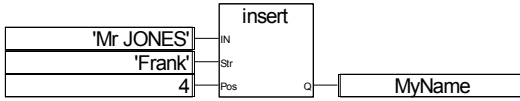
인수:

IN	문자열형	기본이 되는 문자열
Str	문자열형	삽입되는 문자열
Pos	정수형	삽입 장소를 지정하는 번호, 삽입 문자열은 번호 앞에 됩니다. (> 1)
Q	문자열형	문자열이 삽입된 결과의 문자열 = 빈 문자열 : Pos <= 0 = 두개의 문자열의 연결 : Pos 가 IN 문자열장의 길이를 넘은 경우

설명:

기본 문자열에 지정된 문자열을 삽입해서 새로운 문자열이 생성

(* "INSERT" 블록을 이용한 F B D 프로그램 *)



(* 등가한 S T 언어: *)

MyName := INSERT ('Mr JONES', 'Frank ', 4);

(* MyName : 'Mr Frank JONES' *)

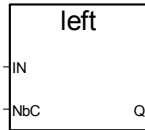
(* 등가한 I L 언어: *)

LD 'Mr JONES'

INSERT 'Frank', 4

ST MyName

LEFT (좌측 문자열 추출)



인수:

IN	문자열형	공백 없는 문자열
NbC	정수형	줄출되는 문자수 (<= IN 문자열장)
Q	문자열형	IN 문자열 좌측 문자열 (문자열의 길이는 NbC) = 빈 문자열 : NbC <= 0 = 완전한 IN 문자열 : NbC >= IN 문자열장

설명:

기본 문자열에서 지정된 문자수만 좌측에서 빼낸다.

(* "LEFT" 와 "RIGHT" 블록을 이용한 F B D 프로그램 *)

(* 등가한 S T 언어: *)

complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);

(* complete_string : '56781234'

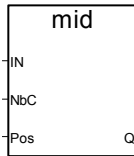
RIGHT 에서 줄출되는 문자열은 '5678'

LEFT 에서 추출되는 문자열은 '1234'
*)

(* 등가한 I L 언어: First done is call to LEFT *)

```
LD      '12345678'
LEFT    4
ST      sub_string (* intermediate result *)
LD      '12345678'
RIGHT   4
ADD     sub_string
ST      complete_string
```

MID (문자열 추출)



인수:

IN	문자열형	공백이 아닌 문자열
NbC	정수형	추출되는 문자수 (<= IN 문자열장)
Pos	정수형	추출되는 문자열의 최초 번호 (최초의 문자 번호는 1)
Q	문자열형	추출된 문자열 (NbC 은 문자열장) =빈 문자열 : 지정된 인수가 부정일 때

설명:

기본 문자열에서 지정된 장소에서 지정된 문자수의 문자열을 추출

(* "MID" 블록을 이용한 F B D 프로그램*)

-

(* 등가한 S T 언어: *)

```
sub_string := MID ('abcdefgh', 2, 4);
```

```
(* sub_string : 'de' *)
```

(* 등가한 I L 언어: *)

```
LD      'abcdefgh'
```

MID 2,4
ST sub_string

MLEN (문자열장)



인수:

IN	문자열형	문자열
NbC	정수형	IN 문자열의 문자열장

설명:

문자열의 문자열 수 (문자열장)

(* "MLEN" 블록을 이용한 F B D 프로그램 *)

```

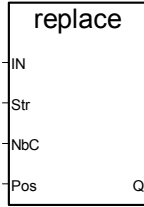
-
(* 등가한 S T 언어: *)
nbchar := MLEN (complete_string);
If (nbchar < 3) Then Return; End_if;
prefix := LEFT (complete_string, 3);
(* 이 프로그램은 문자열에서 좌측 3 문자를 추출해서, prefix 문자열에 넣습니다. 만약,
문자열장이 3 미만일 때는 아무것도 되지 않습니다. *)
  
```

(* 등가한 I L 언어: *)

```

LD      complete_string
MLEN
ST      nbchar
LT      3
RETC
LD      complete_string
LEFT   3
ST      prefix
  
```

REPLACE (문자열 변환)



인수:

IN	문자열형	기본 문자열
Str	문자열형	삽입되는 문자열(NbC 문자를 바꿉니다.)
NbC	정수형	소거되는 문자수
Pos	정수형	삽입되는 최초의 문자 장소의 지정 (최초의 문자는 Pos = 1)
Q	문자열형	지정문자열이 치환된 결과 문자열 - 우선, NbC 문자가 Pos 장소에서 삭제됩니다. - 다음에 삭제되는 문자열 Str 가 이 장소에 삽입됩니다. = 빈 문자열 : Pos <= 0 =두개의 문자열의 결합(IN + Str) : Pos > IN 문자열장 =기본 문자열 : NbC <= 0

설명:

기본 문자열의 일부를 다른 문자열로 치환

(*REPLACE"블록을 이용한 F B D 프로그램*)

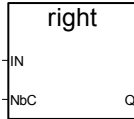
(*등가한 S T 언어:*)

```
MyName := REPLACE ('Mr X JONES, 'Frank', 1, 4);
(* MyName is 'Mr Frank JONES' *)
```

(*등가한 I L 언어: *)

```
LD      'Mr X JONES'
REPLACE 'Frank',1,4
ST      MyName
```

RIGHT (우측 문자열 출력)



인수:

IN	문자열형	공백이 아닌 문자열
NbC	정수형	줄출되는 문자수 ($\leq$ IN 문자열장)
Q	문자열형	IN 문자열의 우측 문자열 (문자열의 길이는 NbC) =빈 문자열 : NbC $\leq$ 0 =완전한 IN 문자열 : NbC $\geq$ IN 문자열장

설명:

기본 문자열에서 지정된 문자수만 우측에서 뽑아낸다.

(* "LEFT" 와 "RIGHT" 블록을 이용한 F B D 프로그램*)

—

(* 등가한 S T 언어: *)

complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);

(* complete_string is '56781234'

RIGHT 에서 줄출되는 문자열 '5678'

LEFT 에서 줄출되는 문자열 '1234'

*)

(* 등가한 I L 언어: First done is call to LEFT *)

LD '12345678'

LEFT 4

ST sub_string (* intermediate result *)

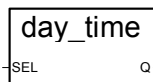
LD '12345678'

RIGHT 4

ADD sub_string

ST complete_string

DAY_TIME



인수:

SEL	정수형	출력 선택 0= 날짜 1= 시각 2= 요일
Q	문자열형	일시 문자열 'YYYY/MM/DD' : SEL = 0 'HH:MM:SS' : SEL = 1 요일 : SEL = 2 (예 : 'Monday')

설명:

문자열로 일시, 요일을 되돌린다.

(* "DAY_TIME" 블록을 이용한 F B D 프로그램 *)

-

(* 등가한 S T 언어: *)

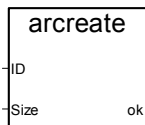
Display := Day_Time (0) + ' ; ' + Day_Time (1);

(* Display 포맷트: 'YYYY/MM/DD ; HH:MM:SS' *)

(* 등가한 I L 언어: First done is call to day_time(1) *)

```
LD      1
DAY_TIME
ST      hour_str    (* intermediate result *)
LD      0
DAY_TIME
ADD     ' ; '
ADD     hour_str
ST      Display
```

ARCREATE (배열 작성)



인수:

ID	정수형	배열 식별자 (0 ~ 15)
Size	정수형	배열 요소 수
ok	정수형	실행 결과 :

- =1 정상으로 배열 작성된때
- =2 부정한 배열 식별자, 또는 이미 배열 작성 종료때
- =3 부정한 배열 요소 수
- =4 메모리 부족

설명:

정수 배열을 작성

주의 : 어플리케이션은 **최대 16** 배열 (a r r a y) 을 정의할 수 있습니다. 배열 요소는 **정수형 값**입니다. 배열 사이즈가 남은 메모리 양에 근접할 경우는 시스템 에러를 일으키는 것도 생각할 수 있습니다.

(* FBD program creating an array of integers *)

-

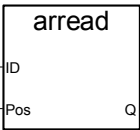
(*등가한 S T 언어:*)

array_error := (ARCREATE (ident, 16) <> 1));

(*등가한 I L 언어:*)

LD ident
ARCREATE 16
NE 1
ST array_error

ARREAD (배열요소 읽어내기)



인수:

ID	정수형	배열 식별자 (0 ~ 15)
Pos	정수형	배열 내의 요소 포지션 (0 ~ size-1)
value	정수형	요소의 값 = 0 (인수가 부정한 때)

설명:

정수치 배열 중에서 지정한 요소를 읽어낸다.

(* 배열을 다루는 F B D 프로그램*)

—

(*등가한 S T 언어:*)

If (array_error) Then Return; End_if;
read_value := ARREAD (ident, index);

(* array_error は ARCREATE コールからかえります *)

(*등가한 I L 언어:*)

LD array_error
RETC
LD ident
ARREAD index
ST read_value

ARWRITE (배열요소 적어넣기)



인수:

ID	정수형	배열 식별자(0 ~ 15)
Pos	정수형	배열 요소 포지션 (0 ~ size-1)
IN	정수형	적어 넣어진 요소값
ok	정수형	실행 결과 =1 적어 넣기가 정상 종료 =2 부정확한 배열 식별자 =3 부정확한 포지션 지정

설명:

정수치 배열요소에 값 적어넣기

(* 배열을 다루는 F B D 프로그램*)

—

(*등가한 S T 언어:*)

```
If (array_error) Then Return; End_if;
write_status := ARWRITE (Ident, Index, value);
(* array_error は ARCREATE 코ールからかえります *)
```

(*등가한 I L 언어:*)

```
LD      array_error
RETC
LD      ident
ARWRITE index,value
ST      write_status
```

F_ROPEN



인수 :

Path 문자열형 파일명

이 파일명에는 퍼스명을 나타내는 기호 (/ \) 을 포함합니다.

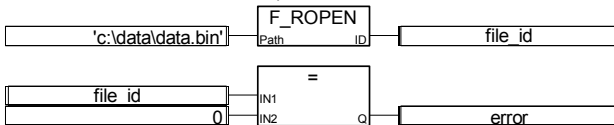
ID 정수형

파일 번호
= 0 에러 발생시 (파일이 발견되지 않았던 것 등)

설명 :

binary 파일을 리드모드로 열립니다. FX_READ, F_CLOSE 가 오픈 후에 사용됩니다. 이 Function 은 ISaGRAF 시뮬레이터 기능에는 포함되어 있지 않습니다.

(* 파일 관리용 F B D 프로그램*)



(* 등가한 S T 언어:*)

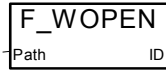
```
file_id := F_ROPEN('c:\data\data.bin');
error := (file_id=0);
```

(* 등가한 I L 언어:*)

```
LD      'c:\data\data.bin'
F_ROPEN
```

ST file_id
EQ 0
ST error

F_WOPEN



인수 :

Path 문자열형 파일명

이 파일명은 퍼스명을 나타내는 기호 (/ \) 를 포함합니다.

ID 정수형

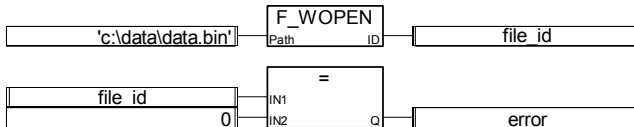
파일 번호

= 0 에러 발생시 (파일이 발견되지 않았던 것 등)

설명 :

binary 파일을 리드모드로 열립니다. FX_READ, F_CLOSE 가 오픈 후에 사용됩니다. 이 Function 은 ISaGRAF 시뮬레이터 기능에는 포함되어 있지 않습니다.

(* 파일 관리용 F B D 프로그램*)



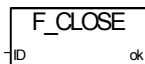
(* 등가한 S T 언어:: *)

```
file_id := F_WOPEN('c:\data \data.bin');
error := (file_id=0);
```

(* 등가한 I L 언어:: *)

```
LD 'c:\data\data.bin'
F_WOPEN
ST file_id
EQ 0
ST error
```

F_CLOSE



인수 :

ID	정수형	파일 번호 (F_OPEN, F_WRITE 되돌리기 값)
ok	boolean 형	되돌리기 값 =TRUE : 정상으로 클로즈 =FALSE : 에러 발생

설명 :

F_WRITE, F_OPEN 에서 열려진 binary 파일을 클로즈합니다. 이 Function 은 ISaGRAF 시뮬레이터 기능에는 포함되어 있지 않습니다.

(* 파일 관리용 F B D 프로그램*)

—

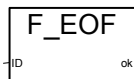
(* 등가한 S T 언어:: *)

```
file_id := F_OPEN('data.bin');
ok := F_CLOSE(file_id);
```

(* 등가한 I L 언어:: *)

```
LD      'data.bin'
F_OPEN
ST      file_id
F_CLOSE      (* file_id is already in the current IL result *)
ST      ok
```

F_EOF



인수 :

ID	정수형	파일 번호 (F_OPEN, F_WRITE 되돌리기 값)
ok	boolean 형	파일의 엔트를 나타냅니다. =TRUE : 파일 읽어내기, 적어 넣기, Function 에서 파일 마지막에 도달한 때 (FM_READ 로 읽어 낸 마지막 내용은 마지막 문자가 EOF 가 아니면, 맞지 않을 가능성이 있습니다.

설명 :

파일 마지막에 도달했는가를 체크합니다. ISaGRAF 시뮬레이터 기능에는 포함되어 있지 않습니다.

(* 파일 관리용 F B D 프로그램*)

==

(* 등가한 S T 언어:: *)

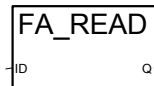
```
file_id := F_ROPEN('data.bin');
WHILE not(F_EOF(file_id))
    VAL := FA_READ(file_id);
END_WHILE;
MESSAGE := 'last val = ' + msg(VAL);
ok := F_CLOSE(file_id);
```

(* 등가한 I L 언어:: *)

```

                LD      'data.bin'
                F_ROPEN
                ST      file_id
                LD      file_id
                F_EOF
                JMPCL   END_O 어드레스 어드레스 F_FILE
NOT_EOF:        LD      file_id
                FA_READ
                ST      VAL
                LD      file_id
                F_EOF
                JMPCL   NOT_EOF (* if not eof, go on reading *)
END_OF_FILE: LD      VAL
                MSG
                ST      val_msg (* conversion of VAL into a message *)
                LD      'last val = '
                ADD     val_msg
                ST      MESSAGE
                LD      file_id
                F_CLOSE
                ST      ok
```

FA_READ



인수 ::

ID	정수형	파일 번호 (F_ROPEN 되돌리기 값) .
Q	정수형	파일에서 읽어 낸 정수치

설명 :

binary 에서 정수치를 읽어냅니다. F_ROPEN, F_CLOSE 와 함께 사용합니다. 앞의 장소에서 sequential 한 파일에 access 합니다. F_ROPEN call 후 최초의 call 에서는 파일의 4 바이트를 읽어냅니다.

매회 call 로 읽어내고 있는 장소를 stack 에 push 합니다.

파일 마지막에 도달한 것을 체크하기 하기 위해 F_EOF 를 사용합니다.

이 Function 은 ISaGRAF simulator 에는 포함되지 않습니다.

(* 파일 관리용 F B D 프로그램*)

■

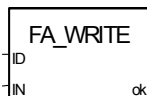
(* 등가한 S T 언어:: *)

```
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);
```

(* 등가한 I L 언어:: *)

```
LD      'voltramp.bin'
F_ROPEN
ST      file_id
FA_READ      (* read vstart *)
ST      vstart
LD      file_id
FA_READ      (* read vend *)
ST      vend
LD      file_id
FA_READ      (* read vinc *)
ST      vinc
LD      file_id
FA_READ      (* read delta_tim *)
TMR          (* conversion into a timer *)
ST      delta_tim
LD      file_id
F_CLOSE
ST      ok
```

FA_WRITE



인수 :

ID	정수형	파일 번호 (F_WOPEN 되돌리기 값)
IN	정수형	파일 적어 넣은 정수치
OK	boolean 형	실행 결과 =TRUE : 정상 종료

설명 :

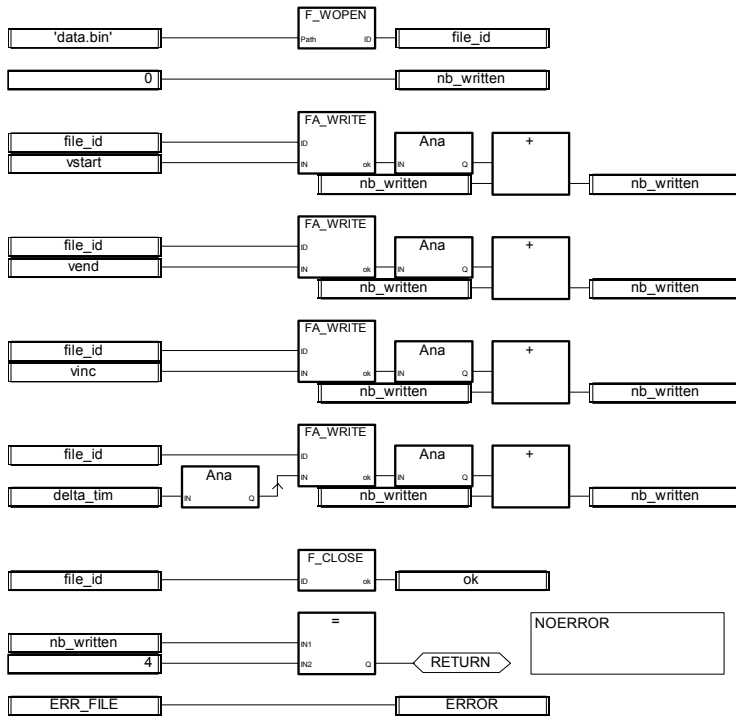
binary 에서 정수치를 읽어냅니다. F_WOPEN, F_CLOSE 와 함께 사용합니다. 앞의 장소에서 sequential 한 파일에 access 합니다. F_WOPEN call 후 최초의 call 에서는 파일의 4 바이트를 읽어냅니다.

매회 call 로 읽어내고 있는 장소를 stack 에 push 합니다.

파일 마지막에 도달한 것을 체크하기 하기 위해 F_EOF 를 사용합니다.

이 Function 은 ISaGRAF simulator 에는 포함되지 않습니다.

(* F B D 프로그램 예 *)



(* 등가한 S T 언어:: *)

```

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;

```

(* 등가한 I L 언어:: *)

```

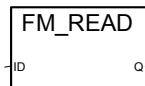
LD      'voltramp.bin'
F_WOPEN
ST      file_id
LD      0
ST      nb_written
LD      file_id      (* write vstart *)

```

```

FA_WRITE    vstart
ANA
ADD         nb_written
ST         nb_written
LD         file_id    (* write vend *)
FA_WRITE    vend
ANA
ADD         nb_written
ST         nb_written
LD         file_id    (* write vinc *)
FA_WRITE    vinc
ANA
ADD         nb_written
LD         (* write delta_tim *)
ANA         (* convert it to an integer *)
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
RETC         (* return if equal 4 *)
LD         ERR_FILE (* else error *)
ST         ERROR
    
```

FM_READ



인수 :

ID	정수형	파일 번호 (F_ROPEN 되돌리기 값) .
Q	문자열형	파일에서 읽혀진 문자열

설명 :

binary 에서 정수치를 읽어냅니다. F_ROPEN, F_CLOSE 와 함께 사용합니다. 앞의 장소에서 sequential 한 파일에 access 합니다. F_ROPEN call 후 최초의 call 에서는 지정장소에서 최초 문자열을 읽어 낼수 있습니다.

매회 call 로 읽어내고 있는 장소를 stack 에 push 합니다.

문자열 마지막에는 NULL(0) EOF ('\n')또는 개행('\r')이 붙어 있습니다

파일 마지막에 도달한 것을 체크하기 하기 위해 F_EOF를 사용합니다.

이 Function 은 ISaGRAF simulator 에는 포함되지 않습니다.

(* 파일 관리용 F B D 프로그램*)

■

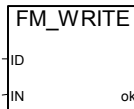
(* 등가한 S T 언어:: *)

```
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);
```

(* 등가한 I L 언어:: *)

```
LD      'voltramp.bin'
F_ROPEN
ST      file_id
FM_READ      (* read status1 *)
ST      status1
LD      file_id
FM_READ      (* read status2 *)
ST      status2
LD      file_id
F_EOF
JMPNC     CLOSE_FILE (* if end of file jump not done *)
D      ERR_FILE
ST      ERROR
LD      file_id
FM_READ      (* read unused_eof_mes *)
ST      unused_eof_mes
CLOSE_FILE: LD      file_id
F_CLOSE
ST      ok
```

FM_WRITE



인수 ::

ID 정수형 파일 번호 (F_WOPEN 되돌리기 값)

IN 문자열형 파일에 적어 넣은 문자열
ok boolean 형 실행 결과
 =TRUE : 정상 종료

설명 : :

binary 에서 정수치를 적어 넣기.

F_CLOSE 와 함께 사용합니다. 앞의 장소에서

문자열 마지막은 NULL(0)로 적어넣습니다.

앞의 장소에서 sequential 한 파일에 access 합니다.

F_WOPEN call 후에 최초의 call 에서는 지정 장소에서 최초의 문자열을 파일에 적어 넣습니다.

매회 call 로 읽어내고 있는 장소를 stack 에 push 합니다.

이 Function 은 ISaGRAF simulator 에는 포함되지 않습니다.

(* 파일 관리용 F B D 프로그램*)

==

(* 등가한 S T 언어:: *)

```
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);
```

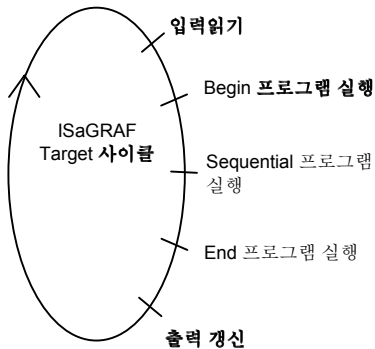
(* 등가한 I L 언어:: *)

```
LD      'trace.txt'
F_WOPEN
ST      file_id
FM_WRITE 'First message' (* write first msg *)
ST      ok
LD      file_id
FM_WRITE 'Last message' (* write second msg *)
ST      ok
LD      file_id
F_CLOSE
ST      ok
```

C. Target 사용자 가이드

C.1 소개

ISaGRAF Target 은 ISaGRAF 어플리케이션이 PC 와 보드상에서 real-time 으로 실행되고 있는 소프트웨어입니다. 아래 그림과 같은 실행 사이클이 됩니다.



Target 사이클은 외부에서 입력 읽기, ISaGRAF 어플리케이션 프로그램의 실행, 외부에서의 출력 갱신으로 구성 됩니다.

제 1부에서는, 정의된 Target 에 대해 해설합니다.

각 Target(DOS, OS-9, VxWorks and NT target) 고유의 설치시에 따른 주의사항, ISaGRAF Target 작동방법, 계속해서 Target 마다의 주의사항, 에러 처리 등에 대한 정보를 정리합니다. 또한, Windows3.1/95/98 과 공존하는 real-time Target : Windows31/95-NT Target 에 관해서는 본 가이드 해설에는 포함되어 있지 않습니다. 별도 매뉴얼을 참조 바랍니다.

제 2부에서는 사용자정의 C언어 Function, Function_Block, 변환 function 의 실행방법에 관해서 해설합니다.

제 3부에서는 workbench 와 Target 간의 통신 protocol 인 MODBUS 에 관해서 Function 코드마다 해설합니다.

제 4부에서는 Target 의 전류이상과 ReStart 를 해설합니다.

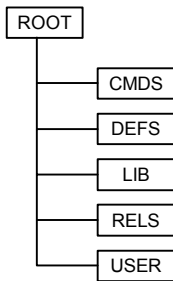
C.2 설치

Target 설치 는 CD_ROM 또는 CD_ROM 에서 작성된 설치용 디스크에서 Target 시스템으로 합니다. 약 1 MB 용량이 Target 시스템에 필요하게 되는 경우도 있습니다. 이 경우는 실행 모듈 이외의 라이브러리도 포함합니다. 또한 각 Target 고유의 주의사항에 관해서는 각 Target 사용법 항을 참조 바랍니다.

또한, 각 Target 처음 디렉토리 readme 파일을 참조 바랍니다.

(예를 들면, 파일명 : target.txt)

Target 시스템 (c:\path) 에는 이하와 같은 디렉토리를 구성할 수 있습니다.



ROOT 디렉토리 : 몇 개의 툴과 readme 파일을 포함합니다.

ARK 디렉토리 : I/O 보드 파일을 포함합니다.

CMDS 디렉토리 : 실행 모듈을 포함합니다.

DEFS 디렉토리 : Header (*.h) 파일을 포함합니다.

LIB 디렉토리 : 라이브러리 파일을 포함합니다.

RELS 디렉토리 : Object 파일을 포함합니다.

사용자 디렉토리 : 사용자정의 C언어 Function, F B 등의 소스 파일과 Header 파일을 포함합니다.

Target 시스템으로 파일 설치가 끝나면, 다음 절의 해설과 같이 각 Target 의 작동이 가능하게 됩니다.

C.3 ISaGRAF DOS target

C.3.1 ISaGRAF 실행 : ISA.EXE

MS-DOS Target 에서는 Single Task 로 ISA.EXE 가 실행됩니다. ISA.EXE 실시때의 옵션은 isa -x 로 볼 수 있습니다. CMDSD 디렉토리에서 실행하십시오.

DOS Target 에서는 workbench Target 간의 통신이 Target 수행에 영향을 주기 때문에 Target 실행 중에 이 통신에 그다지 부하를 걸지 않기를 바랍니다.

Target 은 사용자가 프로그램된 나눠넣기 처리 루틴에는 영향을 주지 않습니다.

☐ 통신 링크 설정: -t 옵션

ISaGRAF Target 은 workbench 디버거와의 통신에 시리얼 통신을 사용합니다. 보드명은 -t 옵션으로 지정합니다. 통신 포트는 COM1, 2, 3, 4 중에서 선택합니다.

기본값없음: 만약, 이 옵션이 지정되지 않은 경우는, workbench 측과의 통신은 불가능합니다. 이때 Target 에는 에러 번호 7 이 표시됩니다.

isanet 통신은 DOS Target 에서는 지원되지 않습니다.

Ethernet 을 이용한 통신은 DOS ISaGRAF target 의 경우엔 지원 되지 않습니다.

ISaGRAF 을 작동하기 전에 **통신 포트파라미터 (parity..)** 설정이 필요하게 됩니다. 이 설정은 workbench 디버거 link 설정 내용과 일치해야 합니다. (상세한 것은 workbench 사용자 가이드 섹션 A 참조)

시리얼통신 포트 설정

MODE COM1:9600,N,8,1

이 설정에 따라 통신 파라미터 는 이하와 같이 설정됩니다.

Baudrate	9600
Parity	없음
Format	8 bits
Flow Control	1 bit

Target 머신 BIOS 설정에 따라 Baudrate 19200 이 지원되는 수가 있기 때문에 주의 바랍니다. 같은 모양의 설정은 ICS Triplex ISaGRAF 사 유틸리티 소프트웨어 (ISAMOD.EXE) 에서도 가능합니다.

예 :

ISAMOD COM1

이 명령어는 MODE COM1:19200,N,8,1 과 같은 형식으로 설정됩니다.

Slave 번호: -s 옵션

이 옵션은 slave 번호를 설정합니다. 1~2 5 5 (단, 1 3 은 제외) 를 사용할 수 있습니다. slave 번호는 통신 link 설정과 함께 사용됩니다. slave 번호는 두개 이상의 Target 이 있을 경우 이들을 구분하기 위해 사용합니다. workbench 디버거를 사용할 때는 link 설정해서 이 slave 번호를 바르게 설정해야 합니다.

기본값: 기본 slave 번호는 1

이것은 workbench 측의 기본값과 같습니다.

DOS-Target 동작 예제:

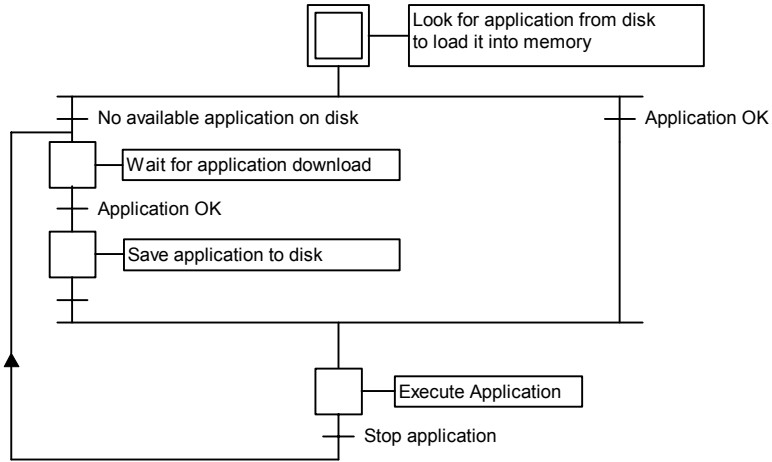
isamod COM1 통신 보트 COM1 을 Baudrate :19200 Parity 없음, 8bit 데이터, stop bit 1 에 설정합니다.

isa -t=COM1 ISaGRAF Target 을 디폴트 slave 번호 1 로 통신 포트 COM1 에 설정합니다.

isa -s=3 -t=COM1 ISaGRAF Target 을 slave 번호 3 으로 통신 포트 COM1 에 설정합니다.

C.3.2 DOS-Target 특징**ISaGRAF 시작**

Target 이 작동될 때 아래의 알고리즘 으로 실행됩니다.



• 정의

어플리케이션 코드 (중간 코드) 는 vinally 데이터에서 workbench 에 따라 생성되고, 다운로드된 것입니다. 이것이 ISaGRAF Target 에 따라 실행됩니다. 심벌 테이블이 동시에 사용되는 경우도 있습니다.

어플리케이션 심벌 테이블은 텍스트 파일로 workbench 에 따라 생성되고, 다운로드된 것입니다. 이 파일에 따라 심벌 (변수명) 과 Target 내부 데이터와 link 부처기를 합니다. 사용자가 심벌을 사용하지 않는 경우 이 파일은 다운로드될 필요는 없습니다. 심벌 테이블 포맷은 섹션 A device 트 프로그램에 해설되고 있습니다.

• 응용프로그램 백업

새로운 어플리케이션이 workbench 디버거에서 다운로드 되면, 어플리케이션 코드는 Target 의 Current 디렉토리에 이하의 파일명으로 보존됩니다.

ISAx1 ISaGRAF 어플리케이션 코드 백업 파일
(x 는 slave 번호)

또한, 어플리케이션 심벌 테이블이 먼저 다운로드 되어 있는 경우 이 심벌 테이블은 같은 Current 디렉토리에 이하의 파일명으로 보존됩니다.

ISAx6 ISaGRAF 어플리케이션 심벌 테이블의 백업 파일
(x 는 slave 번호)

ISaGRAF Target 이 작동되면, 어플리케이션 심벌 코드와 심벌 파일이 동일 디렉토리에서 검색됩니다. 그 후 이것은 메모리 상에 load 됩니다. 심벌 파일이 없는 경우 Target 은 어플리케이션 코드만으로 실행됩니다. 어플리케이션 코드가 없는 경우는 이것이 다운로드 되는 것을 기다리고 있습니다.

Target 전류 ON 일때 디버거에서 다운로드 없이 특정 어플리케이션 Start 시키려면 이 파일들을 Target Current 디렉토리에 copy 해 둘 필요가 있습니다. 파일 시스템이 없는 경우는 가상 디스크에 copy 합니다.

만약, ISaGRAF workbench 가 표준 디렉토리 (\isawin) 에 install 되어 있으면 프로젝트 (MYPROJ) 의 Application Code 는 (즉, 중간 코드) 이하와 같이 됩니다.

\\SAWIN\\APL\\MYPROJ\\appli.x8m

같은 식으로 어플리케이션 심벌 파일은 이하와 같습니다.

\\SAWIN\\APL\\MYPROJ\\appli.tst

예 : 어플리케이션 코드의 copy

isa.exe 가 install 되어 있는 디렉토리 (c:\target\dos\cmds) 에 MYPROJ 프로젝트 어플리케이션 코드를 copy 합니다. 여기서 isa.exe 가 실행되면 MYPROJ 프로젝트를 작동됩니다.

copy \\SAWIN\\APL\\MYPROJ\\appli.x8m

isa11

이러한 Batch command 는 workbench 툴 메뉴에 사용자가 추가하는 것이 가능합니다. (상세한 것은 섹션 A 을 참조합니다.)

에러 처리와 출력 메시지

ISaGRAF Target 소프트웨어에는 에러 검출 처리가 포함되어 있습니다. 에러 메시지와 그 설명은 본 매뉴얼 뒷장 에서 해설합니다.

에러 검출의 흐름은 이하와 같습니다.

- 에러와 에러번호는 ISaGRAF 에러 루틴으로 보냅니다.
- 만약, workbench 측에서 코드 생성 메뉴로 에러 처리 Flag 가 설정되어 있으면, 에러 처리가 되지만, 그렇지 않는 경우는 에러 정보는 무시됩니다.

에러가 처리되는 경우 :

- 에러 번호 (10 진수) 와 인수 (16 진수) 디폴트 출력 장치 (stdout) 에 표시됩니다.
- 이 에러번호와 인수는 ring F I F O BUFFER 에 등록됩니다. BUFFER 사이즈 workbench 코드 생성 메뉴에서 설정 가능합니다. 만약, F I F O 가 가득 차면 새로운 에러가 발생하면, 가장 오래된 에러는 없어집니다.

- 에러정보는 workbench 디버거창 또는 어플리케이션 중의 SYSTEM call_Function 에서 집어낼 수 있습니다.

workbench 디버거가 에러를 검출한 경우는 에러 메시지 창에 에러 메시지가 표시됩니다. 여기서 어플리케이션 실행상태 (실행중, 정지중) 표시에 덧붙혀 에러가 발생한 프로그램명과 변수명, 에러번호, 인수를[]내에 표시합니다.

또한, Target 측에서는 디폴트 출력 장치 stdout 에 에러 메시지가 표시되지만, 이것을 비표시하는 경우는 이하와 같이 ISaGRAF 을 작동하게 됩니다.

isa -t=COM1 -s=1 >NUL



시스템 clock

ISaGRAF Target 은 사이클 타임 관리와 타임변수 관리를 위해 DOS-Target 에서는 표준 타임 체크가 사용되고 있습니다. 이 정밀도는 55ms 가 됩니다.

따라서, 55ms 보다 정밀도가 높은 타임 변수를 가질 수 없습니다. 같은 이유로 55ms 보다 짧은 사이클 타임을 설정할 수 없습니다. 만약, 설정하면 사이클 타임 over follow 에러가 발생합니다. (에러 번호 6 2) 이때는 사이클 타임의 트리거가 무시되고 연속적으로 ISaGRAF Target 이 실행됩니다.

표준 시스템 체크를 변경하고 싶은 메리트 로서는, 상주형 ISaGRAF 이외의 프로그램과 C언어 Function 이 ISaGRAF 실행에 따라 영향을 받지 않는다는 것입니다.



종료 키

DeskTop PC 상에서 ISaGRAF 를 실행하는 경우는 이하의 키 편성으로 ISaGRAF 에서 종료 시킬 수 있습니다.

shift + ctrl + alt

만약, 위의 키를 사용할 수 없는 경우는 P C 를 reset 할 필요가 있습니다.

ctrl + alt + del

물론, 갑작스런 키 입력으로 인한 ISaGRAF 종료를 피하기 위해서 이들 키를 무시하는 ISaGRAF Target 도 있습니다.

그러므로, 이러한 종료에서는 I/O 보드 interface 가 반드시 출력을 OFF 해서 종료하는 것을 보증할 수 없기 때문입니다.

I/O 출력을 OFF 해서 ISaGRAF 을 정상 종료하기 위해서는 디버거창에서 어플리케이션을 정지합니다.

shift + ctrl + alt



응용프로그램 코드 사이즈의 제한

MS-DOS Target 에션 Real Mode 로 실행 중인 어플리케이션 코드 사이즈의 최대값은 64K B 입니다. 만약, 이 이상 크기의 프로그램이 작성되어, 다운로드 되는 경우는 다운로드 후에 ISaGRAF 가 이 어플리케이션을 실행하려고 시스템을 crash 할 가능성이 있습니다.

또한 취급하는 메모리도 640KB convention 메모리를 넘을 수 없습니다.

C.4 ISaGRAF OS9 Target

우선, OS-9 Target 에 필요한 실행 모듈 (CMDS 디렉토리에서) 을 파일 전송 툴로 전송합니다.

그 다음 전송된 ISaGRAF 실행은 명령어 라인에서의 도움말을 참조합니다.

```
isa -?
isaker -?
isatst -?
Isanet -?
```

C.4.1 ISaGRAF Single Task 실행: isa 옵션

ISaGRAF 는 Single Task 로서 실행 가능합니다. 그러나, 이 경우는 통신 OverLoad 가 직접 ISaGRAF 처리 Performance 에 영향을 줍니다. MultiTask 의 경우는 동일 CPU 에서 복수 ISaGRAF Target 이 slave 번호와 통신 포트설정을 변경해서 실행됩니다.

통상, Single Task 의 실행은 Single Task 만 할 수 없는 OS 예를 들면,MS-DOS 상과 ISaGRAF Target 이식 작업할때 proto 타입으로 할 경우가 많습니다.

때문에, ISaGRAF 실행은 MultiTask Mode 에서 하는 것이 바람직하다고 할 수 있습니다.

또한, ISaGRAF Single Task Mode 에서의 실행으로 백그라운드에서 실행중인 Task 와 나눠놓기 처리 등이 방해 받는 일은 없습니다.

☞ 통신 link 설정: -t 옵션

SingleTask 모드로의 실행에서는 디버거와의 통신에 시리얼 통신을 사용합니다. 이 통신 포트를 -t 옵션으로 설정합니다.

기본값없음 : 만약, 이 옵션이 생략되면, 디버거와의 통신은 불가능합니다. 이때 에러 번호 7 이 콘솔에 표시됩니다.

Isanet link 에 따라 통신은 SingleTask 모드에서는 사용할 수 없습니다.

시리얼 linkdevice 는 vinally 데이터 전송 모드로 오픈합니다. (콘트롤 캐릭터 없고, XON/XOFF 없음) 다른 **통신 파라미터**는 ISaGRAF 가 작동되기 전에 설정해 둘

필요가 있습니다. 이 설정 내용은, workbench 측에서의 통신 link 설정과 일치시켜줄 필요가 있습니다.

예 :

xmode /t0 baud=19200

/t0device 통신 baudrate 를 19200 로 설정한다.

Slave number: -s 옵션

이 옵션은 slave 번호를 설정합니다. 1~2 5 5 (단, 1 3 은 제외) 를 사용할 수 있습니다. slave 번호는 통신 link protocol 부에서 사용됩니다. 두개 이상의 Target 이 실행되어 있는 경우에 이들을 식별하기 위해 사용됩니다. workbench 측에서 link 의 설정으로 디버거하는 Target 과 slave 번호를 일치 시킬 필요가 있습니다.

기본값 : slave 번호의 기본값은 1 입니다. (이것은 workbench 측의 디폴트 설정과 같습니다.)

예:

isa -t=/t0 ISaGRAF 는 SingleTask 로 실행. slave 번호 1, 통신 포트 /t0 으로 설정

isa -s=3 -t=/t1 ISaGRAF 를 SingleTask 로 실행. slave 번호 3 통신 포트 /t1 로 설정

isa -t=/t0 &

isa -s=3 -t=/t1 두개의 ISaGRAF 를 SingleTask 로 실행. 하나는 slave 번호 1, 통신 포트 /t0 으로 설정, 또 하나는

C.4.2 ISaGRAF multitasks: isaker, isatst, isanet

isaker, isatst, isanet slave 번호 3, 통신 포트 /t1 로 설정

ISaGRAF Target Kernel reference 를 개정하기 위해, Target 통신 부분을 구분해서 두개의 Task 로 분리합니다. (어플리케이션 실행 Task 는 isaker : Kernel Task, 통신 Task 는 isatst 또는 isanet 이라 합니다.)

이 구성은 유연성이 있고, 예를 들면, **하나의 Kernel Task 에 두개 이상의 통신 Task 를 link 하기도 하고, 하나의 통신 Task 에 최대 네개의 Kernel Task 를**

link 시킬 수 있습니다. 예를 들면, 다른 프로세스 모니터 link 등의 테스트인 복수 Task 와 디버거가 하나의 어플리케이션 Target 과 같은 통신 포트를 공유하고 link 를 뺄 수 있습니다.

Kernel Task 와 통신 Task 는 독립되어 있지만, 작동 조건으로서 Kernel Task 가 최초로 실행된 후에 통신 Task 가 작동되어야 합니다.

멀티 Task 모드에서의 ISaGRAF 가 백그라운드 프로세스와 나눠놓기 처리를 방해하는 일은 없습니다.

C.4.2.1 kernel task 실행: isaker

⇒ **Slave number: -s 옵션**

이 옵션은 slave 번호를 설정합니다. 1 ~ 2 5 5 (단, 1 3 은 제외) 를 사용할 수 있습니다. slave 번호는 통신 link protocol 내부에서 사용됩니다. 두개 이상의 Target 이 실행되고 있는 경우에 이것을 식별하기 위해 사용됩니다. workbench 측에서 link 의 설정으로 디버거하는 Target 과 slave 번호를 일치 시킬 필요가 있습니다.

기본값 : slave 번호의 기본값은 1 입니다. (이것은 workbench 측의 기본 설정과 같습니다.)

C.4.2.2 serial communication task 실행: isatst

⇒ **통신 link 설정: -t 옵션**

싱글 Task 모드로의 실행에서는 디버거와의 통신에 시리얼 통신을 사용합니다. 이 통신 포트를 -t 옵션으로 설정합니다.

기본값 : 없음

만약, 이 옵션이 생략되면, 디버거와의 통신은 불가능합니다. LEO 에러 번호가 7 이 콘솔에 표시됩니다.

Isanet link 에 따라 통신은 isatst 통신 Task 에서는 사용할 수 없습니다.

시리얼 linkdevice 는 vinally 데이터 전송 모드로 오픈 합니다. (Control 캐릭터 없고, XON/XOFF 없음) 다른 **통신 파라미터** 는 ISaGRAF 가 작동되기 전에 설정해 둘 필요가 있습니다. 이 설정의 내용은 workbench 측에서 통신 link 의 설정과 일치 시켜 둘 필요가 있습니다.

예 :

xmode /t0 baud=19200

/t0device 통신 포트를 19200 으로 설정한다.

⇒ **Slave number: -s 옵션**

이 옵션은 salve 번호를 설정합니다. 1 ~ 2 5 5 (단, 1 3 은 제외) 를 사용할 수 있습니다. 이 옵션은 연속으로 최대 네개의 다른 Kernel salve 에 대해 설정 가능합니다. salve 번호는 통신 link protocol 내부에서 사용됩니다. 두개 이상의 Target 을 실행하고 있을 경우에 이들을 식별하기 위해 사용됩니다. workbench 측에서 link 의 설정으로 디버거하는 Target 과 salve 번호를 일치시킬 필요가 있습니다.

기본값 : salve 번호의 기본값은 1 입니다. (이것은 workbench 측의 디폴트 설정과 같습니다.)

⇒ **통신 task 논리 번호: -c 옵션**

이 옵션은 통신 Task 의 논리번호 설정합니다. 동시에 복수 통신 Task 를 관리하는 경우에 사용합니다. 1 ~ 2 5 5 의 번호로 각각의 통신 Task 번호는 다를 필요가 있습니다.

기본값 : salve 번호 (-s 옵션으로 지정) . 이것은 3.0x 버전과 호환성을 가지고 있습니다.

C.4.2.3 Ethernet 통신 task 실행: isanet

⇒ **통신 link 설정: -t 옵션**

Target 통신 Task isanet 은 표준 isanet 통신을 디버거통신으로 사용합니다. 보드번호는 -t 옵션으로 설정합니다.

기본값 : 없음

만약, 이 옵션이 생략되면, 디버거와의 통신은 불가능합니다. 이때 에러 번호는 7 이 콘솔에 표시됩니다.

workbench 측의 link 설정을 Target 측의 통신 Task 와 일치시켜 둘 필요가 있습니다.

ISaGRAF 에 있어 OS-9 Target 측은 서버에 해당하는 디버거측은 지정된 포트 번호에 접속되는 Client 에 해당합니다.

Isanet 통신에 따라 디버거하기 전에 OS-9 의 isanet device 가 바르게 설정, 실행되어 있을 필요가 있습니다. 예를 들면, isanet 통신의 확인은 ping 명령어로 가능합니다.

≡ **Slave number: -s 옵션**

이 옵션은 slave 번호를 설정합니다. 1~2 5 5 (단, 1 3 은 제외) 를 사용할 수 있습니다. 이 옵션은 연속으로 최대 네개의 다른 Kernelslave 에 대해 설정 가능합니다. slave 번호는 통신 link protocol 내부에서 사용됩니다. 두개 이상의 Target 을 실행하고 있는 경우에 이들을 식별하기 위해 사용됩니다. workbench 측에서 link 설정으로 debug 하는 Target 과 slave 번호를 일치시킬 필요가 있습니다.

기본값 : slave 번호의 기본값은 1 입니다. (이것은 workbench 측의 기본설정과 같습니다.)

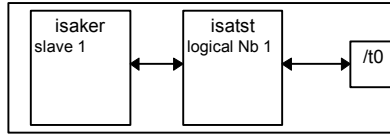
≡ **통신 task 논리 번호: -c 옵션**

이 옵션은 통신 Task 논리번호를 설정합니다. 동시에 복수 통신 Task 를 관리하는 경우에 사용합니다. 1~2 5 5 의 번호에서 각각의 통신 Task 번호는 다를 필요가 있습니다.

기본값 : slave 번호 (-s 옵션으로 지정). 이것은 3.0x 버전과 호환성을 가지고 있습니다.

C.4.2.4 예:

```
isaker &
isatst -t=/t0
```

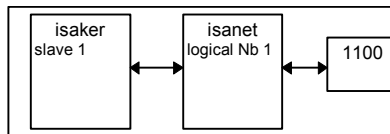


시작:

ISaGRAFKernel 디폴트의 slave 번호 1 로서 실행.

ISaGRAF 시리얼 통신 Task 를 실행. link 설정 내용은 /t0 을 통신포트, 기본 slave 번호 1 에 link, 기본논리번호 1 (= 최종 slave 번호는 디폴트 1)

**isaker &
isanet -t=1100**

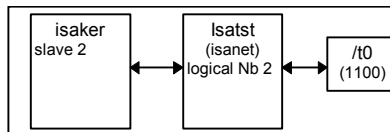


시작:

ISaGRAFKernel Task 를 기본 slave 번호 1 에서 실행.

ISaGRAF isanet 통신 Task 를 실행. link 설정 내용은 포트번호 1100, 기본 slave 번호 1 로 link, 기본논리번호 1 (= 최종 slave 번호는 디폴트 1)

**isaker -s=2 &
isatst -t=/t0 -s=2** (respectively isanet -t=1100 -s=2)

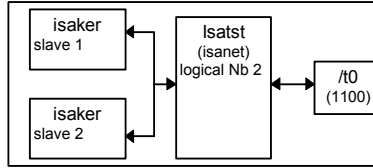


시작:

ISaGRAF KernelTask 를 slave 번호 2 로 해서 실행.

ISaGRAF 시리얼 (isanet) 통신으로 실행. Link 설정 내용은 /t0 통신포트 (포트 번호 1100) , slave 번호 2 로 link, 논리번호는 2 (= 최종 slave 번호가 2)

**Isaker -s=1 &
isaker -s=2 &
isatst -t=/t0 -s=1 -s=2** (respectively isanet -t=1100 -s=1 -s=2)



Starts:

ISaGRAF Kernel Task 를 slave 번호 1 로 실행.

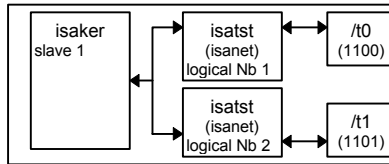
ISaGRAF Kernel Task 를 slave 번호 2 로 실행.

ISaGRAF 시리얼 (isanet) 통신 Task 를 실행. link 설정내용은 /t0 통신포트 (포트번호 1100) , slave 번호 1,2 양쪽에 link 논리번호는 디폴트 2 .

Isaker -s=1 &

isatst -t=/t0 -s=1 -c=1 & (respectively isanet -t=1100 -s=1 -c=1 &)

isatst -t=/t1 -s=1 -c=2 (respectively isanet -t=1101 -s=1 -c=2)



시작:

ISaGRAF KernelTask 를 slave 번호에서 실행.

ISaGRAF 시리얼 (isanet) 통신 Task 를 실행. link 설정 내용은 /t0 통신포트 (포트번호 1100) , slave 번호 1 에서 link, 논리번호 1 .

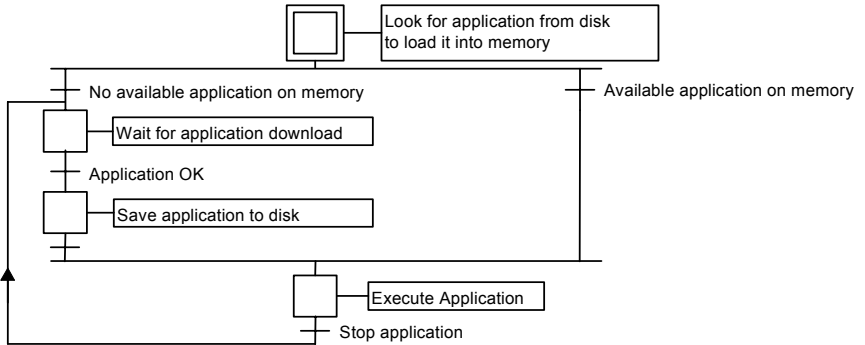
ISaGRAF 시리얼 (isanet) 통신 Task 를 실행. link 설정 내용은 /t1 통신포트 (포트번호 1101) , slave 번호 1 에서 link, 논리번호 2 .

주의 : 시리얼, isanet 통신 Task 는 공존할 수 없습니다.

C.4.3 OS-9 Target 특징

ISaGRAF 시작

ISaGRAF Target 은 이하 알고리즘에 따라 실행됩니다.



• 정의

어플리케이션 코드 (중간 코드) 는 workbench 에서 생성되는 vinally 어플리케이션 데이터 베이스입니다. 이 코드는 Target 에 따라 해석, 실행됩니다. 또한, 어플리케이션 심벌 테이블을 편성해서 사용되는 일도 있습니다. 이 어플리케이션 심벌 테이블은 ASCII 파일에서 Target 내부의 오브젝트와 심벌 (변수) 과의 연속 부치기를 합니다. 이 심벌 테이블은 Target 이 실행될 때 필수적인 것은 아닙니다. 심벌 테이블에 대한 상세한 advanced 프로그램항목을 참조 바랍니다.

• ISaGRAF OS-9 objects / Multi-application

모든 ISaGRAF Target 의 어플리케이션 코드는 ISAxn 파일명이 됩니다. 여기서 x는 Kernel slave 번호, n는 스페이스 번호라는 특별한 의미를 갖고 있습니다. 단, ISAy3 만은 예외입니다. 여기서 y는 통신 Task 논리번호가 됩니다.

동일 C P U 상에서 복수 ISaGRAF Target (Kernel Task 또는 통신 Task) 은 slave 번호와 통신 Task 논리번호가 다르면 실행됩니다. 다른 어플리케이션이라도 동일 I/O 보드의 access 가 가능하기 때문에 어플리케이션측에서 주의해야 합니다. 예를 들면, I/O 드라이브에 Semaphore 관리를 넣는 것도 하나의 방법입니다.

OS-9 Target 오브젝트명 :

파일명 :

ISAx1 ISaGRAF 어플리케이션 코드의 백업파일

ISAx6 ISaGRAF 어플리케이션 심벌의 백업파일

메모리 모듈명 (스페이스 공간) :

ISAx0 ISaGRAF Kernel 시스템 데이터

- ISAx1** ISaGRAF 어플리케이션 코드
- ISAx2** ISaGRAF Kernel realtime 데이터 베이스
- ISAy3** ISaGRAF 통신용 데이터 buffer (메일박스)
- ISAx4** ISaGRAF 온라인 수정용 어플리케이션코드 1
- ISAx5** ISaGRAF 온라인 수정용 어플리케이션코드 2
- ISAx6** ISaGRAF 어플리케이션 심벌

여기서 x 는 slave 번호

사용자는 이들과 같은 이름을 사용하지 않는 것을 필요로 합니다.

• Application 백업

새로운 어플리케이션이 workbench 측에서 Target 으로 다운로드 되면, 어플리케이션 코드가 이하 이름의 파일로서 current 디렉토리에 보존됩니다.

ISAx1 ISaGRAF 어플리케이션 코드의 백업파일 (x 는 slave 번호)

만약 이 코드전에 어플리케이션 심벌 테이블이 다운로드 되어 있으면, 이 심벌 테이블은 이하의 이름 파일로서 current 디렉토리에 보존됩니다.

ISAx6 ISaGRAF application 심벌의 백업파일 (x 는 slave 번호)

ISaGRAF Target 이 스타트할 때, 이들 어플리케이션 코드와 심벌 테이블파일이 current 디렉토리에서 찾아, 데이터 모듈로서 동일 명으로 메모리에 로드 됩니다.

만약, 심벌 테이블이 없는 경우는 심벌이 로드 되지 않는 상태로 Target 이 stop 합니다.

만약, 어플리케이션 코드가 없는 경우 Target 은 어플리케이션 코드가 다운로드를 유지합니다.

전원 ON 일때 자동적으로 Target 이 특정 어플리케이션에서 작동하지 않는 경우는 이하와 같습니다.

workbench 가 설치되어 있는 P C에서 어플리케이션 코드 파일 (ISAx1) 과 심벌 테이블 파일 (ISAx6) 을 Target 을 실행하는 디렉토리에 copy 해 둡니다. 파일 전송 톨과 workbench 디버거 메뉴를 사용합니다.

주의 : ISaGRAF 의 디버거에서 브레이크 포인트의 설정 등은, 만약 어플리케이션 코드가 적어 넣기가 허용되지 않는 경우는 (예를 들면, R OM화 되어 있음) 불가능 합니다. workbench 가 설치 (\ISAWIN 디렉토리) 되어 있는 P C에서 프로젝트명 MYPROJ 의 어플리케이션 코드 파일은 이하와 같습니다.

\ISAWIN\APL\MYPROJ\appli.x6m (Target 상에서의 ISAx1 에 해당).

같은 어플리케이션 심벌 테이블 파일은 이하와 같습니다.

\\SAWIN\\APL\\MYPROJ\\appli.tst (Target 상에서의 ISAx6 에 해당).

⇐ **에러처리와 출력 메시지**

ISaGRAF Target 소프트웨어는 에러검출 처리가 포함되어 있습니다. 에러 메시지와 그 설명은 본 매뉴얼 뒷장 에서 해설합니다.

에러 검출의 흐름은 이하와 같이 됩니다.

- 에러와 에러번호는 ISaGRAF 에러 루틴에 전송됩니다.
- 만약, workbench 측에서 코드 생성 메뉴로 에러 처리 플러그가 set 되어 있으면, 에러 처리하지만, 그렇지 않는 경우 에러정보는 무시됩니다.

에러가 처리되는 경우 :

- 에러번호 (10 진수) 와 인수 (16 진수) 기본출력장치 (stdout) 에 표시됩니다.
- 이 에러번호와 인수는 ring FIFO BUFFER 에 등록됩니다. BUFFER 사이즈는 workbench 코드 생성 메뉴에서 설정 가능합니다. 만약, FIFO 가 가득 차 새로운 에러가 발생하면 가장 오래된 에러는 없어집니다.
- 에러 정보는 workbench 디버거창 또는 어플리케이션 가운데 STEM call 어플리케이션에서 집어낼 수 있습니다.

workbench 디버거가 에러를 검출한 경우는 에러 메시지 창에 에러 메시지가 표시됩니다. 여기서 어플리케이션 실행상태 (실행중, 정지중) 표시에 덧붙여 에러가 발생한 프로그램명과 변수명, 에러번호, 인수들 [] 안에 표시합니다.

또한, Target 측에서는 기본출력장치 stdout 에 에러 메시지가 표시되지만, 이것을 비표시할 경우는 이하와 같이 ISaGRAF 를 작동하게 됩니다.

```
prog_name [options] >>>/nil
```

⇐ **사이클 주기, task behaviors, task priorities**

ISaGRAF Target 사이클 마지막에 (다음 사이클이 start 하기 직전), 이하 알고리즘에서 처리됩니다.

어플리케이션에 사이클 타임이 지정되어 있었던 경우 CPU 는 남은 시간 (지정 사이클 타임 현재 사용한 시간) 을 다른 처리로 돌립니다. 만약, 이 시간이 모자랄 때는 사이클 over flow 가 되고, 다음 scheduler 1 체크분 기다려서, 다음 사이클이 start 합니다.

만약, 사이클 타임이 지정되어 있지 않은 경우 또는 남은 시간이 1tic 이하인 경우는 다음 1tic 후에 다음의 사이클이 start 합니다.

Target 사이클 타임의 정밀도는 OS-9 시스템의 시스템 체크에 해당합니다.

통상, 사이클 타임을 지정해서, C P U 의 공백 시간을 다른 Task 에 돌립니다.

통신 Task 는 통신 데이터가 통신 link 에서 오지 않을 때는 슬리프모드가 되어 있습니다. Kernel Task 에서의 송신 / 수신 파킷에 따라 작동됩니다. 통신 Task 는 Kernel 에 Question 를 던져 Kernel 사이클 마지막에 (동기형(Sync 통신의 경우) Kernel 은 통신 Task 에 대해서 Answer 를 되돌립니다.

ISaGRAF Target Task 의 priority 는 Task 스스로 변경할 수 없습니다. 사용자가 전체 구성에서 판단해서 주어집니다. 예를 들면, 우선도가 낮은 Task 에서 priority 되지 않기 위해서는 OS-9Task 관리 파라미터 (**MIN_AGE** 와 **MAX_AGE**) 를 수정합니다.

⇒ **터미널 모드**

Target 시리얼 통신 protocol 은 캐리지 리턴 (\$0D) 을 3 회 받으면, OS-9 의 S h e l l Task 가 real linkdevice 로 배정되어 있으면, 작동합니다.

따라서, OS-9 의 Shell Prompt 가 어떠한 터미널에도 표시됩니다. 이것을 사용해서 OS-9 의 Shell 에 들어갑니다.

예 :

호스트 P C 측,

- ISaGRAF degger 를 닫습니다.
- W i n d o w s 터미널 섹션을 정확한 통신 파라미터 에서 start 합니다.
- 리턴 키를 3 회 입력합니다.

그러므로 OS-9 의 Shell 에 로그인 가능하게 됩니다.

- **logout 입력**에 따라 터미널 모드를 종료할 수 있습니다.

주의 : 다음번 workbench 디버거에서의 통신을 위해서도 반드시 터미널 모드는 logout 로 종료해 둘 필요가 있습니다.

C.5 ISaGRAF VxWorks target

ISaGRAF Target 을 실행하기 전에 환경 설정을 위해 몇 개의 명령어를 실행해야 합니다. 이들 명령어는 Script 파일에서 실행 가능합니다. 이하에 그 설정에 대해 기술합니다.

C.5.1 시스템 리소스 관리: isassr.o

이 모듈은 어떠한 ISaGRAF Target 의 경우에도 필요합니다. 반드시 최초에 로드 되어야만 합니다. 이 모듈에 따라 복수 Target 실행이 가능합니다.

C.5.2 모듈 : isa.o, isakerse.o, isakeret.o

이하의 모듈이 **isassr.o** 이 로드된 후에 로드됩니다.

isa.o :	ISaGRAF 를 SingleTask 모드 (시리얼 통신 부착) 에서 작동
isakerse.o :	ISaGRAF 를 멀티 Task 모드 (시리얼 통신 부착) 에서 작동
isakeret.o :	ISaGRAF 를 멀티 Task 모드 (시리얼 통신 또는 isanet 통신 부착) 에서 작동

이들 모듈 해설은 나중에 기술합니다.

Serial 통신 link 구성

ISaGRAF Target 은 기본적으로 시리얼통신을 디버거와의 사이에서 합니다. ISaGRAF Target 에서는 이 시리얼 link 의 설정을 하지 않기 때문에 사용자는 다른 방법으로 설정해야 합니다. 단, vinally 데이터 전송모드 (RAW 모드) 이어야 합니다. 이하 ISAMOD () 서버루틴이 제공되기 때문에 사용할 수 있습니다.

```
uchar ISAMOD
(
    char *desc,          /* Serial device name */
    uint32 baudrate      /* Baud rate */
)
```

해설 :

지정된 real link device 를 지정된 baudrate 에서 vinally 데이터 전송 모드로 설정합니다.

되돌리기 값 :

0 (정상시) , BAD_RET (에러시)

workbench 측이 link 설정은 Target 측과 일치시켜 둘 필요가 있습니다.

☐ **시스템 clock rate**

글로벌 변수 CLKRATE (uint32)는 VxWorks 시스템 CLKE 에 초기화되어야만 합니다.

예를 들면,

```
CLKRATE = sysClkRateGet ()
```

CLKRATE 기본값은 60Hz 입니다.

C.5.3 ISaGRAF single task 실행 : isa.o

ISaGRAF 는 SingleTask 로서 실행 가능합니다. 그러나 이 경우는 통신 overload 가 직접 ISaGRAF 처리 performance 에 영향을 줍니다. MultiTask Mode 의 경우는, 동일 CPU 상에서 복수 ISaGRAF Target 이 slave 번호와 통신 포트 설정을 바꿔서 실행됩니다.

통상 SingleTask 로서의 실행은 SingleTask 밖에 할 수 없는 OS, 예를 들면 MS-DOS 와 ISaGRAF Target 이식작업을 할 때 proto 타입으로 실행하는 경우가 많습니다.

때문에 ISaGRAF 실행은 MultiTask Mode 로 하는 것이 바람직하다고 할 수 있습니다.

또한, ISaGRAF SingleTask 에서의 실행으로, 백그라운드에서 실행중인 Task 와 나눠줄기 등을 방해하는 일은 없습니다.

☐ **Slave(s) 등록**

ISaGRAF Target 에 slave 번호를 설정합니다. 1~255 (단, 1 3 은 제외) 까지 사용 가능합니다.

slave 번호는 통신 link protocol 내부에서 사용됩니다. 두개 이상의 Target 을 실행하고 있는 경우에 이들을 식별하기 위해 사용됩니다. workbench 측에서의 link 설정으로 디버거하는 Target 과 slave 번호를 일치 시킬 필요가 있습니다.

slave 번호 등록을 위해 이하의 *isa_register_slave()* 루틴이 제공되고 있습니다.

```
uchar isa_register_slave
(
    uchar slave      /* slave number */
)
```

해설 :

새로운 slave 번호를 멀티 Target 관리 시스템에 등록합니다.

되돌리기 값 :

0 (정상시) , BAD_RET (에러시)

☐ **Application 백업 파일 저장 단위**

글로벌 변수 TSK_FUNIT (char *) 가 어플리케이션 백업파일명으로 사용됩니다. ISaGRAFTarget 은 표준 파일 관리 루틴 (fopen, fread, fwrite, fclose) 을 파일백업에 사용할 수 있습니다.

기본값은 ("")입니다.

예 :

```
TSK_FUNIT = "host name :/C :/ISaGRAF/target/apl/"
```

이 예에서는 *host_name* PC 상의 C드라이브인 ISaGRAF\target\apl\ 디렉토리를 지정하고 있습니다. 마지막 "/"을 잊어 버리면, ISaGRAF\target\ 디렉토리 apl 파일명이 지정되기 때문에 주의해 주십시오.

상세한 것은 나중에 어플리케이션 섹션을 참조 바랍니다.

== **마지막 사이클 컨트롤**

TSK_NBTCKSCHED (uint 32) 변수가 다음 사이클까지 기다리는 시간을 표시합니다. 기본값은 0 (즉, 동일 우선 Task 스케줄 ring) 입니다.

이 값은 Target 작동 전에 set 됩니다. 상세한 것은 사이클 타임, Task priority 섹션을 참조 바랍니다.

== **ISaGRAF target 동작**

환경 설정이 끝나면, isa_main 루틴으로 ISaGRAF Target 을 Spawn 합니다.

```
uchar isa_main
(
uchar slave,      /* Slave number*/
char *com         /* Serial device name */
)
```

해설 :

ISaGRAF Task 작동

되돌리기 값 :

0 (정상시), 기타 (에러시)

slave 번호는 전에 기술한 slave 번호를 등록해서 실행한 번호입니다.slave 번호와 통신 포트가 다르면, 복수 Target 을 작동할 수 없습니다. Target 시 workbench 측에 link 설정과 Target 측의 link 설정이 일치해야만 합니다.

예제

이하에 ISaGRAF 를 SingleTask 모드 (slave 번호 1, 시리얼 link 설정 /tyCo/1) 에서 실행하는 스텝을 나타냅니다.

current 디렉토리는 Target 이 설치되어 있는 디렉토리과 같게 합니다.

isassr.o 모듈 로드

ld < RELS/isassr.o

isa.o 모듈 로드

ld < CMDS/isa.o

시리얼 통신 link 설정

ISAMOD ("/tyCo/1", 19200)

시스템 CLKRATE

CLKRATE = sysClkRateGet ()

slave 번호 등록

isa_register_slave (1)

파일의 스테레주니트 (기본값이라면 생략 가능)

TSK_FUNIT = ""

사이클 마지막의 처리 (기본값이라면 생략 가능)

TSK_NBTCKSCHED = 0

ISaGRAF Target 작동

sp (isa_main, 1, "/tyCo/1")

C.5.4 ISaGRAF multitasks 실행: isakerse.o and isakeret.o

ISaGRAF Target Kernel reference 를 개선하기 위해 Target 통신부분을 구분해 두개의 Task 로 분리합니다. (어플리케이션 실행 Task 는 isaker : KernelTask, 통신 Task 는 isatst 또는 isanet 라고 합니다.)

이 구성은 유연성이 있고, 예를 들면, 하나의 Kernel Task 에 두개 이상의 통신 Task 를 link 하고, 하나의 통신 Task 에 최대 4 개의 Kernel Task 를 link 시킬 수 있습니다. 예를 들면, 다른 프로세스 모니터 ring 등의 Task 인 복수 Task 와 디버거 link 가 하나의 어플리케이션 Target 과 통신 포트를 공유해서 link 를 뺄 수 있습니다.

KernelTask 와 통신 Task 는 독립해 있지만, 작동 조건으로서, KernelTask 가 최초로 실행된 후에 통신 Task 가 작동되어야 합니다.

멀티 Task 모드에서의 ISaGRAF 가 백그라운드 프로세스와 나뉘내기 처리를 방해하는 일은 없습니다.

두개의 모듈이 통신과 하드 웨어 기능에 따라 선택 가능합니다.

- Kernel 과 시리얼 통신 link : isakerse.o

이 모듈로 복수 KernelTask 과 복수 시리얼 통신 Task 를 작동할 수 있습니다.

- Kernel 시리얼 또는 isanet 통신 link : isakeret.o

이 모듈로 복수 KernelTask 와 복수 시리얼 또는 isanet 통신 Task 를 작동할 수 있습니다.

ISaGRAF 의 작동에 관해서는 isakerse.o 와 isakeret.o 모듈 중에서 같은 형식으로 가능합니다. 단, isakeret.o 에 관해서는 통신 Task (tst_main_ex) 작동에 관한 시리얼 link device 명 또는 isanet link 용 포트번호의 어디서라도 설정 가능합니다.

VxWorksTarget 은 서버되어 디버거는 지정된 포트 경유로 접속되는 client 로서 동작합니다.

Kernel(s) 등록

ISaGRAF Target 에 slave 번호를 설정합니다. 1~255 (단, 13은 제외) 까지 사용 가능합니다. slave 번호는 통신 link protocol 내부에서 사용됩니다. 두개 이상의 Target 을 실행하고 있는 경우에 이들을 식별하기 위해 사용됩니다. workbench 측에서 link 설정으로 debug 하는 Target 과 slave 번호를 일치시킬 필요가 있습니다.

slave 번호 등록을 위해 이하의 *isa_register_slave()* 루틴이 제공되고 있습니다.

```
uchar isa_register_slave
(
    uchar slave      /* slave number */
)
```

해설 :

새로운 slave 번호를 멀티 Target 관리 시스템에 등록합니다.

되돌리기 값 :

0 (정상시) , BAD_RET (에러시)

통신 task(s) 등록

ISaGRAF 통신 Task 에 논리번호를 설정합니다. 두개 이상의 통신 Task 를 동시에 사용할 경우에 사용합니다. 1 ~ 2 5 5 의 번호를 사용할 수 있습니다. ISaGRAF 통신 Task 를 작동하기 전에 이 논리번호를 사전에 등록해 두어야 합니다.

통신 Task 논리번호 등록을 위해 이하의 *isa_register_com()* 루틴이 제공되고 있습니다.

```
uchar isa_register_com
(
    uchar com_id      /* com. task identifier */
)
```

해설 :

새로운 통신 Task 논리번호를 멀티 Target 관리 시스템에 등록합니다.

되돌리기 값 :

0 (정상시) , BAD_RET (에러시)

⇒ Application 백업 파일 저장 단위

글로벌 변수 TSK_FUNIT (char *)가 어플리케이션 백업 파일명으로 사용 됩니다. ISaGRAF Target 은 표준 파일관리 루틴 (fopen, fread, fwrite, fclose) 을 파일 백업 사용 가능합니다. 기본값은 ("") 입니다.

예 :

```
TSK_FUNIT = "host name :/C :/ISaGRAF/target/apl/"
```

이 예에서 *host_name* P C 상의 C 드라이브인 ISaGRAF\target\apl\ 디렉토리를 지정하고 있습니다. 마지막의 \을 잊어버리면,ISaGRAF\target\ 디렉토리 apl 파일명이 지정되기 때문에 주의해 주십시오.

상세한 것은 나중에 어플리케이션 백업 파일 섹션을 참조 바랍니다.

⇒ 마지막 사이클 컨트롤

TSK_NBTKSCHED (uint 32) 변수가 다음 사이클까지 기다리는 시간을 나타냅니다. 기본값은 0 (즉, 동일 우선의 Task 스케줄 ring) 입니다.

이 값은 Target 의 작동 전에 set 됩니다. 상세한 것은 사이클 타임, Task priority 섹션을 참조 바랍니다.

⇒ ISaGRAF kernel 동작

환경 설정이 끝나면 isa_main 루틴에서 ISaGRAF Kernel 을 기동 합니다.

```
uchar isa_main
```

```
(
uchar slave,      /* Slave number      */
char *com         /* NOT USED Empty string is OK */
)

```

해설 :

ISaGRAF KernelTask 의 작동

되돌리기 값 :

0 (정상시) 기타 (에러시)

slave 번호는 이전 Kernel slave 번호의 등록 항으로 해설되고 있지만, 두개 이상의 Kernel 이 slave 번호가 다르면 작동 가능합니다.

ISaGRAF 통신 task 동작

환경 설정이 끝나면 ISaGRAF 통신 Task (tst_main_ex) Spawn (起動) 합니다.

```
uchar tst_main_ex
(
char *com,        /* 통신 device 명 */
uchar *slave,     /* Location of a 4 Bytes field specifying kernel(s) slave
                  to link to */
uchar com_id      /* communication task identifier */
)

```

해설 :

SaGRAF 통신 Task 작동

되돌리기 값 :

0 (정상시) 기타 (에러시)

통신 link 로 접속되는 Kernel Task 의 slave 번호로 4 바이트 필드가 확보되어 있습니다. 만약, Kernel 수가 네개 이내 일 때 이 필드는 0 입니다. Task 가 작동된 후에는 이 필드는 사용되지 않습니다.

통신 device 명은 통신 link 에 설정되어 있는 device 명에 해당합니다.

두개 이상의 통신 Task 의 논리번호가 다르면, 작동 불가능합니다. 디버거와의 통신에 workbench 측에서의 link 설정은 Target 측과 일치해야 합니다.

예제:

이하에 예를 나타냅니다.

우선 ISaGRAF Kernel Task 를 slave 번호 1 로 작동합니다.

ISaGRAF 신호 기동 (Kernel slave 번호 1, 통신 Task 논리번호 1, 시리얼 link 통신 device 명 /tyCo/1 로 설정)

또한 ISaGRAF 통신 Task 를 작동합니다. (Kernel slave 번호 1, 통신 Task 논리번호 2, 포트번호 1 1 0 0 isanet 통신으로 설정)

현재 디렉토리는 Target 이 설치되어 있는 곳으로 합니다.

isassr.o 모듈 로드

ld < RELS/isassr.o

isakeret.o 모듈 로드 (만약, isanet 통신이 불필요한 경우는 isakerse.o 모듈을 로드)

ld < CMDS/isakeret.o

시리얼 통신의 설정

ISAMOD ("/tyCo/1", 19200)

시스템 CLKRATE 설정

CLKRATE = sysClkRateGet ()

Kernel slave 번호 등록

isa_register_slave (1)

통신 논리 번호 등록

isa_register_com (1)

isa_register_com (2)

파일 보존 unit 설정 (기본값에서는 생략 가능)

TSK_FUNIT = ""

사이클 엔드 컨트롤 (기본값에서는 생략 가능)

TSK_NBTCKSCHED = 0

ISaGRAF Kernel 작동

sp (isa_main, 1, "")

통신 Task slave link 설정

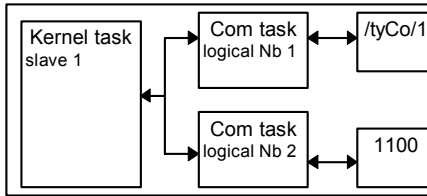
SlavesLink = 0x01000000

ISaGRAF 통신 Task 작동

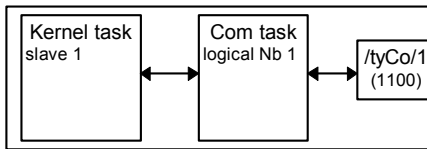
sp (tst_main_ex, "/tyCo/1", &SlavesLink, 1)

sp (tst_main_ex, "1100", &SlavesLink, 2)

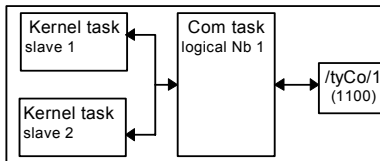
이하의 작동 개요는 아래 그림과 같습니다.



또한, 이하와 같은 구성도 가능합니다.

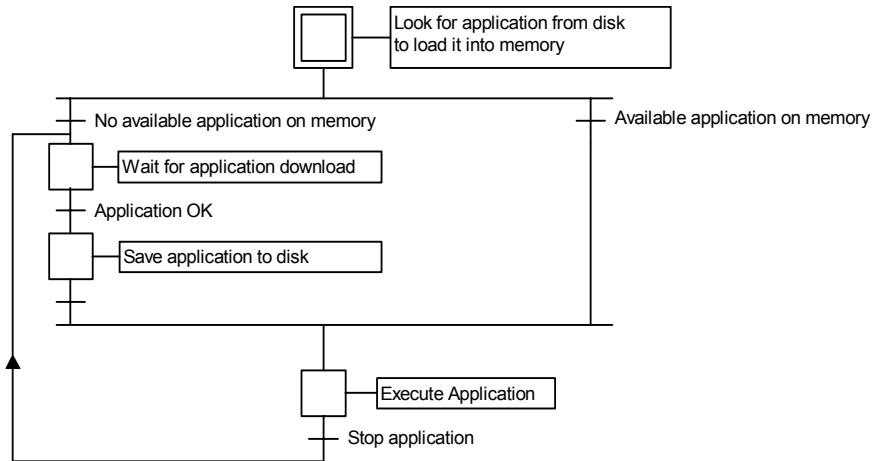


또한, 두개의 Kernel Task 에 대해서 하나의 통신 Task 를 작동한 경우의 예를 아래 그림에서 나타냅니다. 이 경우 slave link 설정은 SlavesLink = 0x01020000 이 됩니다.



C.5.5 VxWorks Target 의 특징

ISaGRAF 시작



정의

어플리케이션 코드 (중간 코드) 는 workbench 에서 생성되는 vinally 어플리케이션 데이터 베이스입니다. 이 코드는 Target 에 따라 해석, 실행됩니다. 또한, 어플리케이션 심벌 테이블과 편성되어 사용되는 수도 있습니다. 이 어플리케이션 심벌 테이블은 ASCII 파일에서 Target 내부의 오브젝트와 심벌 (변수) 과 연속 부치기를 합니다. 이 심벌 테이블은 Target 이 실행될 때 반드시 필요한 것은 아닙니다. 심벌 테이블에 관한 상세한 프로그램 항목을 참조 바랍니다.

ISaGRAF 복수-Target

동일 C P U 상에서 복수 ISaGRAF Target (Kernel Task 또는 통신 Task) 은 slave 번호와 통신 Task 논리 번호가 다르면 실행됩니다. 사용자 자신이 공유 자원을 관리해야 합니다. 예를 들면, I/O 보드등이 여기에 해당합니다. 다른 어플리케이션이어도 동일 I/O 보드 access 가 가능하기 때문에, 어플리케이션측에서 주의 해야 할 필요가 있습니다. 예를 들면, I/O 드라이브에 Semaphore 관리를 넣는 것도 하나의 방법입니다.

Application 백업

새로운 어플리케이션이 workbench 측에서 Target 에 다운로드 되면 어플리케이션 코드는 이하의 파일명으로 보존됩니다. (Target 에서는 표준적인 fopen 을 사용하고 있습니다.)

pathISaX1 ISaGRAF 어플리케이션 코드 (T I C 코드) 백업 (여기서 x 는 slave 번호)

또한, 어플리케이션 심벌 테이블 앞에 다운로드 되어 있던 경우는 이하의 파일명으로 보존되고 있습니다.

pathISaX6 ISaGRAF 어플리케이션 심벌 테이블의 백업 (여기서 x 는 slave 번호)

path 는 ISaGRAF 작동 시에 글로벌 변수 TSK_FUNIT 로 지정되어 있습니다. 디폴트 ("") 는 디스크 파일 unit 가 존재하지 않는 것을 의미합니다.

ISaGRAF Target 의 작동 시에는 current 디렉토리로 이 파일들을 찾아. 발견되면, 메모리상에 로드합니다.

만약 심벌 테이블이 메모리 상에 없으면, 심벌 테이블 없이 어플리케이션은 작동됩니다. 또한, 어플리케이션 코드 (TIC 코드) 가 메모리 상에 없으면 Target 은 어플리케이션 다운로드를 기다립니다.

전원 ON 시에 자동적으로 workbench 측의 디버거와 접속 없이 Target 을 특정 어플리케이션으로 작동하고 싶은 경우는 이하와 같이 합니다.

workbench 가 설치되어 있는 P C에서 어플리케이션 코드 파일과 심벌 테이블 파일을 Target 이 실행하는 디렉토리에 copy 해 둡니다. 파일 전송 톨과 workbench 톨 메뉴 Customized 에서 이 파일을 전송합니다.

또는 어플리케이션 코드 (필요에 대응하는 어플리케이션 심벌 테이블파일) 를 (PROM, EPROM) 메모리 상에 별도 톨을 사용해서 준비해 둡니다. 전원 ON 시에 별도 톨을 사용해서 이들 내용을 R A M 에 로드해서 사용하는 것도 가능합니다. (access 고속화와 BreakPoint 설정을 위해)

ISaGRAF Task 작동 (spawn) 전에 어플리케이션 코드와 어플리케이션 심벌 테이블 메모리 상에서의 어드레스를 지정해 둘 필요가 있습니다. 이하와 같이 해서 S S R 글로벌 변수를 초기화합니다.

SSR[x][1].space = 어플리케이션 코드 어드레스

필요에 따라서

SSR[x][6].space = 어플리케이션 심벌 어드레스

따라서 tasy0ssr.h 파일에 str_ssr 구조체로서 SSR 글로벌 변수를 선언할 수 있습니다.

주의 : ISaGRAF 디버거에서의 브레이크 포인트 설정 등은 만약, 어플리케이션 코드에 적어 넣기가 허용되지 않는 경우는 (예를 들면, ROM화 되어 있음) 불가능 합니다.

workbench 가 설치 (\ISAWIN 디렉토리) 되어 있는 PC에서 프로젝트명 MYPROJ 어플리케이션 코드 파일은 이하와 같습니다.

\ISAWIN\APL\MYPROJ\appli.x6m (Target 상에서의 ISAx1 에 해당).

같은 형식으로 어플리케이션 심벌 테이블 파일은 이하와 같습니다.

\ISAWIN\APL\MYPROJ\appli.tst (Target 상에서의 ISAx6 에 해당).

⇨ **에러처리와 출력 메시지**

ISaGRAF Target 소프트웨어에는 에러 검출 처리가 포함되어 있습니다. 에러 메시지와 그 설명은 본 매뉴얼 뒷장 에서 해설합니다.

에러 검출 흐름은 이하와 같습니다.

에러와 에러번호는 ISaGRAF 에러 루틴에 보내집니다.

- 만약, workbench 측에서 코드 생성 메뉴로 에러 처리 플러그가 set 되어 있으면, 에러 처리되지만, 그렇지 않는 경우는 에러 정보는 무시됩니다.

에러가 처리되는 경우 :

- 에러 번호 (10 진수) 와 인수 (16 진수) 기본출력장치 (stdout) 에 표시됩니다.
- 이 에러 번호와 인수 ringFIFO BUFFER 에 등록됩니다. BUFFER 사이즈는 workbench 코드 생성 메뉴에서 설정 가능합니다. 만약, FIFO BUFFER 가 가득 차서 새로운 에러가 발생하면, 가장 오래된 에러는 없어집니다.
- 에러 정보는 workbench 디버거창 또는 어플리케이션 내의 SYSTEM call 어플리케이션에서 꺼낼 수 있습니다.

workbench 디버거가 에러를 검출한 경우는 에러 메시지 창에 에러 메시지가 표시됩니다. 여기서는 어플리케이션 실행상태 (실행중, 정지중) 표시에 덧붙여 에러가 발생한 프로그램명과 변수명, 에러번호, 인수를 []내에 표시합니다.

Target 에서 에러가 검출한 경우, 에러 번호는 기본 stdout 출력으로 표시됩니다. 이 표시 리다이렉트는 VxWorks 에서 이하 루틴을 사용 합니다.

ioGlobalStdSet()

또는 **ioTaskStdSet()**

나중의 루틴에서는 Kernel 과 통신 Task 는 에러 표시하지 않습니다.

⇒ 사이클 타임, Task, Task priority

- ISaGRAF Target 사이클 마지막에 (다음 사이클이 start 하기 직전), 이하의 algorithm 으로 처리됩니다.

어플리케이션에 사이클 타임이 지정되어 있었던 경우는 CPU는 남았던 시간 (지정 사이클 타임 - 현재 사용한 시간) 이외의 처리로 회전합니다. 만약, 이 시간이 부족할때 사이클 over follow 가 되고, ISaGRAF 작동시에 set 되어 있는 TSK_NBTCKSCHED tic 분만 기다리게 합니다.

만약, 사이클 타임이 지정되어 있지 않는 경우, 또는 남은 시간이 1tic 이하인 경우는 ISaGRAF 작동시에 set 되어 있는 TSK_NBTCKSCHED 체크분만 기다리게 합니다.

Target 사이클 타임의 정밀도는 V x W o r k s 시스템 tic 정밀도에 해당합니다.

통상 사이클 타임을 지정해서 CPU 공백 시간을 V x W o r k s 상의 다른 Task 에 회전시킵니다.

- 통신 Task 는 통신 데이터가 통신 link 에서 오지 않을 때는 sleep 모드가 되어 있습니다. Kernel Task 에서의 송신 / 수신 파킷에 따라 작동됩니다. 통신 Task 는 Kernel 에 Question 을 던져, Kernel 사이클 마지막에 (동기형(Syn 통신의 경우) Kernel 은 통신 Task 대해서 Answer 로 돌립니다.

ISaGRAFTarget Task priority 는 Task 자체 변경이 불가능합니다. 사용자가 전체 구성에서 판단해 주게 됩니다.

C.6 ISaGRAF NT target

C.6.1 ISaGRAF 실행

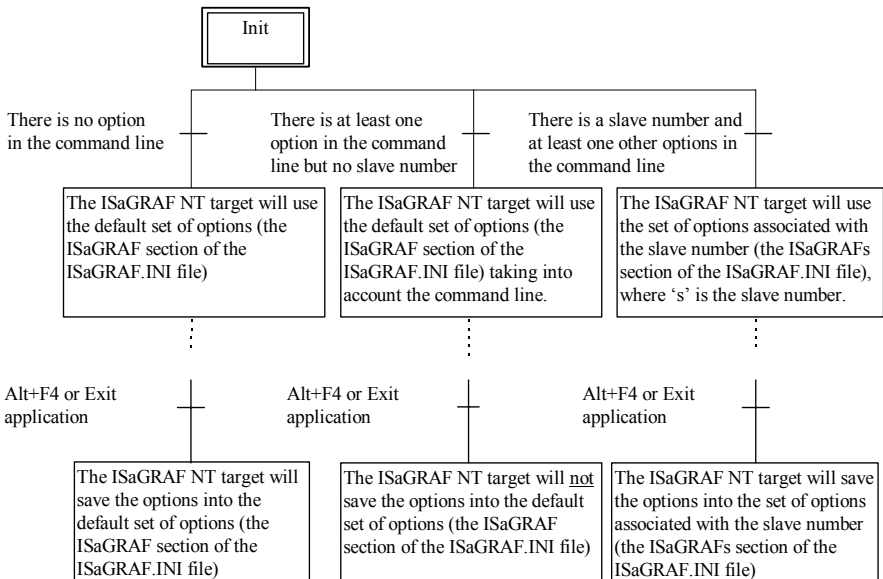
N T 환경에서 ISaGRAF Target (WISAKER.EXE) 은 Kernel thread 와 통신 thread 를 가진 하나의 Task 가 됩니다. 이것을 복수로 동시에 실행할 수 있습니다. 단, WISAKER.EXE 를 작동할 때는 slave 번호와 중복되지 않도록 해야 합니다.

본 Target 의 실행에 따라 다른 나눠내기 처리 프로그램을 방해하는 일은 없습니다.

대상으로 하고 있는 N T 로서 WindowsNT 3.51 이후 (NT4.0 이 바람직하다.) 가 됩니다.

C.6.2 일반 정보 옵션

Target 실행시 옵션 설정은 이하 다이어그램에 보존되어, 읽혀집니다.



또한, ISaGRAF.ini 파일은 WISAKER.EXE 가 존재하고 있는 current 디렉토리에 보존됩니다.

Slave number: -s 옵션

이 옵션은 slave 번호를 설정합니다. 1~255 (단, 1 3 은 제외) 까지 사용 가능합니다. 이 옵션은 연속으로 최대 4 개의 다른 Kernel slave 대해서 설정 가능합니다. slave 번호는 통신 link protocol 내부에서 사용됩니다. 두개 이상의 Target 을 실행하고 있는 경우에 이들을 식별하기 위해 사용됩니다. workbench 측에서의 link 설정으로 debug 하는 Target 과 slave 번호를 일치시킬 필요가 있습니다.

기본값 : slave 번호의 기본값은 1 또는 ISaGRAF.INI 내에서의 설정치

예 :

N T Target 을 slave 번호 2 에서 작동합니다.

WISAKER.EXE -s=2

사용자 I / F : N T Target 의 메인 창에서 「Options」 - 「Slave」 명령어로 slave 번호를 선택해서 start 해 고칠 수 있습니다.



마우스에서 상하 버튼을 선택해서 slave 번호를 선택할 수 있습니다. 단, 선택된 slave 번호를 유효하게 하기 위해서는 NT-Target 에서 한번 더 start 가 필요하게 됩니다.

통신 link 설정: -t 옵션

ISaGRAF Target 은 시리얼 link 통신과 isanet 통신을 디버거통신에 사용합니다. 포트 지정에 -t 옵션을 사용합니다. 시리얼 통신에는 COM1, COM2, COM3, COM4 중에 선택 가능하고, isanet 통신엔, 포트번호 1100 이상의 번호를 사용할 수 있습니다.

기본값 : isanet 통신에서 포트번호는 1100 입니다. 또 시리얼 통신에서는 COM1 이 됩니다. 단, ISaGRAF.ini 파일에 설정치가 있으면 그것을 우선으로 합니다.

주의 : 디폴트의 통신수단은 isanet 통신으로 되어 있습니다.

예 :

시리얼 통신 COM2 을 사용해서 Target 을 작동

WISAKER -t=COM2

isanet 통신 포트번호 1101 을 사용해서 Target 을 작동

WISAKER -t=1101

시리얼 통신 설정 :

-t = COMx 옵션을 첨가하고 있는 경우에 한해 이하의 옵션이 또한
선택 가능합니다.

시리얼 통신으로의 옵션 설정 항목

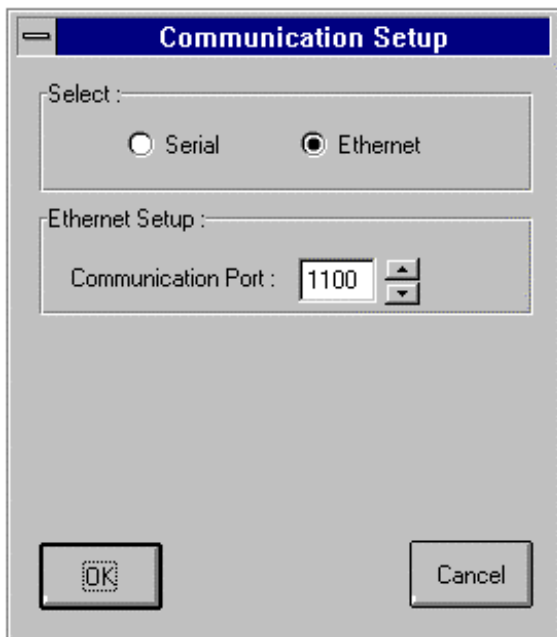
옵션	값	의미
baud	600 1200 2400 4800 9600 19200	Baud
vinally	N e o	Vinally 없음 짝수 홀수
data	7 or 8	데이터길이
stop	1 or 2	Stop bit 길이
Flow	H n	하드웨어 제어 없음

위 테이블에서의 기본값은 baud 19200, parity 없고, 8 데이터 장, 1 Stop bit , follow 제어 없음

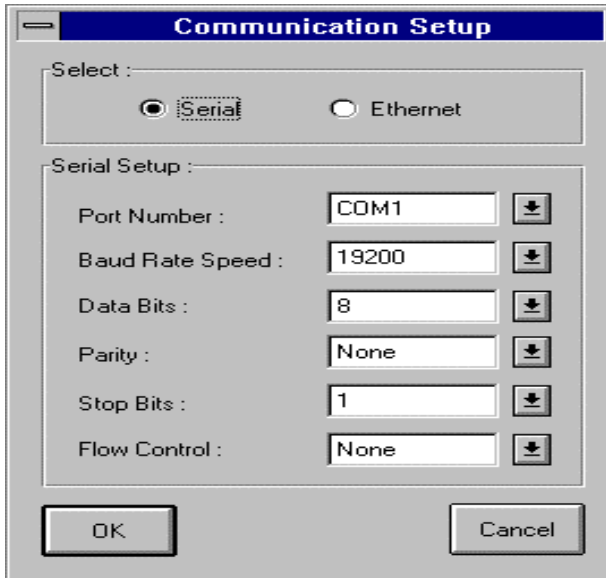
예 :

WISAKER -t=COM1 baud=1200 data=8 vinally=n stop=2

사용자 I / F : NT-Target 의 Main window 「Options」 - 「Communication」
command 에서 선택



시리얼 통신 또는 isanet 통신의 선택이 가능합니다. isanet 통신을 선택한 경우는 포트번호도 선택 가능 합니다. 이들 번호는 workbench 측의 link 설정내용과 일치해야 할 필요가 있습니다.



시리얼 통신을 선택하면 설정화면이 표시됩니다. 이 설정은 workbench 측의 link 설정내용과 일치해야 할 필요가 있습니다.

virtual 보드의 그래픽 시뮬레이션 : -x 옵션

만약, I/O 접속 에디터 (섹션A 참조) 에서 I/O 보드가 virtual 로 선택되어 있었던 때에 이 옵션이 지정되면, 그 I/O 보드만이 그래픽 시뮬레이션 됩니다. 선택 가능한 값은 0 또는 1 이 됩니다. 1 이 지정되면 그래픽 시뮬레이션 됩니다.

기본값 : 기본값은 0 입니다. 또는 ISaGRAF.ini 파일 내용이 기본값이 됩니다.

예 :

I/O 보드 시뮬레이션 첨가로 Target 작동

WISAKER -x=1

사용자 interface : 메뉴 체크박스에서 선택/비선택을 택할 수 있습니다.

주의 : 그래픽 시뮬레이션을 행할 때는 realtime Target 의 realtime 은 보증할 수 없기 때문에 주의 바랍니다. 사이클 타임 over follow 에러 빈도가 높게 됩니다. 시뮬레이션을 비선택하면 정상 상태로 돌아옵니다.

ISaGRAF NT-Target priority 설정 : -p 옵션

Target 은 N T 환경에서 실행하기 때문에 thread 우선도를 지정하는 것은 의미있는 일입니다. 예를 들면, 어떤 특성의 비판적인 ISaGRAF 어플리케이션 Target 을 높은 priority 에 설정해서, 기타 Target 을 보다 낮은 priority 로 설정해서 백그라운드에서 실행시킬 수 있습니다.

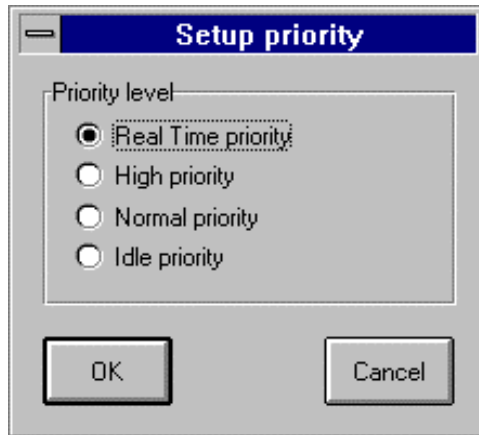
설정 가능한 값은 0, 1, 2, 3 에서 0 이 최고 높고, 3 이 가장 낮은 priority 가 됩니다.

예 :

WISAKER -p=0

WISAKER -p=1

사용자 I / F : N T Target 메인 창 「Options」 - 「Priority」 명령어로 지정합니다.



가장 높은 priority 는 real time 이고, 가장 낮은 priority 는 Idle 입니다.

0: Real time priority

1: High priority

2: Normal priority

3: Idle priority

주의 : Real time priority 는 administrator 권한이 없으면 설정할 수 없습니다.

예 :

wisaker -t=COM1

slave 번호를 디폴트 1, 통신 포트를
COM1 에서 Target 을 작동

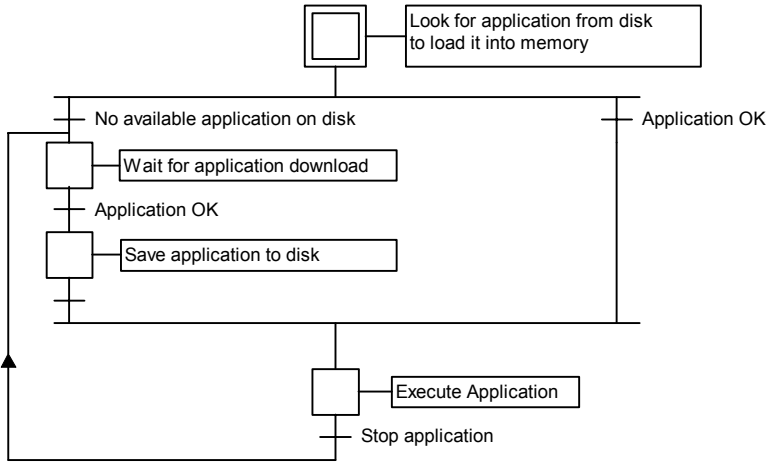
wisaker -s=3 -t=COM1

slave 번호를 3, 통신포트를 COM1 에서
Target 작동

C.1.3 NT-Target 의 특징

ISaGRAF Target 의 작동

ISaGRAF Target 은 이하의 algorithm 따라 실행됩니다.



정의

어플리케이션 코드 (중간 코드) 는 workbench 에서 생성되는 vinally 어플리케이션 데이터 베이스입니다. 이 코드는 Target 에 따라 해석, 실행됩니다. 또 어플리케이션 심벌 테이블과 편성되어 사용되는 일도 있습니다. 이 어플리케이션 심벌 테이블은 ASCII 파일에서 Target 내부의 오브젝트와 심벌 (변수) 과의 연속 부치기를 합니다. 예를 들면, D D E interface 와 변수명에 의한 I/O 보드 시뮬레이션에는 이 심벌 테이블이 사용됩니다.이 심벌 테이블은 Target 이 실행될 때 반드시 필요한 것은 아닙니다. 심벌 테이블에 대한 상세한 것은 advised 프로그램 항을 참조 바랍니다.

ISaGRAF 복수-타겟

동일 C P U 상에서 복수 ISaGRAF Target 은 slave 번호와 통신 Task 논리번호가 다르면, 실행됩니다. 사용자 스스로 공유자원을 관리할 필요가 있습니다. 예를 들면, I/O 보드 등이 거기에 해당합니다. 다른 어플리케이션이 있어도, 동일 I/O 보드의 액세스가 가능하기 때문에

어플리케이션측에서 주의 해야 할 필요가 있습니다. 예를 들면, I/O 드라이브에 SEMAPHORE 관리를 넣는 것도 하나의 방법입니다.

• Application 백업

새로운 어플리케이션이 workbench 측에서 Target 에 다운로드 되면, 어플리케이션 코드는 이하의 파일명으로 Target Current 디렉토리에 보존됩니다.

ISAx1 ISaGRAF 어플리케이션 코드 (T I C 코드) 백업 (여기서 x 는 slave 번호)

또한 어플리케이션 심벌 테이블이 먼저 다운로드 되어 있었던 경우는 이하의 파일명으로 보존되고 있습니다.

ISAx6 ISaGRAF 어플리케이션 심벌 테이블 백업 (여기서 x 는 slave 번호)

ISaGRAF Target 작동 시 에는 current 디렉토리에서 이들 파일을 찾아서 발견되면, 메모리상에 로드합니다.

만약 심벌 테이블이 메모리상에 없으면 심벌 테이블이 없이도 어플리케이션은 작동됩니다. 또한 어플리케이션 코드 (TIC 코드) 가 메모리상에 없으면 Target 은 어플리케이션 다운로드를 유지합니다.

전원 ON 시에 자동적으로 workbench 측의 디버거와의 접속 없이 Target 을 특정 어플리케이션으로 작동하고 싶은 경우는 workbench 가 설치되어 있는 PC 디렉토리에서 어플리케이션 코드의 파일과 심벌 테이블 파일을 Target 이 실행하는 디렉토리에 copy 해 둡니다.

workbench 가 install (\ISAWIN 디렉토리) 되어 있는 PC 에서 프로젝트명 MYPROJ 어플리케이션 코드의 코드 파일은 이하와 같습니다.

\ISAWIN\APL\MYPROJ\appli.x8m (Target 상에서의 ISAx1 에 해당).

같은 형식으로 Application Symbol Table 파일은 이하와 같습니다.

\ISAWIN\APL\MYPROJ\appli.tst (Target 상의 ISAx6 에 해당).

예 :

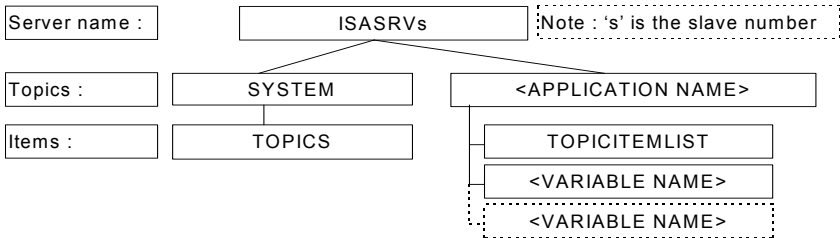
WISAKER.EXE 가 Install 되고 있는 디렉토리에서 이하의 명령어가 실행되면 WISAKER.EXE 는 `myproj` 어플리케이션을 찾아내 실행합니다.

copy \ISAWIN\APL\MYPROJ\appli.x8m isa11

이러한 명령어는 batch 파일로 등록해 두면, workbench 툴 메뉴 customized 에 따라 직접 메뉴에서 실행 가능합니다. (상세한 것은 프로그램 관리 섹션 참조)

DDE

NT-Target 은 DDE (Dynamic Data Exchange) 서버 에도 있습니다. 각종 소프트 웨어가 DDE client 로서 NT-Target 과 접속되어, 변수 교환이 가능합니다. 예를 들면, MS-Excell 에서는 DDE 통신을 개입시켜 NT-Target 정수형 변수 그래픽 애니메이션이 가능합니다. 이 DDE 를 유효하게 하기 위해서는 어플리케이션 심벌 테이블 파일이 필요하게 됩니다. DDE 에 관계하는 항목은 이하와 같습니다.



- « **ISASRVs** » DDE 서버명에서 's' 는 slave 번호입니다.
- « **SYSTEM** » 표준 토픽명에서 토픽명 « TOPICS » 을 지정하기 위해 필요합니다.
- « **TOPICS** » 정의되어 있는 토픽명 리스트를 표시합니다. 이것은 실행중인 ISaGRAF Target 어플리케이션명을 가리킵니다.
- « **어플리케이션명** » ISaGRAF 어플리케이션명입니다.
- « **TOPICITEMLIST** » 지정된 토픽명에서 DDE 로 선택가능한 리스트를 표시합니다.
- « **변수명** » 어플리케이션 변수명입니다.

예 : Excell 의 cell 에서 ISaGRAF 어플리케이션 "rfwash" 변수

_____ "waterlevel" 값을 표시

= ISASRV1|rfwash!waterlevel

DDE advise loop rate : ISaGRAF NT target: -d 옵션

DDE client 에서 DDE request 는 변수의 수가 많아지면, Forwarding 을 위한 오버헤드가 커집니다. 이것에 대해 어드바이스 loop 는 사용자 스스로 변경한 변수만을 client 에 넘깁니다. 이 방법에서 통신 오버헤드를 최소한으로 확보할 수 있습니다. 이 모드에서는 서버가 정기적으로 어드바이스 loop 에 지정된 변수의 변화가 일어나지 않았는지 체크합니다. 이 체크의 주기를 DDE 어드바이스 loop water(?)라 합니다.

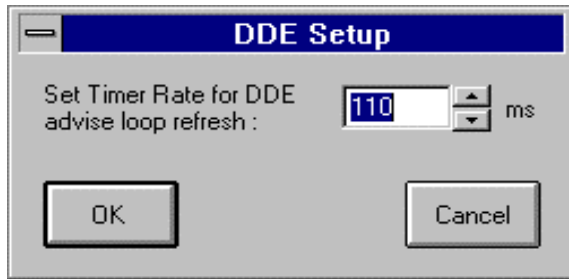
이 옵션을 사용해서 DDE 어드바이스 loop 설정시간을 ms (미리 초) 단위로 설정 가능합니다.

기본값 : 기본값은 1000 [ms]또는 ISaGRAF.INI 파일에 지정되어 있는 값입니다.

예 :

WISAKER -d=100

사용자 I / F : NT-Target 의 메인 창 「**옵션**」 - 「**DDE**」 명령어에서 설정할 수 있습니다.



에러관리와 출력 메시지

ISaGRAF Target 소프트웨어에는 에러 검출 처리가 포함되어 있습니다. 에러 메시지와 그 설명은 본 매뉴얼 뒷장 에서 해설합니다.

에러 검출의 흐름은 이하와 같습니다.

- 에러와 에러번호는 ISaGRAF 에러 루틴으로 전송됩니다.
- 만약, workbench측에서 코드 생성 메뉴로 에러 처리 Flag 가 set 되어 있으면 에러 처리되지만, 그렇지 않는 경우는 에러 정보는 무시됩니다.

에러가 처리되는 경우 :

- 에러 번호 (10 진수) 와 인수 (16 진수) 가 WISAKER.EXE 창에 표시됩니다.
- 이 에러 번호와 인수는 ring FIFO BUFFER 에 등록됩니다. BUFFER 사이즈는 workbench 측에서 코드 생성 메뉴에서 설정 가능합니다. 만약 FIFO BUFFER 가 가득 차서 새로운 에러가 발생하면, 가장 오래된 에러는 없어집니다.
- 에러 정보는 workbench 디버거창 또는 어플리케이션 내의 SYSTEM call Function 에서 꺼낼 수 있습니다.

workbench 디버거가 에러를 검출한 경우는 에러 메시지 창에 에러 메시지가 표시됩니다. 여기서는 어플리케이션 실행상태 (실행중, 정지중) 표시에 더해서 에러가 발생한 프로그램명과 변수명, 에러번호, 인수를 [] 내에 표시합니다.

작동시 메시지는 WISAKER.EXE 창에 표시되지만, 이것은 slave 번호, 통신 link 설정, DDE 서버명에서 이루어집니다.

☐ 시스템 clock

NT Target 은 하드웨어 고유의 영향을 받지 않도록 사이즈의 동기(Sync)와 타임 변수 refresh 에는 표준 타임인 10 [ms]를 사용합니다.

이 경우 10ms 보다 정밀도가 높은 타임 변수를 정의 할 수 없습니다. 같은 이유로 10ms 이하의 사이클 타임을 설정한 경우는 에러 번호 62 인 사이클 타임 over follow 가 발생합니다. 상세한 것은 이하의 절을 참조 바랍니다.

주의 : 또한 본 NT-Target 에서는 PC 하드웨어 시스템에 따라 고정밀도의 MultiMedia timer (1ms) 를 사용 가능한 것에 대응한 시스템 clock 처리부를 집어넣고 있습니다. 이 경우는 1ms 정밀도의 타임 변수를 정의할 수 있습니다.

☐ 주기개요 / target 동작

ISaGRAF Target 사이클 마지막에 (다음 사이클이 시작되기 직전) 이하의 algorithm 으로 처리됩니다. 어플리케이션에 사이클 타임이 지정되어 있던 경우 CPU 는 남은 시간 (지정 사이클 타임-현재 사용한 시간) 이외의 처리를 회전시킵니다. 만약 이 시간이 부족할 때 사이클 over follow 가 되고, 1-tic 분만 기다립니다.

만약, 1-tic 분만 타임이 지정되지 않을 경우 또는 남은 시간이 1-tic 이하인 경우 CPU 는 1-tic 분만 기다립니다.

따라서, NT-Target 의 타이밍 정밀도는 WindowsNT 시스템 tic (예를 들면, 10ms) 에 대응합니다. 통상 사이클 타임을 지정해서 NT 상의 CPU 시간을 ISaGRAF 이외의 처리로 돌리는 것도 가능하게 됩니다.

또한 설정 사이클 타임이 어플리케이션 프로그램에 대해서 충분한 여유가 없으면, 다른 Task 가 겹치기 때문에 주의 바랍니다. 또한, CPU performance 와 메모리 용량이 충분치 않은 경우에도 같은 현상이 일어납니다.

☐ 종료 키

DeskTop PC 에서 NT-Target 을 테스트하고 있는 경우 ISaGRAF Target 의 실행을 멈추기 위해서 키 입력에 따라 예기치 않은 정지를 방지할 수 있습니다. 이때 키 입력은 다음과 같습니다.

alt + F4

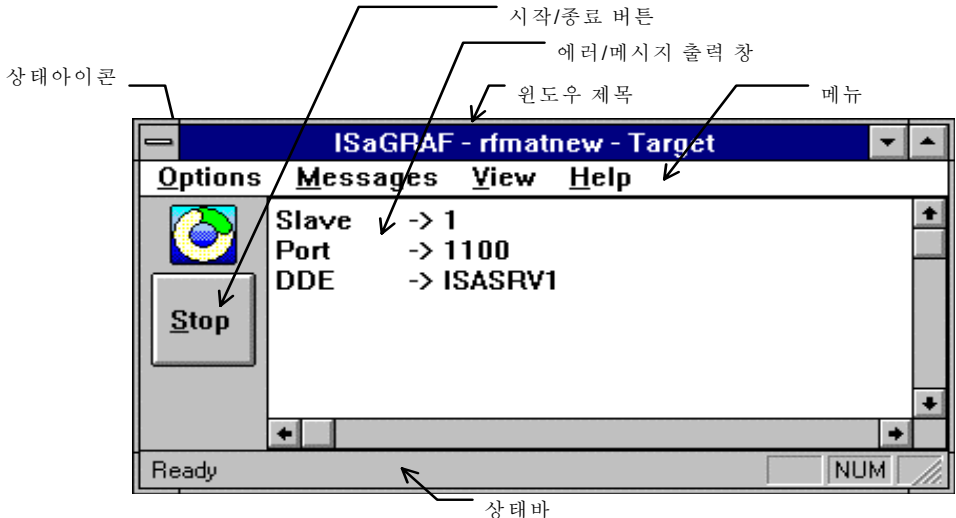
이 정지 방법에서는 I/O 보드의 처리가 클로즈 되지 않고 종료할 위험성도 있습니다. 따라서 통상의 ISaGRAF Target 정지에는 이하의 2 가지 중에 할 것을 권합니다.

- workbench 디버거 start / stop 커맨드를 사용한다.
 - Target (WISAKER.EXE) 창의 STOP 버튼 / 메뉴를 선택한다.
- 이들 중 어떤 경우라도 I/O 보드는 클로즈 처리된 후 종료합니다.

또한, 디버거가 온라인 접속 중에 Target 창을 닫을 경우 디버거가 응답하지 않을 경우가 있는데 이 때는 강제적으로 NT 상에서 종료해 주십시오.

C.6.3 사용자 interface

ISaGRAF 의 NT-Target GUI 의 해설은 아래와 같습니다.:



[주된 구성 요소]

창 타이틀

메뉴바
 상태 아이콘
 시작 / 종료 버튼
 에러 & 메시지 표시
 상태바

창 타이틀 바의 구성은 « ISaGRAF - 어플리케이션명 - target »이 되고, 실행중인 어플리케이션명이 표시됩니다. 만약, 실행되는 어플리케이션이 없는 경우는 « **ISaGRAF - - Target** »표시됩니다.

☞ **ISaGRAF NT target 메뉴바:**

메뉴 바에는 4 개의 메뉴가 있습니다.

Options
 Messages
 View
 Help

● "옵션" 메뉴

「Options」 메뉴에서는 이하의 옵션 설정이 가능합니다.

Slave

는 slave 번호를 변경합니다. 변경된 옵션 값은 다음 Target 에서 다시 start 한 후에 반영됩니다. 이 옵션은 Target 작동시에 명령어 라인에서 옵션 첨가로 작동되는 경우는 사용할 수 없습니다.

Communication

는 통신을 설정합니다. 변경된 옵션 값은 다음 Target 에서 다시 start 한 후에 반영됩니다. 이 옵션은 Target 작동시에 s 옵션 이외의 옵션이 설정된 경우에는 사용할 수 없습니다.

DDE

는 DDE 어드바이스 loop water 를 설정합니다. 변경된 옵션 값은 다음 Target 에서 다시 start 한 후에 반영됩니다. 이 옵션은 Target 작동시에 s 옵션 이외의 옵션이 설정된 경우에는 사용할 수 없습니다.

Simulate I/O

는 체크박스를 tic 하는지 여부를 설정합니다. 이 옵션 값은 다음 Target 에서 다시 start 한 후에 반영됩니다. (단, workbench 상에서 I/O 접속 에디터로 I/O 보드는 virtual 에 설정되어 있을 필요가 있습니다.)

Priority

는 Priority 를 변경합니다. 변경된 옵션 값은 즉시 반영됩니다.

Default

는 이하의 옵션 설정의 디폴트 값을 읽어냅니다.

- Communication
- DDE
- 창 표시장소, 사이즈 조정

변경된 옵션 값은 다음 Target 의 재 start 후에 반영됩니다. 옵션은 Target 작동시에 s 옵션 이외의 옵션이 설정된 경우엔 사용할 수 없습니다.

• "Messages" 메뉴

「Messages」 메뉴는 표시를 관리합니다.

Acknowledge 는 에러표시가 되어 있는 경우를 확인하기 위한 명령어입니다. Status 아이콘의 빨간불이 꺼집니다.

Clear 는 표시되어 있는 모든 메시지를 창에서 청소합니다.

• 「View」 메뉴

「View」 메뉴는 어플리케이션 상태를 표시합니다.

Information 는 Target 구성과 어플리케이션 정보 Dialog Box 를 열 수 있습니다. (상세한 것은 나중에 기술)

• 「Help」 메뉴

「Help」 메뉴는 온라인 document 표시가 가능합니다.

☐ **ISaGRAF NT target 아이콘:**

상태 아이콘은 NT-Target 의 상태를 표시합니다.

- 어플리케이션 실행중에 아이콘은 회전합니다.

- 어플리케이션이 없는 경우 (또는 어플리케이션이 정지중) 는

아이콘 회전은 정지합니다.

- 에러 메시지가 표시되어 있을 때는 아이콘 중심에서 빨간 불이 켜졌다 꺼졌다 합니다. 이 점멸이 멈춘 경우는 « Messages »메뉴 « Acknowledge »명령어를 선택하든지 « Clear »명령어를 선택합니다. 에러에 관한 정보는 나중에 해설합니다.

이하에서 아이콘 상태를 나타냅니다.

:

	에러 없음	에러 표시 있음 (중심부가 빨각게 점등)
Application 실행		
application 없음		

ISaGRAF NT target 시작/종료 버튼:

N T Target 창 의 시작/종료 버튼은 디버거의 시작/종료 명령어와 같은 기능을 가집니다. 버튼상의 텍스트는 Target 상태를 반영하고 있습니다. 즉, Target 가 실행 중 일 때 버튼상의 텍스트는 « Stop » 이라 표시되고, Target 이 정지 중 일 때는 « Start » 라고 표시됩니다. 만약, 어플리케이션이 없는 경우 « Start » 표시가 되어 있을 때에 버튼을 눌러도 다시 « Start » 표시로 되돌아옵니다.

ISaGRAF NT target, 일반 정보

「View」 - 「Information」 커맨드

N T Target 창에서 「View」 - 「Information」 커맨드에 따라 이하의 Dialog Box 에 따른 Target 구성과 실행중인 어플리케이션의 구성 정보를 표시할 수 있습니다.

ISaGRAF NT kernel : Global Information

General Setup

Slave number:

Communication:

DDE Setup

Advise loop rate: ms

Server name:

Topics (Items) names:

Application

Status:

Running mode:

Code size: bytes

Data size: bytes

이 Dialog Box 에는 3 개의 토픽이 포함되어 있습니다.

a) **Target 설정 :**

- slave 번호
- 통신 설정 (통신 link 가 isanet 의 경우 포트번호에 덧붙인 N T 상에서 사용 가능한 I P 번호도 표시됩니다.)

b) **DDE 설정 :**

- 어드바이스 loop 설정시간
- DDE 서버명
- DDE 토픽명과 아이템명 (통상은 < > 내부는 실제 어플리케이션명과 변수명 등을 바꿀 수 있습니다.)

c) **어플리케이션 :**

- 어플리케이션이 실행중 일 때는 그 어플리케이션 이름, 만약 어플리케이션이 없는 경우는 'No application' 문자가 표시됩니다.
- 어플리케이션 실행 모드를 표시합니다. 구체적으로 어플리케이션이 중간 코드 (T I C 코드) 를 소프트웨어 처리 (interpreter) 되면서 실행하고 있는 경우는 « TIC code / Software processed »로 표시됩니다. 또, C 소스코드를

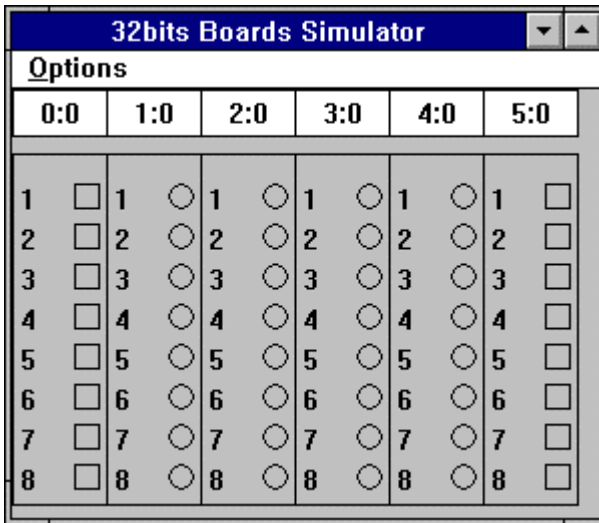
처음으로 compile 되어 시작하고 있는 경우는 « C compiled » 라 표시합니다.
만약, 어플리케이션이 없는 경우는 « No application »이라 표시됩니다.

- 중간 코드 사이즈를 바이트 표시합니다. 만약, 실행 모드가 « C compiled »가 된 경우 이 필드는 0 이 됩니다.

데이터 사이즈를 바이트 표시합니다 이 데이터 사이즈는 Target realtime 데이터 사이즈와 변수 테이블 사이즈의 합계를 가리킵니다.

ISaGRAF NT target 가상 보드 시뮬레이션:

N T Target 창의 「Option」 - 「Simulate I/O」 커맨드가 선택되어 있는 경우는 어플리케이션이 start 하면 이하의 시뮬레이션 창이 열립니다.



I/O 접속 에디터 구성 (I/O 보드의 수, 변수명의 수 등) 에 따라 이 창은 변경을 받습니다. 위의 번호 < s : b > 는 슬롯 번호와 보드 번호 (식별자) 를 나타냅니다. 이 번호는 0 에서 시작하고 있습니다. 보드가 virtual 또는 시뮬레이션 보드가 선택되어 있는 동시에 « Simulate I/O »가 체크되어 있을 때, 이 시뮬레이터 창은 Start/Stop 버튼으로 열렸다 닫혔다 합니다.

이 창은 I/O 루틴 call 에 의해 호출됩니다.

시뮬레이터 창의 「Options」 메뉴는 두개의 항목을 포함하고 있습니다.

Variable names 은 변수명을 보드 오른쪽 옆에 표시합니다. 단, 어플리케이션 심벌 테이블이 미리 중간 코드 앞에 다운로드 되어 있을 때로 한합니다.

Hexadecimal values 는 각 정수형 값을 16 진수 표시가 가능합니다. Default 표시는 10 진수입니다.

예를 들면, 변수 명을 표시하는 것은 아래와 같습니다.

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" 프로그래밍

C.7.1 개요

본 매뉴얼은 이미 ISaGRAF 개념을 이해하고, workbench 의 사용 경험을 가진 사용자를 대상으로 합니다. ISaGRAF 표준 라이브러리내의 **변환 function** , **C언어 평선**, **C 언어 function block** 을 사용해 자동화 어플리케이션을 개발 가능하게 되면, 다음은 “사용자정의” 변환 function , C언어 평선, C언어 function block 의 개발이 가능합니다. 이것은 사용자가 새로운 라이브러리를 작성하고, ISaGRAF 타겟의 기능을 확장하는 것을 의미합니다.

C언어 개발 환경과 몇 개의 C언어 프로그램의 경험이 있으면, 사용자는 이 매뉴얼을 사용해서 ISaGRAF 타겟 제어의 최적의 customize 가 가능하게 됩니다. 따라서 타겟 PLC의 성능뿐만 아니라 ISaGRAF workbench 이용 환경의 질적인 향상도 가능합니다.

이 문서내의 정보는 특정 타겟 시스템에 한정된 것은 아닙니다. 그러나, 싱글 태스크 시스템에는 적용할 수 없는 것이 있습니다. (멀티 tasking 기능을 요구하는 등)

표준 ISaGRAF Workbench 특징

ISaGRAF workbench 는 많은 C언어 라이브러리관리기능을 제공합니다. 자동화 어플리케이션측에서 보면 C언어 변환 function , C언어 평선, C 언어 평선 블록은 “**black box**” 이고, 이 interface 에 의해서만 정의됩니다.

ISaGRAF 라이브러리관리는 기존 라이브러리에 새로운 component 를 추가하고 새로운 **ST/FBD** 프로그램에서 사용하기 위한 interface 정의를 위해 사용합니다. 또한 ISaGRAF 라이브러리관리는 C언어 변환 function , C언어 평선, C언어 function block 인 C언어 소스 코드의 template 생성과 편집기능을 가지고 있습니다. 라이브러리관리에 관한 더욱 자세한 정보는 **ISaGRAF 사용자가이드 (섹션 A)** 를 참조해 주십시오.

"C" 언어에 의한 개발

ISaGRAF workbench 에는 C언어 컴파일러와 크로스컴파일러는 포함되어 있지 않습니다. 사용자는 ISaGRAF Kernel 에 C언어 컨포넌트를 짜넣은 타겟 시스템에 적합한 C언어 컴파일러를 별도로 준비해야만 합니다.

크로스컴파일러를 사용하는 경우에는 ISaGRAF workbench 는 사용자가 정의하는 MS-DOS 커맨드 파일 (*.bat) 을 실행하기 위해 프로그램 매니저에 등록되어 있는 아이콘을 사용할 수 있습니다. 이 때 사용하는 크로스컴파일러는 DOS 창에서 뛰어야만 합니다. 만약 그렇게 하지 않을 경우는 Windows 를 종료하고 순수한 DOS 상에서 컴파일러, link 를 실행해 주십시오. 또한 workbench 「툴」 메뉴를 Customize 해서 사용자커맨드를 workbench 에 등록하는 것도 가능합니다. (상세한 것은 workbench 사용자가이드 섹션A을 참조)

기술 메모

ISaGRAF 라이브러리 매니저에서는 사용자가 각각의 라이브러리에 텍스트 형식의 설명을 기술할 수 있게 되어 있습니다. 이 기술메모는 어플리케이션 개발자를 위한 사용자 s 가이드이고, 어플리케이션으로 바르게 변환 function 와 function , function block 을 기술하기 위한 유익한 정보가 됩니다.

변환 function , C언어 평션, function block 을 어플리케이션 개발자가 그것들을 ISaGRAF function 로 사용하기 위해서는 기술메모 중에서 사용되는 법이 정의되어야 합니다.

C언어 평션의 기술메모는 이하와 같이 기술되어야만 합니다.

- ☐ function 처리내용의 상세한 설명
- ☐ 호출 인수 (입력파라미터) 설명
- ☐ 되돌리기 값 (출력파라미터) 설명
- ☐ 호출 인수, 되돌리기 값의 상세한 설명
- ☐ 사용법

C언어 function block 을 위한 기술메모는 이하와 같이 기술되어야 합니다.

- ☐ 블록처리 내용의 상세한 설명
- ☐ 호출 인수 (입력파라미터) 설명
- ☐ 되돌리기 값 (출력파라미터) 설명
- ☐ 호출 인수, 되돌리기 값의 상세한 설명
- ☐ 사용법

변환 function 를 위한 기술메모는 이하와 같이 기술되어야 합니다.

- ☐ 입력변수에 사용하는 경우 정확한 의미
- ☐ 출력변수에 사용하는 경우 정확한 의미

❑ 처리 가능한 값의 제한 (한계)

또한 기술메모에는 이하와 같은 정보도 필요하게 됩니다.

- ❑ 변환 function , function , function block 분별용 식별자
- ❑ 유지와 갱신을 위한 정보
- ❑ 지원하는 타겟 시스템
- ❑ 멀티태스크 시스템 특유의 특징
- ❑ 요구되는 시스템 서비스와 메모리, 드라이브 등

C.7.2 "C" 변환 functions

ISaGRAF workbench 에는 심플한 아나로그 입출력 변환을 타겟상에서 실행하기 위해 **선형변수 테이블**이 준비되어 있습니다. 이 변환 테이블은 엄밀히 연속증가 또는 연속감소 기능에 한정되어 있습니다. 전혀 C언어에 의한 개발은 필요 없습니다. 이들 톨에 관한 상세한 설명은 ISaGRAF 사용자가이드를 참조해 주십시오.

변환 테이블로 대응할 수 없는 경우는 C언어로 특수한 조작을 기술하는 것으로 사용자는 어떤 복잡한 변수에서도 이용 가능하게 됩니다. 기본적으로 변환 function 는 입력 및 출력변수에 대해 정의 가능합니다. 어느쪽 일방적으로 정의되지 않아도 시스템 crush 를 막기 위해 실장과 테스트는 ISaGRAF Kernel 변환 function 를 짜넣기 전에 해야만 합니다.

변환 function 는 C언어로 기술되고 컴파일러 되고, ISaGRAF Kernel 과 link 됩니다. 새로운 기능이 확장된 Kernel 은 그 확장된 기능을 프로젝트로 사용하기 전에 타겟상에 설치되어야만 합니다. 새로운 변환 function 기능은 workbench simulator 에서는 확인할 수 없습니다. ISaGRAF 어플리케이션은 비표준 변환 function 를 사용하기 전에 simulation 해 둘 필요가 있습니다.

ICS Triplex ISaGRAF Inc 사가 제공하는 표준 변환 function 인 C언어 소스 코드는 ISaGRAF workbench 에 편성되어 있습니다. 그것들은 새로운 변환 function 를 작성하기 위해 예로서 사용 가능합니다. 그러나, 여러가지 어플리케이션으로 그것들을 사용하기 위해서는 그 변환 변환 function 를 개조해야 할 것입니다. 표준 변환 function 는 ISaGRAF simulator 에서도 지원되고 있습니다.

주의 : 변환 function 는 어플리케이션의 입출력의 처리와 같은 시기에 I / O 매니저에 따라 실행됩니다. 변환 function 의 처리시간은 ISaGRAF 어플리케이션 사이클 타임에 포함됩니다. 사이클 타임을 필요 이상으로 늘이지 않기 위해서 사용자는 변환 function 중에서 창 operation 을 절대로 사용하지 않는 것이 필요합니다.

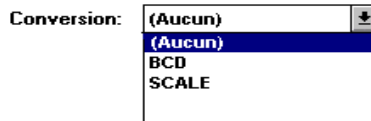
☐ **ISaGRAF 라이브러리에 function 추가**

ISaGRAF 라이브러리 매니저는 workbench 상의 ISaGRAF 라이브러리에 새로운 변환 function 를 추가하기 위해 사용합니다. 변환 function 라이브러리를 선택해서 “파일 - 신규작성” 메뉴를 선택합니다. 변환 function 는 미리 정의된 interface 를 사용해서 workbench 상에서는 파라미터 없이 정의합니다.

새로운 변환 function 를 작성하려면 그 **기술메모**를 기술해 주십시오. 새로운 변환함수인 C 언어 template 가 자동으로 작성됩니다.

☐ **ISaGRAF 프로젝트에서 conversion 사용**

정의된 변환 function 는 입출력 아나로그 변수 파일로서도 사용할 수 있습니다. 변수에 변환 function 를 부가하기 위해서는 모음 에디터를 작동해 아나로그 입출력을 선택해 파라미터 를 입력합니다. “**변수**” 의 필드가 setup 을 위한 것입니다.



리스트에는 변환 function 와 변환 테이블 양쪽이 나옵니다. 같은 이름을 상○하는 것은 불가능합니다. ISaGRAF Kernel 에 기능 확장 또는 정의되지 않은 변환 function 를 변수에 부가할 수 없습니다.

☐ **표준 "C" interface**

변환 function interface 는 항상 같은 포맷을 가집니다. 인수와 되돌리기 값은 구조체에 따라 주고받게 됩니다. 이 구조체는 "TACN0DEF.h"으로 정의되어 있습니다.

```

/*
  Name: tacn0def.h
  Target conversions definition file
*/

#define DIR_INPUT 0           /* direction = input conversion */
#define DIR_OUTPUT 1         /* direction = output conversion */

```

```

typedef int32  T_ANA;           /* integer ANA type          */
typedef float  T_REAL;         /* real ANA type             */

typedef struct {                /* conversion structure      */
    uint16 number;              /* conversion number (reserved) */
    uint16 direction;           /* conversion direction       */
    T_REAL *before;             /* value before conversion    */
    T_REAL *after;              /* value after conversion     */
} str_cnv;

#define ARG_BEFORE (*(arg->before))
#define ARG_AFTER  (*(arg->after))
#define DIRECTION  (arg->direction)

/* eof */

```

"str_cnv" 구조체는 interface 를 정의하고 있습니다. 변환 function C언어의 인수는 이 구조체의 버턴입니다. 구조체 멤버 "**number**"는 변환 function 번호 (ISaGRAF 라이브러리 위치) 이고, 프로그램 중에서 사용할 필요는 없습니다.

구조체 멤버 "**direction**"는 변환 function 가 입력변수에 이용되고 있는지 출력 변수에 이용되고 있는가를 나타냅니다. 그 값은 입력 변환 시는 **DIR_INPUT**, 출력 변환시는 **DIR_OUTPUT** 이 됩니다.

구조체 멤버 "**before**"는 변환 전의 값을 나타냅니다. 이 멤버 변수는 입력 변환인지 출력 변환인지에 따라 다른 의미를 가집니다. "**direction**"이 **DIR_INPUT** 인 경우 입력 변환 function 에 대해서는 입력 디바이스를 읽은 값이 됩니다. "**direction**"이 **DIR_OUTPUT** 인 경우 출력 변환 function 에 대해서는 프로그램된 논리식에 따른 값이 됩니다.

구조체 멤버 "**after**"는 변환 후의 값을 나타냅니다. 이 멤버 변수는 입력 변환인지 출력 변환인지에 따라 다른 의미를 가집니다. "**direction**"이 **DIR_INPUT** 인 경우 입력 변환 function 에 대해서는 프로그램된 논리식에 따른 값이 됩니다. "**direction**"이 **DIR_OUTPUT** 인 경우 출력 변환 function 에대해선 출력 디바이스에 출력하는 값이 됩니다.

프로그램은 **before** 와 **after** 구조체 멤버를 직접 액세스 하기위해"**ARG_BEFORE**" 와 "**ARG_AFTER**" 정의어를 사용할 수 있습니다. 가공된 변수는 **단정도 유동 소수점형** 변수입니다. 변환을 정수의 아나로그 값에 적용된 경우 변환된 결과는 Long 형의 정수치가 됩니다. 이것은 같은 변환 function 가 실수, 정수의 양쪽 아나로그 입출력 값에 대해 사용할 수 있다 것을 의미합니다.

소스 코드

변환 function 는 입출력 변수 양쪽에 사용되는 것으로 C 소스 코드는 입력 변환부와 출력 변환부로 나뉘집니다. 구조체 멤버 "**direction**"는 이중에 어느것을 선택하기 위해 사용됩니다. ISaGRAF 라이브러리매니저는 자동으로 C 언어 소스 코드의 테이블을 생성해 줍니다. **IF** 문의 구조에 따라 각기 call 됩니다. 이하의 리스트가 변환 function C 언어 소스 코드의 template 입니다.

```
/*
    conversion function
    name: sample
*/

#include <tasy0def.h>
#include <tacn0def.h>

void CNV_sample (str_cnv *arg)
{
    if (DIRECTION == DIR_INPUT) { /*INPUT CONV*/
    }
    else { /*OUTPUT CONV*/
    }
}

/*
```

이하의 function 는 변환 function 이름에 따라 ISaGRAF I / O매니저와 link 를 정의하고 있습니다. 이 function 는 ISaGRAF libaray 매니저에 의해 자동으로 생성됩니다. */

```
UFP cnvdef_sample (char *name)
{
    sys_strcpy (name, "SAMPLE"); /* 변환 function 이름을 주다 */
    return (CNV_sample); /* function 되돌리기 값 지정 */
}
```

function 기능을 완성시키기에 가장 좋은 방법은 입력변환과 출력변환을 따로 local function 로 기술하는 것입니다. 이 function 들은 리스트에 나타나는 것과 같이 메인 루틴 **IF** 문의 구조에 따라 각각 call 됩니다.

include 파일 "**TASY0DEF.H**"에는 ttmxpa 에 의존하는 정의가 포함되어 있습니다. 그중에는 **UFP** 형도 정의되어 있습니다. 이것은 void 형 function 포인터를 가리키고, function 선언을 위해 사용됩니다.

≡ projects 와 "C" 의 연결

변환 function 실행측과 ISaGRAF 프로젝트에서의 변환 function 와의 연속부치기는 변환 function 이름에 따라 됩니다. 어떤 종류의 선언 function 부가 변환 function C 소스 코드에 부가됩니다. 이 선언 function 부는 어플리케이션이 시작한 때 한번만 실행되고, ISaGRAF/I/O 매니저에 변환 function 이름을 전합니다. 이하에서는 표준적인 선언 선언 function 의 포맷을 나타냅니다.

```
UFP cnvdef_xxx (char *name)
{
    strcpy (name, "XXX");          /* 변환 function 명*/
    return (CNV_xxx);              /* function 되돌리기 값*/
}
/* (xxx 는 변환 function 명) */
```

strcpy 문 안에서 사용되는 function 의 이름은 **대문자**이어야만 합니다. 그래서 변환 function 의 코드중 및 선언 function 중의 이름은 **문자**이어야만 합니다.

"CNV_"과 **"cnvdef_"** 접두어를 사용해서 기존의 C 언어의 예약어와 ISaGRAF 라이브러리이름에 간섭하는 일없이 사용자가 function 에 이름을 부칠 수 있습니다.

기타 처리도 변환 function 의 초기화를 실현하기 위해 선언 function 부에 추가 가능합니다. ISaGRAF 시스템에서 이 선언 function 부는 어플리케이션을 시작할 때 한번만 call 되는 것에 주의 해 주십시오.

편성된 변환 function 를 위한 선언 function 는 예를 들면, ISaGRAF 어플리케이션 중에서 사용되고 있지 않아도 call 됩니다. ISaGRAF 어플리케이션에서 사용되고 있는 변환 function 가 Kernel 에 편성되지 않는 경우는 치명적인 에러를 일으킬 수 있기 때문에 주의해 주십시오.

새로운 function 를 Kernel 과 link 하기 전에 사용자는 **"GRCN0LIB.C"**라는 이름의 소스 코드를 편성해 리스트에 새로운 function 를 추가해야만 합니다. **"GRCN0LIB.C"**에는 단 function 정의의 배열만 있을 뿐입니다. 이 배열은 C 언어로 기술된 변환 function 와 동적으로 link 하기 위해 어플리케이션이 초기화될 때 읽을 수 있습니다. 이하에 그 파일의 예를 나타냅니다.

```
/* File "GRCN0LIB.c" - 변환 function 의 표준 라이브러리에 */

#include <tasy0def.h>                                /* required for types definition */
```

```

extern UFP cnvdef_scale (char *name);      /* declaration function for SCALE
conv */
extern UFP cnvdef_bcd (char *name);        /* declaration function for BCD conv */

UFP_LIST CNVDEF[ ] = {                    /* array of declaration functions for */
                                           /* integrated conversion functions */

    cnvdef_scale,
    cnvdef_bcd,

    NULL };

/* end of file */

```

CNVDEF 배열은 NULL 버튼으로 끝나야 합니다. 이 조건이 충족되지 않으면 실행시에 crash 가능성이 있습니다. **CNVDEF** 배열이 정의되지 않으면, 새로운 ISaGRAF Kernel 를 link 생성할 때 “미정의(未定義) 변수에러”가 일어납니다.

이 파일에 function 명을 추가 기술함으로 새로운 Kernel 은 기존의 변환 function 에 추가되는 형으로 만들어집니다. 또한 **CNVDEF** 배열에 프로젝트로 사용하는 변환 function 만 기술하고, 그 프로젝트에 특화(特化)한 Kernel 에 카스트 마이즈하는 것도 가능합니다. “**GRCN0LIB.C**” 파일은 어플리케이션 코드가 생성될 때 ISaGRAF 코드 generate 에 따라 자동적으로 생성됩니다. 이 파일은 ISaGRAF 프로젝트 디렉토리로 만듭니다. 그리고 이 프로젝트에서 사용되고 있는 변환 function 만을 그룹화하고 있습니다.

제한

ISaGRAF 라이브러리는 최대 **128** 개의 변환 function 를 가질 수 있습니다. 어떤 타입의 처리에서도 변환 function 중에서 실행 가능하지만, 변환 function 는 ISaGRAF 사이클에 같은 시기에 호출되고, function 실행시간은 직접 사이클 타임에 영향을 주기 때문에 대단한 주의가 필요합니다.

C.7.3 "C" Functions

C언어 평선은 표준 ST 와 FBD 언어의 확장을 위해 사용합니다. C언어 평선에 따라 시스템 call 를 시작해서 통신 ISaGRAF 어플리케이션과 다른 태스크와의커뮤니케이션을 위한 서비스를 실행할 수 있습니다. C언어로 기술된 function 는 compile 되고 ISaGRAF Kernel 과 link 되어 확장된 Kernel 이 만들어 집니다. 그 확장된 기능을 프로젝트로 사용할 때는 타겟상에 새롭게 기능 확장된 Kernel 이 설치되어야만 합니다.

새로운 C 언어 평선 기능은 simulator 에는 편성되지 않습니다. 따라서, ISaGRAF 어플리케이션은 비표준 function 를 사용하기 전에 simulation 해둘 필요가 있습니다.

주의 : C 언어 평선은 어플리케이션 사이클로 같은 시기에 ISaGRAF Kernel 에 의해 실행됩니다. function 의 처리시간은 ISaGRAF 어플리케이션 사이클 타임에 직접 영향을 줍니다. 사이클 타임을 필요 이상으로 늘이지 않기 위해선 사용자는 변환 function 중에서 창 operation 을 사용하지 말아야 합니다.

ISaGRAF 라이브러리에 function 추가

ISaGRAF 라이브러리 매니저는 workbench 상의 ISaGRAF 라이브러리에 새로운 C 언어 평선을 추가하기 위해 사용합니다. C 언어 평선 라이브러리를 선택해서 “파일 - 신규” 메뉴를 선택합니다. 새로운 function 를 작성하려면 그 기술메모를 기술해 주십시오. 새로운 function C 언어 소스 코드 template 가 자동적으로 작성됩니다.

메뉴 “편집 - 파라미터” 를 선택해 function 입력 파라미터 와 출력 파라미터 를 정의합니다.

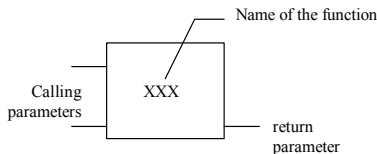
ISaGRAF 프로젝트에서 C 언어 function 의 사용

편성된 C 언어 평선은 ISaGRAF 프로젝트의 프로그램 중에서 표준 function 와 같은 형식으로 사용 가능합니다. C 언어 평선은 ST, FBD, SFC 언어 가운데 호출해서 사용 가능합니다.

ST 언어 중에서 C 언어 평선을 call 은 function call 과 같은 문법으로 됩니다. function 입력 파라미터 는 function 명 뒤 괄호 안에 씁니다. 또 인수가 복수일 때는 콤마로 구분합니다. 그래서 그 표현 그 자체가 되돌리기 값이 됩니다. C 언어 평선은 대입문과 복잡한 표현에 사용됩니다. 이하는 대입문으로 사용된 C 언어 평선의 예를 나타냅니다.

result := ProcName (par1, par2, ... parN);

FBD 언어에서 C 언어 평선을 call 하려면 표준 function 와 같은 식으로 사용 가능합니다. 그 입력 파라미터 는 function block 의 좌측에 접속되고, 출력 파라미터 는 우측에 접속됩니다. 이하는 function block 의 표준 형태를 나타냅니다.



C언어 평선은 예를 들면, **SFC**function block (**P** 와 **N** 의 속성을 가졌음) 또는 transition 에 추가되는 조건문에서 호출 됩니다.

☐ C언어function interface 의 정의

ISaGRAF 라이브러리매니저 「편집」 메뉴의 「파라미터 」 커맨드로 새로운 C언어 평선 입력 파라미터 와 출력 파라미터 를 정의합니다. function 는 최대 **31** 의 입력 파라미터 와 항상 하나의 출력 파라미터 를 가집니다. 아래 다이어로그박스는 C언어 평선파라미터 를 기술하기 위한 것입니다.

모든 ISaGRAF 데이터형이 파라미터 로서 사용 가능합니다. (boolean 형 값, 정수형 값, 실수형 값, 타임형 값, 가변장 문자열형 값) 정수와 실수형은 정확하게 설정해야만 합니다. 아래의 표는 ISaGRAF 데이터형과 C언어형의 대응을 나타냅니다.

boolean 형	unsigned long	Unsigned 32 bit word: 1=true / 0=false
정수형	long	Signed integer 32 bit word
실수형	float	Single precision floating value
타임형	unsigned long	Unsigned integer 32 bit word (unit is 1 millisecond)
가변장 문자열형	char *	Character string.

가변장 문자열형 값이 C언어 평선에 전송된 때 NULL 캐릭터를 포함할 수 없습니다. C 소스 코드로 전달되는 문자열은 NULL 캐릭터로 터미네이트(?) 되고 있습니다. 출력 파라미터 는 리스트 중에 마지막에 있어야 합니다. 파라미터 터미네이트는 이하에 표시된 룰을 따라야만 합니다.

- 최대 1 6 문자
- 최초 문자는 영문자 (a~z)이어야만 합니다.
- 이후 문자는 문자, 숫자, 언더스코아(_)이어야만 합니다.
- 대문자 소문자 구별은 없습니다.

function 파라미터 에 같은 이름은 사용할 수 없습니다. 입력 파라미터 에 출력 파라미터 과 같은 이름을 사용할 수 없습니다. 같은 이름의 파라미터 는 다른 function 내에선 사용 가능합니다. 디폴트 출력 파라미터 의 이름은 "**Q**"입니다. 이 이름은 자유롭게 변경 가능합니다. 파라미터 명은 C언어 소스 코드 내에서의 변수명에 대응하고 있습니다.

"**삽입**" 버튼은 선택되어 있는 파라미터 앞에 새로운 파라미터 를 삽입하기 위해 사용합니다. "**삭제**" 버튼은 선택되어 있는 파라미터 를 삭제하기 위해 사용합니다. "**arrange**" 버튼은 출력 파라미터 가 리스트 마지막에 되도록 자동으로 파라미터 나열을 바꿉니다. "**OK**"버튼을 누르면 function 에 관한 interface 정의를 보존하고, 다이아로그박스를 닫습니다. "**캔슬**" 버튼은 function 에 관한 interface 정의를 변경시키지 않고 다이아로그박스를 닫습니다..

Function "C" interface

function 의 interface 는 파라미터 에 의존합니다. 입력 파라미터 와 출력 파라미터 는 구조체로서 전달됩니다. 이 구조체는 "**GRUS0nnn.H**" 라는 파일로 정의됩니다. "**nnn**"는 ISaGRAF 라이브러리매니저가 관리하는 function 의 번호입니다. 이하에 삼각 function "**SIN**"을 계산시키는 C언어 평선의 예를 나타냅니다.

```
/* 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 */
```

ISaGRAF 데이터형과 C언어형의 대응을 아래표에서 나타냅니다. ISaGRAF 데이터형은 C언어형으로서 정의 파일로 정의되고 있습니다.

boolean 형	T_BOO	long (32 bits)
정수형	T_ANA	Long
실수형	T_REAL	float (32 bits - single precision)
타임형	T_TMR	Long
가변장 문자열형	T_MSG	char * (32 bits - char pointer)

"**str_arg**"구조체 각각의 멤버는 function 파라미터 각각에 대응합니다. 되돌리기 값은 구조체의 마지막 멤버입니다. 인수는 function 파라미터 정의에서 설정한 순서로 리스트에

나옵니다. 대문자의 정의문자는 `function` C언어로의 실장에 있어서 직접파라미터에 액세스하기 위해 정의되어 있습니다. 이름은 ISaGRAF 라이브러리매니저를 사용해 정의한 것으로 되어 있습니다.

C언어 정의 파일은 ISaGRAF 라이브러리매니저를 사용해서 `function`의 `interface`를 변경할 때마다 갱신됩니다. 따라서 `function`의 실장과 ISaGRAF 프로그램 사이의 완전한 정합성(整合性)을 보증합니다.

소스 코드

이하는 C언어 평선 소스 코드의 표준적인 template를 나타냅니다.

```
/* Example of 사용자 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 */
}

/*
```

이하의 `function`는 C언어 평선의 초기화와 그 선언을 위해 사용됩니다. 이 `function`명에 따라 Kernel과 link가 실현됩니다. 이 C언어 평선은 ISaGRAF 라이브러리매니저를 사용해서 작성합니다. */

```
UFP uspdef_sample (char *name)
{
    strcpy (name, "SAMPLE");          /* 평선명 */
    return (USP_sample);              /* 평선 되돌리기 값 */
}

/* end of file */
```

"TASY0DEF.H" include 파일에 의존하는 정의를 위해 필요합니다. 또 `void`형 `function` 포인터를 나타내는 `UFP`형의 정의도 포함하고 있습니다. 이것은 `function`의 선언을 위해 사용됩니다.

프로젝트와 C언어 연결

C언어 평선 실행측과 ISaGRAF 프로젝트 내에서의 C언어 평선과의 연속 부치기는 C언어 평선 이름에 따릅니다. 어떤 종류의 선언 `function`부가 C언어 평선의 C소스 코드에

부가됩니다. 이 선언 function 부는 어플리케이션이 시작될 때 한번만 실행되고, ISaGRAF Kernel 에 C 언어 평선의 이름을 전합니다. 이하는 표준 선언 function 포맷을 나타냅니다.

```
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) */
```

strcpy 문 안에서 사용되는 function 의 이름은 **대문자**이어야만 합니다. 그리고 function 의 실장을 위해 선언 function 중의 이름은 소문자 이어야만 합니다. "**CSP_**"와 "**uspdef_**" 접두어를 사용해서 기존의 C언어의 예약어와 ISaGRAF library 이름에 간섭하는 일 없이 사용자가 function 에 이름을 기입할 수 있습니다.

기타 처리도 C 언어 평선의 초기화를 실행하기 위해 선언 function 부에 추가 가능합니다. ISaGRAF 시스템에서 이 선언 function 부는 어플리케이션이 시작될 때 한번만 call 되는 것을 주의 해 주십시오.

편성된 function 를 위한 선언 function 는 예를 들면, ISaGRAF 어플리케이션 중에서 사용되고 있지 않아도 call 됩니다. ISaGRAF 어플리케이션에서 사용되고 있는 function 가 Kernel 에 편성되어 있지 않는 경우는 치명적인 에러를 일으킬 수 있기 때문에 주의해 주십시오.

새로운 function 를 Kernel 과 link 하기 전에 사용자는 "**GRUS0LIB.C**"라는 소스 코드를 편집해서 리스트에 새로운 function 를 추가해야만 합니다. "**GRUS0LIB.C**"에는 단, function 정의의 배열이 있을 뿐입니다. 이 배열은 C언어로 기술된 function 와 동적으로 link 하기 위해 어플리케이션이 초기화될 때 읽을 수 있습니다. 이하는 그 파일의 예를 나타냅니다.

```
/* 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 USPDEF[ ] = {      /* array of declaration functions */
                                /* for integrated functions */
    uspdef_fc1,
    uspdef_fc2,
    uspdef_fc3,
    uspdef_fc4,
```

```
NULL };

/* end of file */
```

USPDEF 의 배열은 NULL 포인터로 끝나야 합니다. 이 조건이 충족되지 않으면 crash 의 가능성이 있습니다. **USPDEF** 배열이 정의되지 않으면 새로운 ISaGRAF Kernel 을 link 생성할 때에“ 미정의 변수 에러” 가 일어납니다.

이 파일에 function 명을 추가하면 새로운 Kernel 은 기존 function 에 추가되는 형으로 작성됩니다. 또 **USPDEF** 배열에 프로젝트로 사용하는 function 만 기술하고, 그 프로젝트에 특화(特化)한 Kernel 에 카스트 마이즈하는 것도 가능합니다. "**GRUS0LIB.C**" 파일은 애플리케이션이 생성될 때 ISaGRAF 코드 generate 에 따라 자동으로 생성됩니다. 이 파일은 ISaGRAF 프로젝트 디렉토리에 만들어집니다. 그리고, 프로젝트에서 사용되고 있는 function 만 그룹화하고 있습니다.

≡ 제한

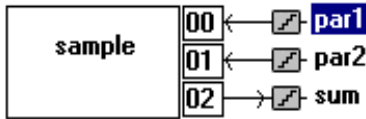
ISaGRAF 라이브러리는 최대 **255** 개의 C언어 평선을 가질 수 있습니다. 어떤 타입의 처리에서도 C언어 평선 중에서 실행 가능하지만, C언어 평선은 ISaGRAF 사이클에 같은 시기에 호출되기 때문에 C언어 평선의 실행 시간은 접속 사이클 타임에 영향을 주기 때문에 상당한 주의가 필요합니다.

≡ C언어function 의 예

이하에 덧셈을 실행하는 "**sample**" function 를 예로 프로그램의 흐름을 소개합니다. 이하의 리스트가 function 기술메모입니다.

이름:	SAMPLE
기술:	실수 (아나로그 변수) 가산
작성일:	1995 12 24
작자:	ICS Triplex ISaGRAF Inc
입력 파라미터 :	par1, par2: 실행 아나로그
출력 파라미터 :	sum 실수 아나로그
proto type:	sum := sample (par1, par2);

다음에 나타내는 것은 function interface 입니다.



다음은 C 언어 소스 코드의 헤드 파일입니다.

```
/* File: GRUS0255.h - C 언어 function 정의 - Name: sample */

/* 표준 ISaGRAF 데이터 타입의 정의 */

typedef long T_BOO;
typedef long T_ANA;
typedef float T_REAL;
typedef long T_TMR;
typedef char *T_MSG;

/* 입력/출력 파라미터 구조체의 정의 */

typedef struct {
    T_ANA _par1;    /* 입력 파라미터 #1 */
    T_ANA _par2;    /* 입력 파라미터 #2 */
    T_ANA _sum;     /* 출력 파라미터 */
} str_arg;

/* 입력/출력 파라미터를 액세스하기 위한 식별자 */

#define PAR1        (arg->_par1)
#define PAR2        (arg->_par2)
#define SUM         (arg->_sum)

/* end of file */
```

이하의 리스트는 C 언어 소스 코드입니다.

```
/* File: GRUS0255.c - C 언어 function - Name: SAMPLE */

#include "tasy0def.h"    /* required for types definition */
#include "grus0255.h"    /* C function source header */

/* C 메인 루틴 : 덧셈 계산 부분 */

void USP_sample (str_arg *arg)
```

```

{
    SUM = PAR1 + PAR2;
}

/* ISaGRAF Kernel 과의 다이내믹 link 의 정의*/

UFP uspdef_sample (char *name)
{
    strcpy (name, "SAMPLE");
    return (USP_sample);
}
/* end of file */

```

C.7.4 "C" FUNCTION BLOCKS

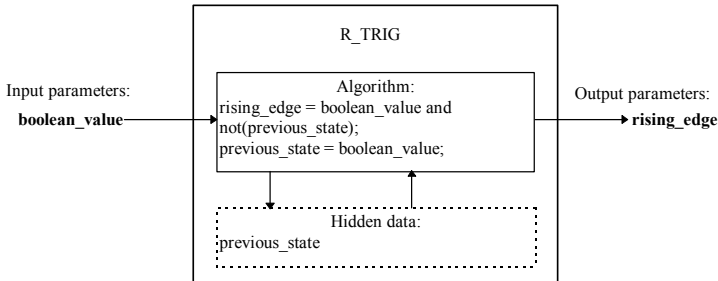
C 언어 function 블록은 static 데이터와 그 처리를 관련 짓는 것입니다. 그것들은 static 데이터 처리가 가능한 완전한 C 언어 function 입니다. C 언어 function 는 통상은 표준 ST 와 FBD 언어 확장을 위해 사용합니다. 데이터의 가공을 위해 사용되는 C 언어 function 과 다르고, function 블록은 static 데이터 처리가 가능합니다. 따라서, function 블록의 algorithm 은 시계열(時系列)데이터 관리도 가능하게 됩니다.

C 언어로 기술된 function 블록은 컴파일러되고 ISaGRAF Kernel 과 link 됩니다. 새롭게 확장된 기능을 프로젝트에서 사용하기 위해서는 기능확장된 Kernel 이 타겟상에 설치되어 있을 필요가 있습니다. 새로운 function 블록의 기능은 simulator 에서 확인 할 수 없습니다. ISaGRAF 어플리케이션은 비표준 function 블록을 사용하기 전에 simulation 해둘 필요가 있습니다.

주의 : function 블록은 어플리케이션에서 같은 시기에 ISaGRAF Kernel 에 의해 실행됩니다. function 블록의 처리시간과 호출 사이즈는 ISaGRAF 어플리케이션 사이클 타임에 직접 영향을 줍니다. 허용된 최대 사이클 타임을 늘이지 않기 위해 사용자는 function 블록 안에서 창 operation 을 사용하지 않는 것이 필요합니다.

☞ **function block 선언**

function 블록은 static 데이터와 처리를 종합 시킨 오브젝트입니다. 이하의 "R_TRIG" function 블록은 boolean 형 값 변수의 오르기 edge 를 검출하는 function 블록의 예입니다.



위의 그림에서 숨겨진 static 변수 "**previous_state**"는 edge 를 검출하기 위해 필요한 데이터입니다. 이 변수는 어플리케이션 안에서 사용되는 복수 "**R_TRIG**" function 블록에서 각각 다른 데이터이어야만 합니다.

ST 언어로 사용되는 function 블록 instance 는 모음중에 선언되어 있어야만 합니다. 왜냐하면, function 블록은 내부에 숨겨진 데이터를 가지고 있기 때문입니다. 각각의 function 블록 copy (instance) 는 하나 하나 다른 이름이어야만 합니다. function 블록의 이름은 라이브러리매니저로 정의합니다. 정의된 function 블록 instance 의 이름 부치기는 모음 에디터로 합니다.

FBD 언어로 사용하는 function 블록은 선언이 필요 없습니다. 왜냐하면, ISaGRAF 의 FBD 에디터는 자동으로 사용한 function 블록 instance 를 선언해 주기 때문입니다. FBD 에디터에 따라 자동적으로 선언된 function 블록 instance 는 항상 편집중인 프로그램의 **local 변수**가 됩니다.

ISaGRAF 라이브러리에서 function block 추가

ISaGRAF 라이브러리 매니저는 workbench 상의 ISaGRAF 라이브러리에 새로운 C언어 function 블록을 추가하기 위해 사용합니다. C언어 function 블록 라이브러리를 선택해서 「파일」 - 「신규작성」 메뉴를 선택합니다. 새로운 function 블록을 작성하려면 그 **기술메모**를 기술해 주십시오. 새로운 function 블록의 C언어 소스 코드의 template 는 ISaGRAF 라이브러리 매니저에 의해 자동으로 작성됩니다. 메뉴 「편집」 - 「파라미터」를 선택해서 function 블록의 입력 파라미터와 출력 파라미터를 정의합니다.

ISaGRAF 프로젝트에서 "C" function block 사용

편성된 C언어 function 블록은 ISaGRAF 프로젝트의 프로그램으로 사용할 수 있게 됩니다. C언어 function 블록은 ST 와 FBD 언어에서 call 해서 사용 가능합니다.

ST 언어 중에서 C언어 function 블록을 call 하면 function call 과 같은 문법이 됩니다. function 블록 입력 파라미터 (인수) 는 function 명 뒤의 괄호에 씁니다. 또, 복수의 경우는 콤마로 구분합니다. 출력 파라미터 (되돌리기 값) 에 액세스하려면 function 명과 출력 파라미터 명에서 편성되어 만들어진 컴퍼넌트명을 사용합니다. 컴퍼넌트명은 function 명과 출력 파라미터 명을 dot(.)로 구분해서 나타냅니다. 예를 들면 컴퍼넌트명

FBINSTNAME.parname

은 "FBINSTNAME"라는 이름의 function 블록 "parname"라는 이름의 출력 파라미터 를 표시합니다.

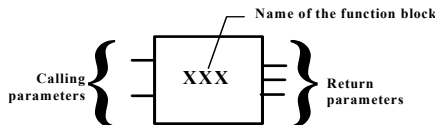
ST 언어중에 사용되는 function 블록 instance 는 모음중에서 선언되어야만 합니다. function 블록 각각의 copy (instance) 는 unite 한 이름으로 식별되어야만 합니다. 이하는 ISaGRAF 모음에서의 function 블록 instance 선언의 예를 나타냅니다.

instance :TRIG1 TRIG2	type:	R_TRIG R_TRIG
--------------------------	-------	------------------

다음에 이것들이 선언된 instance 를 ST 언어 중에서 사용한 예를 나타냅니다.

```
TRIG1 (boo_input1);
TRIG2 (boo_input2);
Command := (TRIG1.Q & TRIG2.Q);
```

FBD 프로그램은 어떤 C언어 function 블록에서도 call 가능합니다. function 는 표준 function 블록과 같은 형식으로 다릅니다. 이 입력 파라미터 는 좌측에 접속되고, 그 출력 파라미터 는 우측에 접속됩니다. 표준 function 블록을 다음과 같이 나타냅니다.



FBD 언어로 사용하는 function 블록은 선언할 필요는 없습니다. ISaGRAF FBD 에디터는 자동으로 사용된 function 블록 instance 를 선언하고 있기 때문입니다. 자동적으로 선언된 function 블록 instance 는 항상 편집되고 있는 프로그램의 local 변수가 됩니다. 이하에 앞의 ST 언어의 예를 FBD 언어로 프로그램한 예를 나타냅니다.

☐ "C" function block 정의

ISaGRAF 라이브러리매니저의 「편집」 메뉴 「파라미터」 커맨드로 새로운 C 언어 function 블록 입력 파라미터와 출력 파라미터를 정의합니다. function 는 최대 32 인 인수 (입력 파라미터와 출력 파라미터의 합계) 를 갖고 있습니다. C 언어 function 블록과 다른 function 블록은 복수 출력 파라미터를 가질 수 있습니다. 아래 다이아로그박스는 C 언어 function 블록 파라미터를 기술하기 위한 것입니다.

다이아로그박스의 상부 리스트는 C 언어 function 블록의 파라미터를 나타내고 있습니다. 여기서 function 파라미터의 순서는 function call proto 타입을 기초해서 가장 위쪽에 입력 파라미터가 있고, 그 다음에 출력 파라미터가 됩니다. 다이아로그박스 하부는 선택중인파라미터를 상세하게 기술하기 위해 사용됩니다.

- 파라미터 이름
- 입력파라미터 / 출력파라미터의 선택
- 파라미터 변수형

모든 ISaGRAF 데이터형이 파라미터로서 사용 가능합니다. (boolean 형, 정수형, 실수형, 타임형, 가변장 문자열형) 정수와 실수형은 구별할 필요가 있습니다. 아래 표에 ISaGRAF의 데이터형과 C 언어형의 대응을 나타냅니다.

boolean 형	unsigned long	Unsigned 32 bit 워드: 1=true / 0=false
정수형	long	Signed integer 32 bit 워드
실수형	float	단정도 유동소수점 값
타임형	unsigned long	Unsigned integer 32 bit word (단위 1 ms)
가변장 문자열형	char *	문자열

가변장 문자열형 값이 C 언어 function 블록에 전송될 때 문자열에는 NULL 캐릭터를 포함할 수 없습니다. 왜냐하면, C 소스 코드에 전달되는 문자열은 NULL 캐릭터에서 터미네이트되어 있기 때문입니다. 출력 파라미터는 리스트 중에 마지막에 있어야 합니다. 파라미터의 터미네이트는 이하에 나타내는 룰에 따라야만 합니다.

- 최대 16 문자
- 최초의 문자는 영문자 (a~z)이어야만 합니다.
- 이후의 문자는 문자, 숫자, 언더스코어(_)이어야만 합니다.
- 대문자, 소문자의 구별은 없습니다.

function 블록파라미터 에 같은 이름은 사용할 수 없습니다. 입력파라미터 에 출력파라미터 와 같은 이름을 사용할 수 없습니다. 같은 이름의파라미터 는 다른 function 내에서는 사용 가능합니다. 파라미터 명은 C 언어 소스 코드내에서의 변수명에 대응하고 있습니다.

"**삽입**" 버튼은 선택된 파라미터 앞에 새로운파라미터 를 삽입하는 것에 사용합니다. "**삭제**" 버튼은 선택된파라미터 를 삭제하는 데 사용합니다. "**arrange**" 버튼은 출력파라미터 가 리스트의 가장 마지막이 되도록 자동으로 파라미터 나열을 바꿉니다. "**OK**" 버튼을 누르면 function 에 관한 interface 정의를 보존하고, 다이아로그박스를 닫습니다. "**캔슬**" 버튼은 function 에 관한 interface 정의를 변경하지 않고 다이아로그박스를 닫습니다.

Function block "C" interface

function 블록 interface 는 그 파라미터 정의에 의존합니다. 인수는 구조체로서전달됩니다. 이 구조체는 "**GRFB0nnn.H**"라는 파일에서 정의됩니다. "**nnn**"는 ISaGRAF 라이브러리매니저가 관리하는 function 의 번호입니다. 되돌리기 값은 역시 "**GRFB0nnn.H**"라는 파일로 정의된 번호에 따라 표현됩니다.

이하에 "**LIM_ALARM**" function 블록 C 언어 interface 의 예를 나타냅니다.

```
/* function block interface - name: sample */

/* 표준 ISaGRAF 데이터 타입 */

typedef long T_BOO;
typedef long T_ANA;
typedef float T_REAL;
typedef long T_TMR;
typedef char *T_MSG;

/* 입력 파라미터 구조체 */

typedef struct {
    /* CALL */ T_BOO _par1;
    /* CALL */ T_BOO _par2;
} str_arg;

/* str_arg 구조체로 액세스 */

#define PAR1 (arg->_par1)
#define PAR2 (arg->_par2)

/* 출력 파라미터 논리번호 */

#define FBLPNO_Q1 0
```

```
#define FBLPNO_Q2 1

/* end of file */
```

ISaGRAF 데이터형과 C 언어형의 대응을 아래 표로 나타냅니다. ISaGRAF 데이터형은 C 언어형으로서 정의파일로 정의되고 있습니다.

boolean 형	T_BOO	long (32 bits)
정수형	T_ANA	Long
실수형	T_REAL	float (32 bits – 단정도 유동소수)
타임형	T_TMR	Long
가변장 문자열형	T_MSG	char * (32 bits – char 포인터)

"str_arg" 구조체 각각의 멤버는 function 파라미터 각각에 대응합니다. 되돌리기 값은 구조체의 마지막 멤버입니다. 인수는 function 파라미터 정의로 설정한 순번으로 리스트에 나옵니다. 대문자의 정의문자는 function C 언어로 실장에 있어 직접 파라미터에 액세스하기 위해 정의되어 있습니다. 이름은 ISaGRAF 라이브러리매니저를 사용해서 정의한 것으로 되어 있습니다.

출력 파라미터 (되돌리기 값) 에 적용된 번호 순은 function 파라미터 정의로 설정한 것입니다. 제 1의 출력파라미터의 논리번호는 항상 0입니다.

C 언어 소스 프로그램에 있어서는 출력 파라미터를 나타내는데 논리번호 없이 정의된 식별자가 사용되어야만 합니다. 따라서 interface 정의를 변경해도 소스파일의 재컴파일러를 쉽게 할 수 있게 됩니다.

C 언어의 정의 파일은 ISaGRAF 라이브러리매니저를 사용해서 function 블록 interface를 변경할 때마다 갱신됩니다. 이것은 function 블록 실장과 ISaGRAF 프로그램 사이의 완전한 정합성을 보증합니다.

소스 코드

function 블록 C 언어에서의 실장은 세개의 엔트리 포인트로 나눠집니다.

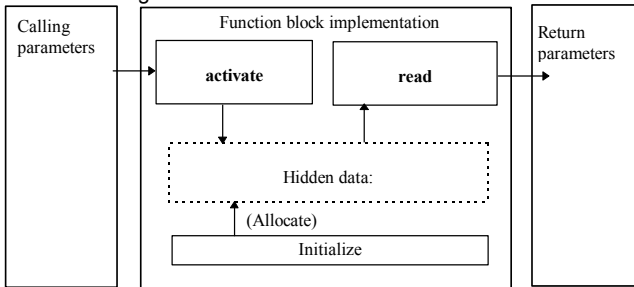
- 초기화 서비스
- 실행 서비스 - 입력 파라미터에 대한 처리
- 출력 파라미터 읽어 내기 서비스

같은 function 블록 각각의 instance 에는 동일 코드가 사용됩니다. 코드는 copy 되어 사용된다는 의미는 아닙니다. static 데이터는 각각의 instance 에 관련되어 있습니다. 이 static 데이터는 ISaGRAF 프로그램에 의한 직접 액세스는 불가능합니다. 이것은 function 블록 instance 의“ local 변수 즉, 숨겨진 변수 (hidden variable) ” 로서 유지됩니다.

“ 실행 서비스 ” 는 타겟 사이클 룰에 instance 가 사용될 때마다 부릅니다. 그래서, 입력 파라미터 를 처리하고 관련된 데이터를 갱신합니다. 이 서비스는 function 블록 메인 처리를 나타냅니다.

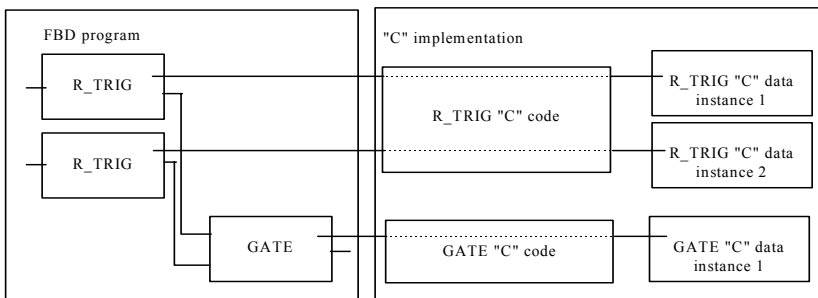
출력 파라미터 의“ 읽어내기 서비스 ” 는 ISaGRAF Kernel 에 따라 어떤 instance 의 되돌리기 값의 현재치를 읽기 위해 부릅니다. 특별한 계산 등은 이 서비스에는 없습니다. 단, 숨겨진 데이터를 ISaGRAF 어플리케이션에 전송하는 것만 처리됩니다.

Functional diagram:



• function 블록의 static 데이터

function 블록은 static 데이터와 그 처리에 관련합니다. 데이터 구조는 같은 function 블록 각각의 instance 에 배당됩니다. ST 언어와 FBD 중에서 사용될때마다 하나의 데이터 구조와 그 instance 가 배당됩니다. 다음 예에서 C언어 데이터 구조와 FBD 로 사용된 function 블록 instance 과의 관련을 나타냅니다.



각각의 instance 데이터 구조가 필요로 하는 메모리는 ISaGRAF 시스템에 따라 확보됩니다.
instance 데이터 구조체를 나타내는 포인터가 “실행” 과 “읽어내기” 서비스일 때 전달됩니다.

ISaGRAF 라이브러리메니저는 자동으로 데이터 구조체 정의에 필요한 C 언어 소스 코드 template 를 생성합니다. 데이터 구조체는 항상 "**str_data**"라는 이름이 됩니다. 프로그래머는 interface 호환성을 보증하기 위해서도 이 이름을 바꾸지 않는 것이 좋을 것입니다. 숨겨진 데이터 (hidden data) 는 일반적인 되돌리기 값의 이미지와 함께 내부 변수로서 그룹화되고 있습니다. function 블록의“ 읽어내기” 서비스는 되돌리기 값에 액세스하기 위해서만 사용됩니다. 그리고, 다른 처리를 위해서는 사용할 수 없습니다.

- 초기화 서비스

function 블록의 “초기화” 서비스는 ISaGRAF Kernel 에 따라서 어플리케이션이 시작한 때 한번 call 됩니다. C 언어 프로그래머는 이때 시스템에 대해 하나의 instance 에 필요한 메모리 사이즈를 지정할 수 있습니다. 이하에 표준적인 “초기화” 서비스를 나타냅니다.:

```
uint16 FBINIT_xxx(uint16 hinstance)
{
    /* "xxx" ≐ F B G */
    return (sizeof(str_data));
}
```

"**hinstance**"인수는 instance 번호입니다. 이것은 ISaGRAF 내부 처리에 사용되고, 프로그램에는 사용할 수 없습니다. 초기화 서비스는 하나의 instance 에 필요한 데이터의 바이트를 되돌립니다. 요구하는 메모리 양은 **64K** 바이트를 넘어서는 안됩니다. 초기화 서비스에서는 다른 처리는 불가능합니다. 초기화의 C언어 소스 코드는 function 블록이 만들어졌을 때에 자동적으로 ISaGRAF 라이브러리매니저가 작성합니다.

- **실행 서비스**

“ 실행 ” 서비스는 어플리케이션에서 사용되는 function 블록 각각의 instance 에 대해 매회 매회 타겟 사이클 중에서 call 됩니다. 이 서비스는 입력파라미터 데이터를 처리하고, 숨겨진 static 데이터 (hidden data) 와 출력파라미터 를 갱신하기 위해 function 블록을 메인 처리합니다. 이하는 표준적인 “ 실행 ” 서비스 template 를 나타냅니다.

```
void FBACT_xxx (
uint16 hinstance,           /* "xxx" 는 F B 명 */
                             /* instance 논리번호 */
str data *data,             /* data: instance 데이터에서의 포인터 */
```

```

    str_arg *arg                /* arg: 입력파라미터 구조체에서의 포인트 */
    )
    {
        /* ..... */
    }

```

"**hinstance**" 인수는 instance 의 번호입니다. 이것은 ISaGRAF 내부처리에서 사용되고 프로그램에는 사용할 수 없습니다. "**data**"인수는 instance 구조체 Far 형 포인트입니다. "**arg**" 인수는 인수값을 가진 구조체 Far 형 포인트입니다. 프로그래머는 "**arg**" 구조체 멤버에 액세스하기 위해서는 C 언어 헤드에 정의한 식별자를 사용할 필요가 있습니다.

“ 실행 ” 서비스 algorithm 은 입력파라미터 ("**arg**"구조체 멤버) 의 처리와 "**data**"구조체 데이터를 갱신합니다. 이하는 **R_TRIG**function 블록 (오르기 edge 검출) 의 “ 실행 ” 서비스를 나타냅니다.

```

/* FB 에 등록되어 있는 C 헤드 정의 */

typedef struct {                /* 입력파라미터 */
    T_BOO _clk;                /* trigger 입력 */
} str_arg;

#define CLK    (arg->_clk)

/* FB instance 데이터 구조체 */

typedef struct {
    T_BOO prev_state;          /* trigger 입력 の直前の狀態 */
    T_BOO edge_detect; /*오르기검출 신호. 출력파라미터 이미지 */
} str_data;

/*실행 서비스 */

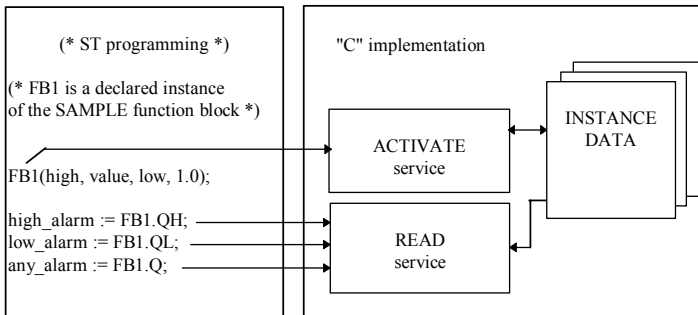
void FBACT_trig (uint16 hinstance, str_data *data, str_arg *arg)
{
    data->edge_detect = (T_BOO)(CLK && !data->prev_state);
    data->prev_state = CLK; /* calling 파라미터 */
}

```

C 언어 소스 코드 template 는 function 블록이 만들어졌을 때 자동적으로 ISaGRAF libart 매니저가 작성합니다.

• 출력 파라미터 읽어내기 서비스

“읽어내기” 서비스는 ST 언어와 FBD 로 function 블록의 되돌리기 값이 참조될 때마다 call 됩니다. 하나의 출력 파라미터 를 얻기 위해 사용됩니다. 다음 선편은 “읽어내기” 서비스가 ST 프로그램을 실행중에 실행되고 있는 모양을 표현하고 있습니다.



같은 출력 파라미터 에 대한 “읽어내기” 서비스는 동일 사이클에서 한번 이상 call 될수 있기 때문에 이 서비스 중에 특별한 처리는 불가능합니다. 숨겨진 데이터를 ISaGRAF 어플리케이션에 보내는 처리만을 합니다. 이하는 표준적인 “읽어내기” 서비스 template 를 나타냅니다.

/* 출력 파라미터 값을 형변환 (CAST) 해서 copy 해 처리 */

```
#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)
```

/* 출력 파라미터 읽어내기 서비스: 각 출력 파라미터 마다 읽어내기 */

```
void FBREAD_xxx (      /* "xxx" 는 FB 명 */
uint16 hinstance,     /* instance 논리번호 */
str_data *data,       /* instance 데이터 구조체에서의 포인트 */
uint16 parno,         /* 출력 파라미터 논리번호 */
void *value)          /* 파라미터 값을 copy 해 buffer */
{
    switch (parno) {
```

```

        case FBLPNO_XX: /* ... */ break;
        case FBLPNO_YY: /* ... */ break;
        /* .... */
    }
}

```

"**hinstance**" 인수는 instance 번호입니다. 이것은 ISaGRAF 내부처리에 사용되고, 프로그램에는 사용할 수 없습니다. "**data**" 인수는 instance 구조체 Far 형 포인터입니다.

"**parno**" 인수는 요구된 출력 파라미터 의 논리번호입니다. 인수의 값을 가진 구조체 Far 형 포인터입니다. 출력 파라미터 의 식별을 위해 C언어 헤드에 정의한 식별자를 사용해 주십시오. 이러한 식별자는 "**FBLPNO_**"라는 접두어로 시작됩니다. "**value**" 인수는 액세스한 출력 파라미터 를 copy 하기 위한 buffer 을 나타내는 FAR 포인터입니다. 이 인수로 나타나는 데이터형은 출력 파라미터 ISaGRAF 에서의 데이터형과 일치합니다. 이하의 표는 ISaGRAF 데이터형과 C언어 데이터형을 나타냅니다.

boolean 형	long	32 bit unsigned word (0=false / 1=true)
정수형	long	32 bit signed word
실수형	float	32 bit 단정도 유동소수 값
타임형	long	32 bit unsigned word (단위 : 1ms)
가변장 문자열형	char *	Array of characters

다음 마크는 액세스된 출력 파라미터 의 데이터형에 대응한 buffer 를 액세스하기 위해 사용됩니다.

```

#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)

```

이들은 값과 파라미터 를 ISaGRAF buffer 에 copy 하기 위해 공통적으로 사용되는 처리입니다.

```

/* boolean 형 파라미터 에 대해서 */
*BOO_VALUE = 파라미터 _value;

/* 정수형 파라미터 에 대해서 */
*ANA_VALUE = 파라미터 _value;

/* 실수형 파라미터 에 대해서 */

```

```

    *REAL_VALUE = 파라미터 _value;
/* 타임형 파라미터 에 대해서 */
    *TMR_VALUE = 파라미터 _value;
/* 가변장 문자열형 파라미터 에 대해서 */
    strcpy (*MSG_VALUE, 파라미터 _value);

```

C 언어 소스 코드 template 는 function 블록에서 만들어졌을 때 자동적으로 ISaGRAF 라이브러리매니저가 작성합니다.

• C언어 function 블록 소스 파일 예

이하는 C 언어 function 블록의 표준인 C 언어 template 를 나타냅니다.

```

/* function 블록 (xxx 는 FB 명) */

#include <tasy0def.h>
#include <grfb0nnn.h> /* nnn 는 라이브러리의 F B 배치에 다른 번호 */

/* 각 F B instance 의 숨겨진 데이터에 대한 구조체 */
typedef struct {
    /* 멤버 정의 */
} str_data;

/* 초기화 서비스 : 숨겨진 데이터에 필요한 사이즈를 되돌린다. */
word FBINIT_xxx (uint16 hinstance)
{
    return (sizeof (str_data));
}

/* 실행 서비스: 입력파라미터 처리 */
void FBACT_xxx (uint16 hinstance, str_data *data, str_arg *arg)
{
    /* ... */
}

/* 출력 파라미터 값을 형변환해서 copy */
#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)

/* 출력 파라미터 읽어내기 서비스: 각 출력 파라미터 마다 읽어내기 */
void FBREAD_xxx (uint16 hinstance, str_data *data, uint16 parno, void *value)

```

```

{
    switch(parno)
    {
        case FBLPNO_XX: *??_VALUE = ...; break;
        case FBLPNO_YY: *??_VALUE = ...; break;
        ....
    }
}

```

/ 이하의 template 는 function 블록 선언과 초기화에 사용됩니다. function 블록 이름을 사용해서 ISaGRAF Kernel 과 link 를 연결합니다. 이 template 는 라이브러리매니저에 의해 자동적으로 생성됩니다. */*

```

ABP fbldf_xxx (char *name, IBP *initproc, RBP *readproc)
{
    strcpy (name, "XXX");
    *initproc = (IBP)FBINIT_xxx;
    *readproc = (RBP)FBREAD_xxx;
    return ((ABP)FBACT_xxx);
}

/* end of file */

```

include 파일 "**TASY0DEF.H**"는 시스템에 의존하는 정의가 필요합니다. 여기에는 실장된 서비스에 대한 Far 포인터를 나타내는 데이터형 선언도 포함되어 있습니다.

== 프로젝트와 C언어function Block 과의 연결

C언어 function 블록의 실장측과 ISaGRAF 프로젝트와의 연속 부치기는 그 function 의 이름에 따릅니다. 어떤 종류의 선언 function 부가 C언어 function 블록 의 C소스 코드에 부가됩니다. 이 선언 function 부는 어플리케이션을 시작했을 때 한번만 실행되고, ISaGRAF Kernel 에 C언어 function 블록 이름을 전합니다. 이하는 표준적인 선언 function 의 포맷을 나타냅니다.

```

ABP fbldf_xxx (char *name, IBP *initproc, RBP *readproc)
{
    strcpy (name, "XXX");           /* FB 명 */
    *initproc = (IBP)FBINIT_xxx;    /* 초기화 서비스 */
    *readproc = (RBP)FBREAD_xxx;    /* 읽어내기 서비스 */
    return ((ABP)FBACT_xxx);        /* 실행 서비스 */
}

/* xxx 는 FB 명 */

```

strcpy 문 가운데서 사용되는 function 블록의 이름은 **대문자**이어야만 합니다. 그리고 function 의 실장을 위한 선언 function 내의 이름은 **소문자**이어야만 합니다.

"FBACT_", "FBINIT_", "FBREAD_" , "fbldf_" 와 같은 접두어를 사용함으로써 기존 C 언어의 예약어와 ISaGRAF 라이브러리에 간섭하지 않고, 사용자가 function 에 이름을 기입할 수 있습니다. demf 접두어 이외의 선언 function 를 추가해 사용할 수는 없습니다.

편성된 function 블록의 선언 function 는 예를 들면, ISaGRAF 어플리케이션 내에서 사용되고 있지 않아도 Kernel 에서 call 됩니다. ISaGRAF 어플리케이션에서 사용되고 있는 function 블록이 Kernel 에 편성되지 않는 경우는 치명적인 에러를 일으킬 수 있기 때문에 주의해 주십시오.

새로운 function 블록을 Kernel 과 link 하기 전에 사용자는 " GRFB0LIB.C"라는 이름의 소스 코드를 편성해 리스트에 새로운 function 블록을 추가해야만 합니다. " GRFB0LIB.C"에는 단 선언 서비스 배열이 있을 뿐입니다. 이 배열은 C 언어로 기술된 function 블록과 동적으로 link 하기 위해 어플리케이션이 초기화될 때 읽을 수 있습니다. 이하는 그 파일의 예를 나타냅니다.

```
/* File: grfb0lib.c - F B 편성 */

#include <tasy0def.h>

extern ABP fbldf_fb1(char *name, IBP *init, RBP *read);
extern ABP fbldf_fb2(char *name, IBP *init, RBP *read);

FBL_LIST FBLDEF[] = {
    fbldf_fb1,
    fbldf_fb2,

    NULL };

/* end of file */
```

FBLDEF 배열은 NULL 포인터로 끝나야 합니다. 이 조건이 충족되지 않으면 crash 가능성이 있습니다. **FBLDEF** 배열이 정의되어 있지 않으면 새로운 ISaGRAF Kernel 을 link 생성시에 " 미정의 변수에러" 가 발생합니다.

이 파일에 function 명을 추가함에 따라 새로운 Kernel 은 기존 function 에 추가되는 형으로 만들어집니다. 또, **FBLDEF** 배열에 프로젝트에서 사용하는 function 만을 기술하고, 이 프로젝트에 특화한 Kernel 에 카스트 마이즈하는 것도 가능합니다. "GRFB0LIB.C"파일은 어플리케이션이 생성될 때 ISaGRAF 코드 generate 에 따라 자동적으로 생성됩니다. 이 파일은

ISaGRAF 프로젝트 디렉토리에 작성됩니다. 그리고, 프로젝트에서 사용되고 있는 function 만을 그룹화하고 있습니다.

제한

ISaGRAF 라이브러리는 최대 **255** 개의 C 언어 function 블록을 가질 수 있습니다. 각 타입의 function 블록은 같은 같은 프로젝트 중에 최대 **255 회** instance 를 가질 수 있습니다.

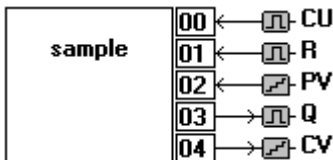
C 언어 function 블록은 ISaGRAF 사이클에서 같은 시기에 호출되기 때문에 C 언어 function 블록의 실행 시간은 직접 사이클 타임에 영향을 주기 때문에 상당한 주의가 필요합니다.

완성 예제

이하는 업카운트를 실행하는 "sample"function 블록을 예로 프로그램의 흐름을 소개합니다. 이하의 리스트가 function 블록의 기술메모입니다.

이름:	SAMPLE
기술:	Up counter
작성일:	01 February 1994
작성자:	ICS Triplex ISaGRAF Inc
입력파라미터 :	CU : counting input R : reset command PV : maximum programmed value
출력파라미터 :	Q : max detection CV : counting result
Proto type:	SAMPLE (count, reset_command, maximum_value); max_detect := SAMPLE.Q; count_result := SAMPLE.CV;

다음은 function block interface 를 나타냅니다.



다음은 C 언어 헤드를 나타냅니다.

```
/* function block interface – FB 명: SAMPLE */
```

```
/* 표준 ISaGRAF 데이터 타입 정의 */
```

```
typedef long T_BOO;
typedef long T_ANA;
typedef float T_REAL;
typedef long T_TMR;
typedef char *T_MSG;
```

```
/* 입력파라미터 구조체 정의 */
```

```
typedef struct {
    T_BOO _cu;
    T_BOO _r;
    T_ANA _pv;
} str_arg;
```

```
/* 입력 파라미터의 액세스용 식별자 */
```

```
#define CU (arg->_cu)
#define R (arg->_r)
#define PV (arg->_pv)
```

```
/* 출력파라미터의 논리번호 첨가 */
```

```
#define FBLPNO_Q 0
#define FBLPNO_CV 1
```

```
/* end of file */
```

이하의 리스트는 C 언어 function 블록의 소스 코드입니다.

```
/* function 블록 – FB 명 SAMPLE */
```

```
#include <tasy0def.h> /*데이터형 정의 */
```

```
#include <grfb0255.h> /* FB 용 C 소스 헤드 */
```

```
/* FB instance 데이터용 구조체 정의*/
```

```
typedef struct {
T_BOO overflow; /* TRUE: if 현재치 >= 프로그램 설정치*/
T_ANA value; /* 카운트 내의 현재치 */
} str_data;
```

/* 초기화 서비스: instance 데이터용 메모리 영역 */

```
word FBINIT_sample (uint16 hinstance)
{
    return (sizeof (str_data));
}
```

/* 실행 서비스: 카운트업 algorithm */

```
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;
}
```

/* 출력파라미터 를 형변환해서 ISaGRAF buffer 에 copy */

```
#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)
```

/* 읽어내기 서비스: 각 출력파라미터 마다 읽어내기 */

```
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->value; break;
    }
}
```

/* ISaGRAF Kernel 과 다이내믹 link 를 위한 선언 function */

```
ABP fbldef_sample (char *name, IBP *initproc, RBP *readproc)
{
    strcpy (name, "SAMPLE");
    *initproc = (IBP)FBINIT_sample;
    *readproc = (RBP)FBREAD_sample;
    return ((ABP)FBACT_sample);
}
```

/* end of file */

C.7.5 컴파일 / 연결 기술

ISaGRAF workbench 및 타겟에는 각 타겟에 대응한 C 컴파일러와 linker 는포함되어 있지 않습니다. 그러나 이 장에서는 ISaGRAF 라이브러리매니저가 생성하는 파일을 이용하기 쉽도록 하는 동시에 컴파일러와 linker 같은 기타 툴 상에서 이들 파일을 이용하기 위해 주요한 테크닉에 대해 설명합니다.

☐ "C" 소스 파일

변환 function , function , function 블록인 C 언어 소스파일은 ISaGRAF 라이브러리매니저가 **ISAWIN\LIB\DEFS** 와 **ISAWIN\LIB\SRC** 디렉토리에 작성됩니다. 이 소스 파일의 이름은 ISaGRAF 라이브러리에 존재하는 변환 function , function , function 블록에 대응하는 번호로 구성됩니다. 이하는 사용되고 있는 파일명을 나타냅니다.

\isawin\lib\defs\TACN0DEF.H	변환 function 정의 파일
\isawin\lib\src\GRCN0nnn.H	변환 function 소스파일
\isawin\lib\defs\GRUS0nnn.H	function 정의 파일
\isawin\lib\src\GRUS0nnn.C	function 소스파일
\isawin\lib\defs\GRFB0nnn.H	function 블록 정의 파일
\isawin\lib\src\GRFB0nnn.C	function 블록 소스 파일

(nnn 는 변환 function , function , function 블록에 대응하는 번호)

주의: 라이브러리 element 의 이름을 바꿔서 copy 한 경우는 그 텍스트와 소스 코드는 ISaGRAF 라이브러리 매미저에 의해 변경되지 않습니다. C 언어 소스 파일을 매뉴얼로 갱신해야만 합니다.

\ISAWIN\LIB\USPNUMS 라는 파일은 ISaGRAF 라이브러리 매니저가 관리하는 C 언어 function 블록 번호와 이름 관계를 나타냅니다. 이 파일의 예를 이하에 나타냅니다. :

```

1    funct_A
10   funct_B
16   funct_C
```

\ISAWIN\LIB\ FBLNUMS 라는 파일은 ISaGRAF 라이브러리 매니저가 관리하는 C 언어 function 블록의 번호와 이름 관계를 나타냅니다. 이 파일의 예를 이하에 나타냅니다. :

```

0    fbl_A
1    fbl_B
2    fbl_C
```

ISAWIN\LIB\ CNVNUMS 라는 파일은 ISaGRAF 라이브러리 매니저가 관리하는 변환 function 의 번호와 이름 관계를 나타냅니다. . 이 파일의 예를 이하에 나타냅니다. :

```
0    SCALE
1    BCD
```

이들 파일은 변환 function , function 블록, C 언어 function 의 작성, 이름의 변경, copy 와 삭제마다 ISaGRAF 라이브러리 매니저에 의해 자동적으로 갱신됩니다. ISaGRAF 코드 generate 는 어플리케이션 코드가 생성되면, 자동적으로 다음 파일을 생성합니다.

isawin\apl\ppp\GRCN0LIB.C	프로젝트에 사용되고 있는 모든 변환 function 의 배열선언
isawin\apl\ppp\GRUS0LIB.C	프로젝트에 사용되고 있는 모든 C 언어 function 의 배열선언
isawin\apl\ppp\GRFB0LIB.C	프로젝트에 사용되고 있는 모든 function 블록의 배열선언.

(**ppp** 는 ISaGRAF 프로젝트의 이름입니다.)

이들 파일은 어떤 프로젝트에서 사용되고 있는 변환 function , function , function 블록만의 기능을 가진 프로젝트 전용의 새로운 ISaGRAF Kernel 을 build 하기 위해 link 시에 사용됩니다.

☐ 소스 코드 다운로드

타겟 시스템이 컴파일러등의 툴을 지원하고 있는 경우는 ISaGRAF 라이브러리 매니저에 의해 작성된 C언어 소스파일 (*.C)과 정의 파일 (*.H)을 타겟 시스템측에 다운로드할 필요가 있습니다. Windows 에 부속 터미널 소프트웨어 등을 사용해서 다운로드합니다.

또, 소스 코드 보존을 타겟 시스템측에서 할 경우는 function interface 부의 변경이 ISaGRAF 라이브러리 매니저로 될때마다 qusrudehlls 정의 파일을 타겟측에 다운로드 해 둘 필요가 있습니다.

workbench 툴 메뉴를 custom 화해 파일의 다운로드 등의 처리를 batch 파일로 등록해 둘 수 있습니다. (상세한 것은 workbench 프로그램 관리)

☐ 크로스 컴파일러 이용

타겟이 workbench 상 P C 의 경우와 타겟 시스템용의 cross 컴파일러환경이 이 P C 상에 있는 경우, 소스 파일을 P C 상에서 직접 관리하는 것도 가능합니다.

이 경우 사용자는 변환 function , C 언어 function, function 블록 소스파일을 완성시키기도 하고, 변경하기 위해서는 ISaGRAF 라이브러리 매니저를 사용합니다. 또, workbench 툴 메뉴를 카스터마이즈해서 컴파일러와 link 를 위한 커맨드를 모아 batch 파일로 등록해 두는 것도 가능합니다.

변환 function , C 언어 function, function 블록이 P C 상에서 컴파일러, link 된 경우 사용자는 이 새로 작성된 ISaGRAF Kernel (새로운 function 이 편성된 Kernel) 을 어플리케이션을 실행하기 전에 타겟 시스템 상에 다운로드할 필요가 있습니다. 만약, 타겟 시스템이 다른 P C 의 경우 플로피 디스크와 네트워크를 이용해서 타겟 시스템 상으로 다운로드합니다.

ISaGRAF kernel 라이브러리와 연결

주의 :

이하의 해설은 일반적인 정보이고, 사용하는 타겟 시스템에 따라 다를 수도 있습니다. 각 타겟에 대한 상세한 타겟 디스크상의 헬프 또는 README 파일을 참조 바랍니다.

ISaGRAF 타겟 시스템에는 변환 function , C 언어 function, function 블록을 컴파일러, link 하기 위해 많은 utility 가 들어 있습니다. "ISACC"라는 커맨드 파일은 컴파일러하기 위한 것입니다. 사용자는 이 파일을 변경할 수 있습니다. ISaGRAF 타겟에는 이하의 두가지 실장 방법이 있습니다.

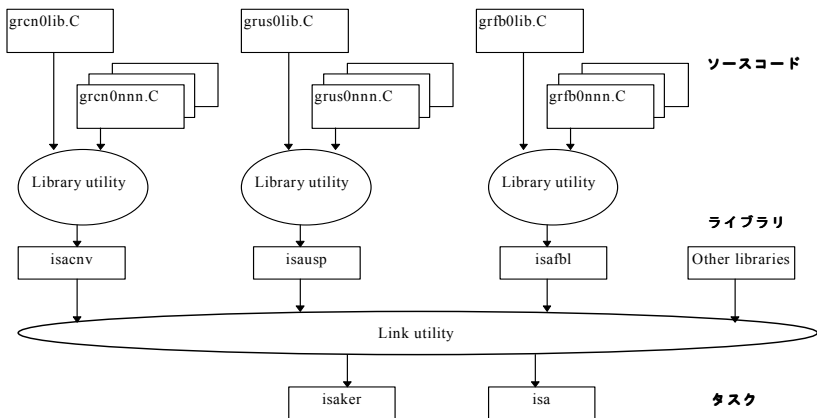
- 싱글태스크 ISaGRAF: 모든 기능이 하나의 프로그램으로 실행됩니다.
- 멀티태스크 ISaGRAF: 통신타스크 (thread) 가 별도로 실행됩니다.

어떤 경우라도 C 언어 부품은 같은 라이브러리에 그룹화되기 때문에 싱글 태스크와 멀티태스크에서도 C 프로그래머에 있어서 차이는 아무것도 없습니다. 사용자 C 언어 function 라이브러리는 link 된 결과 싱글태스크 버전에서는 통상 isa 라는 태스크가 되고, 멀티태스크 버전에서는 통상 isaker 라는 태스크가 됩니다. ISaGRAF 타겟 시스템 내부의 Kernel 부분은 하드웨어에 의존하지 않도록 되어 있습니다. 이 부분에서는 IEC 언어 프로그램을 실행하고, 전용변수 데이터 베이스를 갖고 있습니다.

사용자 function 를 편성한 ISaGRAF Kernel 을 작성할 때 제 1 스텝은 그 프로젝트에 필요한 모든 변환 function , C 언어 function, function 블록 라이브러리를 작성합니다.

라이브러리	내용
ISAUSP	- GRUS0LIB.obj (function 선언 배열) - 편성하는 모든 function 의 오브젝트 파일
ISAFBL	- GRFB0LIB.obj(function 블록 선언의 배열) - 편성하는 모든 function 블록 오브젝트 파일
ISACNV	- GRCN0LIB.obj(변환 function 선언의 배열) - 편성하는 모든 변환 function 의 오브젝트 파일

다음에 프로그래머는 이 새로운 라이브러리들을 다른 여기까지 의존하고 있던 ISaGRAF Kernel 의 오브젝트 파일과 라이브러리와 함께 link 해야만 합니다. 컴파일러, link 되는 모양을 아래에 나타냅니다.



이하에 link 시에 필요한 오브젝트와 라이브러리 리스트를 나타냅니다.

isaker 를 빌드하기위한 모듈:

Object Module: **tast0mai**
Object Module: **tats0com**

Kernel 라이브러리: **isaker**
Kernel 라이브러리: **isaoem**

사용자라이브러리: **isausp** 사용자정의 function

사용자라이브러리:	isafbl	사용자정의 function 블록
사용자라이브러리:	isacnv	사용자정의 변환 function
Kernel 라이브러리:	isasy	
System libraries:	(타겟의 "C" 컴파일러매뉴얼 참조)	

isa 을 만들기 위한 모듈:

Object Module:	tast0mai	
Object Module:	tast0com	
Kernel 라이브러리:	isaker	
Kernel 라이브러리:	isatst	
Kernel 라이브러리:	isaoem	
사용자라이브러리:	isausp	사용자정의 function
사용자라이브러리:	isafbl	사용자정의 function 블록
사용자라이브러리:	isacnv	사용자정의 변환 function
Kernel 라이브러리:	isasy	
System 라이브러리:	(타겟의 "C" 컴파일러매뉴얼 참조)	

프로그래머는 오브젝트와 라이브러리의 순번을 위의 그림에서 나타난 순번이어야만 합니다. 오브젝트와 라이브러리는 타겟 시스템에 따르고 표준적인 확장자 (".lib", ".obj", ".l", ".r"...) 를 가집니다.

⇒ **Required 컴파일러 링크시 필요 옵션**

컴파일러, link 시에 편리한 옵션을 선택하는 것도 가능합니다. 어떤 처리를 변환 function , function , function 블록으로 하는가에 따라 달라지지만, 어떤 처리에선 기타 시스템 라이브러리 (연산 그래픽 등) 도 link 시에 필요하게 됩니다.

ISaGRAF Kernel 의 모든 C언어 소스 파일은 **LARGE** 모델로 컴파일러되고 있습니다. 프로그래머는 같은 메모리 모델로 변환 function , function , function 블록도 컴파일러해야만 합니다.

특별한 정수 (DEFINE) 가 컴파일러를 위해 정의되고 있습니다. 변환 function , function , function 블록의 소스 코드는 시스템에 의존하지 않기 위해 사용됩니다. 이 정수는 타겟 시스템과 CPU타입을 나타내고 있습니다. 이하에 정의의 예를 나타냅니다.

DOS.DOS 베이스 시스템 (INTEL CPU)
ISAWNT Windows-NT 베이스 시스템 (INTEL CPU)
OS9.OS9 시스템 (MOTOROLA CPU)
VxWorks VxWorks 시스템 (MOTOROLA CPU)

ISaGRAF 타겟 소프트웨어와 함께 공급되는 **ISACC** 커맨드 파일에는 그 컴파일러커맨드 중에서 이 편리한 정수가 어느 정도 설정되어 있는지를 나타내고 있습니다.

⇒ 지원되는 컴파일러

변환 function , function , function 블록을 기존의 ISaGRAF Kernel 에 link 하기 위해 이하의 컴파일러를 지원하고 있습니다.

Microsoft MSC 7.00 컴파일러	DOS 타겟
Microsoft MSVC 4.2 컴파일러	Windows-NT 타겟
Microware ULTRA-C 컴파일러	OS-9 타겟
Tornado 1.0; GNU Toolkit 2.6	VxWorks 타겟

그 이외의 개발환경을 사용하는 경우는 연락 주십시오.

⇒ 요약

이하에 새로운 사용자가 변환 function , function , function 블록을 개발할 때 실시해야 할 작업을 정리합니다.

- ⇒1. ISaGRAF 라이브러리매니저를 사용해서 새로운 element (변환 function , function , function 블록) 를 작성합니다. (이름은 커맨드 기입) C언어 template 가 자동으로 작성됩니다.
- ⇒2. ISaGRAF 라이브러리매니저를 사용해서 interface (입력파라미터 , 출력파라미터) 를 정의합니다. 만약, 그 element 가 function 와 function 블록이면 C언어 헤드파일이 자동으로 작성됩니다.
- ⇒3. ISaGRAF 라이브러리매니저를 사용해서 element 의 기술메모를 기술합니다.

- ⇒4. ISaGRAF 라이브러리매니저를 사용해서 변환 function , function , function 블록 처리 algorithm 을 기술한 C언어 소스파일을 완성시킵니다. 여기서, 소스 코드 준비를 모두 완료합니다.
- ⇒5. ISaGRAF 라이브러리매니저 「옵션」 메뉴인 「**논리번호 표시**」를 선택해서, 새로운 element 에 몇번이 매겨지고 있는지 확인합니다. 이 번호는 ".C"와 ".H" 파일명에 관련되어 부쳐집니다.
- ⇒6. ".C" 와 ".H"파일을 ISaGRAF 타겟 라이브러리와 태스크, 컴파일러가 설치되고 있는 타겟 시스템에서 다운로드합니다.
- ⇒7. 새로운 사용자개발 소스 코드 컴파일러를 실행하고, 에러가 나면 수정합니다. "ISACC" 커맨드 파일은 컴파일러를 실행하기 위해 사용 가능합니다.
- ⇒8. "GRXX0LIB.C" 소스파일에 새로운 element 명을 선언 function 부를 추가합니다. 이 파일은 정의되어 있는 element 의 배열입니다. (??는 function 논리번호)
- ⇒9. "GR??0LIB.C" 파일을 컴파일러합니다.
- ⇒10. 라이브러리 작성시에 사용하는 오브젝트 파일 "isaXXliby"에 새로운 element 오브젝트 파일명을 추가합니다.
- ⇒11. 라이브러리를 작성시합니다. 새로운 Kernel 를 작성하기 위해 link 를 실행합니다.
- ⇒12. 새롭게 완성된 Kernel 를 타겟 시스템에 설치합니다.
- ⇒13. 새로운 element 작동과 interface 를 테스트하기 위해 ISaGRAF 샘플 어플리케이션을 작성합니다.

C.8 Modbus 연결

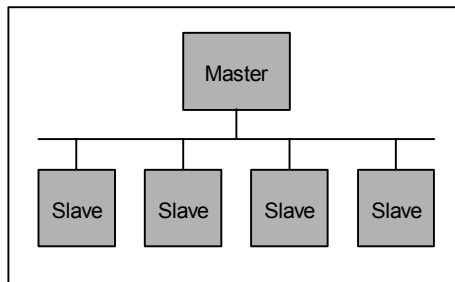
ISaGRAF 어플리케이션이 개발되고, 동작이 확인되면, ISaGRAF 타겟과 예를 들면, S C A D A 시스템 등과 접속할 수 있습니다.

ISaGRAF 는 각종 필드버스, 네트워크와 접속할 수 있지만, 이 항에서는 심플한 네트워크로서 MODBUS protocol 를 사용해서 설명합니다. MODBUS 에는 R S 2 3 2 C 와 R S 4 8 5 시리얼 link 가 필요하게 됩니다.

ISaGRAF workbench 와 타겟을 접속하고 있는 통신 protocol 은 MODBUS 준거하고 있기 때문에 ISaGRAF workbench 를 대신하는 MODBUS 마스터에서 ISaGRAF 타겟 데이터의 읽고 쓰기가 가능합니다.

C.8.1 MODBUS 네트워크 / 프로토콜

MODBUS 네트워크는 한대의 마스터 스테이션과 복수 slave 스테이션으로 구성됩니다. 통상 마스터에는 S C A D A 시스템, slave 에는 P L C가 사용됩니다.



마스터는 각 slave 에 대해서 slave 번호를 사용해 리퀘스트를 보냅니다. 그리고 slave 로 부터의 응답을 기다리고, 다음 리퀘스트를 송신합니다.

각 통신 패킷에는 리퀘스트 번호, slave 번호, 1 6 b i t 체크섬 (CRC) 이 포함되어 있습니다.

리퀘스트 송신 후 타임 아웃 이내에 응답이 되돌아오지 않았던 경우는 지정된 회수분 리퀘스트에 재송신합니다. 그래도 응답이 없는 경우는“ 차단상태” 가 에러를 냅니다.

타임아웃 시간과 재시도 횟수 설정은 마스터 스테이션측에서 해야합니다.

slave 측에서 리퀘스트 처리로 에러가 발생한 경우는 원래 응답 대신에 에러 메시지가 되돌아옵니다.

MODBUS 는 Modicon사의 protocol 이고, 국제표준 규격은 아니기 때문에 MODBUS 호환이라 하더라도 다소 사양의 차이가 있을 수도 있습니다.

예를 들면, 이하와 같은 항목에 차이가 보여집니다.

- 실장되어 있는 function 코드의 리스트
- 어드레스 mapping
- RTU (Binary 코드) 또는 ASCII protocol
- etc...

C.8.2 ISaGRAF 실행

Application 변수 접근

ISaGRAF 타겟 통신은 이하 5 종류의 MODBUSfunction 코드를 지원하고 있습니다.

1	읽어내기 N bits
3	읽어내기 N words
5	적어넣기 1 bit
6	적어넣기 1 word
16	적어넣기 N words

ISaGRAF 타겟의 어플리케이션 변수는 네트워크 어드레스를 통해 액세스됩니다.

이때 workbench 변수모음 에디터에 대한 네트워크 어드레스가 미리 정의되어 있어야 합니다. 변수는 이하의 것을 가리킵니다.

- boolean 형, 정수형, 실수형 변수
- 입력, 출력, 내부변수
- local, 글로벌 변수

boolean 형의 값을 적어넣을 경우 function 코드는 5, 6, 16 이 사용됩니다.

FALSE 변수의 적어넣기는 0, TRUE 변수의 적어넣기에는 0 이외의 값이 적힙니다.

boolean 형의 값을 적어넣을 경우 function 1, 3 이 사용됩니다. function 코드 1에서는 bit 필드가 대응합니다. function 코드 3 이 사용되면, 바이트로 실행된 TRUE 변수는 1 6 진수 0xFFFF 로 읽어냅니다.

정수형의 값을 적어넣을 경우 function 6, 1 6 이 사용됩니다. 정수형의 값은 1 6 bit 정수로 -32768 ~ +32767 까지의 값을 다룹니다. ISaGRAF 타겟 내에서는 32bit 데이터.

정수형의 값을 적어넣을 경우 function 코드 3 이 사용됩니다. 값은 1 6 bit 정수로 -32768 ~ +32767 까지의 값을 취합니다. ISaGRAF 타겟 내에서는 32bit 데이터 표현이므로, 정수형 값이 최대값, 최소값 (-32768 ~ +32767) 을 넘는 경우는 최대, 최소값에 뭉쳐집니다.

실수형 값은 MODBUS 리퀘스트에서는 액세스할 수 없습니다.

주의:

- ISaGRAF 변수는 ISaGRAF workbench 모음에서 네트워크 어드레스가 설정되어 있는 경우에 한해서 액세스 가능합니다.
- ISaGRAF 타겟은 여러 코드를 관리하지 않습니다. 따라서 여러가 발생한 경우라도 마스터에 대한 응답은 오지 않습니다.

약칭:

Slv	slave 번호
Nbw	Words 수
Nbb	Bytes 수
Nbi	Bits 수
AddH	네트워크 어드레스 (High Byte)
AddL	네트워크 어드레스 (low byte)
vH	값 (High Byte)
vL	값 (low byte)
V	바이트 값
bfd	bit 필드 값 (nbb バイト分)
crcH	체크섬(High Byte)

crcL

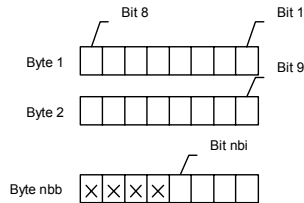
체크섬(low byte)

function 1 : N bit 읽어내기

네트워크 어드레스 addH/addL 에서 시작되는 nbi bit (boolean 형 값) 읽어내기

Question	Slv	01	AddH	addL	00	nbi	crcH	crcL
Answer	Slv	01	Nbb	bfd	...		crcH	crcL
			Byte 1		Byte nbb			

bfd 은 nbb 바이트 분(分)의 지역 bit 필드 번호로 이하의 포맷에서 나타냅니다.



Bit 1 은 네트워크 어드레스 addH/addL 에 따른 boolean 값

Bit nbi 은 네트워크 어드레스 addH/addL + nbi -1 에 따른 boolean 값

X는 미정의(未定義) 값

function 3 : N 워드 읽어내기

네트워크 어드레스 addH/addL 에서 시작되는 nbw 워드분 읽어내기

Question	slv	03	addH	addL	00	nbw	crcH	crcL
Answer	slv	03	nbb	vH	VL	...	crcH	crcL

nbb 은 vH, vL 바이트 수의 모든 합계 바이트 수입니다.

function 5 : 1bit 적어넣기

네트워크 어드레스 addH/addL 의 boolean 값 (bit) 적어넣기

Question	slv	05	addH	addL	vH	00	crcH	CrcL
----------	-----	----	------	------	----	----	------	------

Answer	slv	05	addH	AddL	vH	00	crcH	CrcL
--------	-----	----	------	------	----	----	------	------

function 6 : 1 워드 적어넣기

네트워크 어드레스 addH/addL 의 정수형 값 (워드) 적어넣기

Question	slv	06	addH	addL	vH	vL	crcH	CrcL
----------	-----	----	------	------	----	----	------	------

Answer	slv	06	addH	addL	vH	vL	crcH	crcL
--------	-----	----	------	------	----	----	------	------

function 1 6 : N 워드 적어넣기

네트워크 어드레스 addH/aadL 에서 시작되는 nbw 워드 적어 넣기 (nbw = 2nbw)

Question	Slv	10	addH	addL	00	nbw	nbb	vH	vL	...	crcH	crcL
----------	-----	----	------	------	----	-----	-----	----	----	-----	------	------

Answer	Slv	10	addH	addL	00	Nbw	crcH	crcL
--------	-----	----	------	------	----	-----	------	------

예 :

function 1 의 예 :

slave 번호 1 의 네트워크 어드레스 0x1020 에서 15bit 읽어 내기의 경우

Question	01	01	10	20	00	0F	79	04
----------	----	----	----	----	----	----	----	----

Answer	01	01	02	00	12	39	F1
--------	----	----	----	----	----	----	----

읽어 낸 값은 0x0012, 2 진수에서는 0000 0000 0001 0010 이 됩니다.

가장 우측의 bit 이 어드레스 0x1020 이 논리값 변수가 됩니다.

가장 좌측의 bit 이 어드레스 0x102F 가 논리값 변수가 됩니다.

이 예에서는 어드레스 0x1021 과 0x1024 가 TRUE 가 되고, 기타 어드레스는 FALSE 가 됩니다.

Function16 의 예:

slave 번호 1 의 네트워크 어드레스 0x2100 から 2 words 를 적어 넣는다. 적힌 값은 0x1234 와 0x5678 입니다.

Question	01	10	21	00	00	02	04	12	34	56	78	1C	CA
----------	----	----	----	----	----	----	----	----	----	----	----	----	----

Answer	01	10	21	00	00	02	4B	F4
--------	----	----	----	----	----	----	----	----

파일 전송

MODBUS protocol 은 기본적인 서비스만을 규정하고 있기 때문에 메이커 특성의 각종 function 코드에 따라 확장되고 있습니다.

ISaGRAF 에 따른 MODBUS 의 경우는 이하의 제한이 있습니다.

- ISaGRAF 변수만 액세스 가능합니다.
- 다량의 데이터를 고속으로 전송할 수 없습니다.

때문에, ISaGRAF 는 Modbus 라이크한(?) 파일 전송용 protocol (remote 파일 관리 protocol) 를 설명하고 있습니다. 이 방법으로는 이하의 내용이 가능합니다.

- Vinally, ASCII 파일의 다운로드
- Vinally, ASCII 파일의 업로드
- 가상 또는 물리 메모리를 매개로 해서 데이터 교환

이를 위해 ISaGRAF 의 통신 link 로 ISaGRAF 와는 다른 어플리케이션과 ISaGRAF 타겟과의 통신이 가능하게 됩니다.

이 protocol 은 이하의 concept 에 기초를 두고 있습니다.

- ISaGRAF 타겟측 파일은 **remote file**
- 마스터 컴퓨터측의 파일은 **local file**
- 파일 내의 바이트 데이터는 3 2 bit 의 **base address** 에 1 6 bit 의 **byte address** 를 사용해 액세스됩니다.

remote 파일명을 선택하거나, base address 를 선택하거나, remote 파일의 데이터를 읽어 내기와 적어 내기의 요구는 1 6 bit 의 byte address 를 사용해서 됩니다.

function 17: 데이터 적어 넣기

nbb 은 vH, vL 의 합계 바이트 수가 됩니다.

Question	Slv	11	AddH	AddL	00	nbb	nbb	vH	vL	...	crcH	crcL
Answer	Slv	11	addH	AddL	00	Nbb	crcH	CrcL				

리퀘스트의 내용은 addH/addL 어드레스 값으로 대신합니다.

– **0xF000: remote 파일명의 초기화**

nbb 는 vH, vL 파일에서 나타내는 파일명의 문자 수에 해당합니다. 문자열의 마지막에는 NULL (\0) 을 포함합니다.

만약 파일이 없는 경우 파일은 작성됩니다. (속성은 : Write + Read + + Execute)

– **0xF002: base 어드레스를 지정값으로 변경**

nbb 는 4 가 됩니다. 최초의 vH/vL 바이트는 지정값인 High 워드에 해당합니다. 뒤는 Low 워드에 해당합니다. 32bit 값을 사용할 수 있습니다.

base 어드레스가 변경된 후는 이 변경된 어드레스가 이후 읽어내기, 적어 넣기에 사용됩니다. 이 변경 리퀘스트가 없으면, 디폴트 어드레스는 0 입니다.

– **0xF004: 파일 삭제**

nbb 는 0 이 됩니다.

파일이 있어 삭제 가능하면, 삭제됩니다.

– **> 0xF004: 예약**

– **< 0xF000: 바이트 적어 넣기**

지정 바이트 데이터가 remote 파일에 주어진 순번 (좌에서 우로) 으로 이미 지정된 remote 파일에 적힙니다. 적어 넣은 어드레스는 지정된 어드레스(addH/addL)를 이전에 선택 종료의 base 어드레스에 덧붙혀진 것입니다. 어드레스가 파일 사이즈를 넘고 있는 경우는 파일 사이즈가 커집니다. 이 function 에서 파일 사이즈를 작게 할 수는 없습니다.

function 18: 데이터 읽어내기

Question	Slv	12	addH	AddL	00	Nbb	crcH	crcL
----------	-----	----	------	------	----	-----	------	------

Answer	Slv	12	nbb	V	V	...	crcH	crcL
--------	-----	----	-----	---	---	-----	------	------

데이터를 읽어내는 어드레스는 addH/addL 로 지정되어 있습니다. 이 값은 F000 보다 적어야만 합니다. 선택된 remote 파일명에서 선택종료 base 어드레스에 addH/addL 어드레스를 추가한 어드레스에서 순번에 지정된 바이트 수 (nbb) 를 읽어냅니다.

값은 V 파일에서 좌에서 우의 순번으로 읽어냅니다.

예 :

remote 파일명: 'target.fil'선택.

Question	01	11	F0	00	00	0B	0B	74	...	00	25	9F
----------	----	----	----	----	----	----	----	----	-----	----	----	----

Answer	01	11	F0	00	00	0B	8F	0E
--------	----	----	----	----	----	----	----	----

base 어드레스 0x10000 을 선택.

Question	01	11	F0	02	00	04	04	00	01	00	00	76	11
----------	----	----	----	----	----	----	----	----	----	----	----	----	----

Answer	01	11	F0	02	00	04	6E	CA
--------	----	----	----	----	----	----	----	----

4 바이트 적어 넣기: 개시절대 어드레스 0x107D0, 적어 넣기 값 01,02,03,04.

Question	01	11	07	D0	00	04	04	01	02	03	04	28	6F
----------	----	----	----	----	----	----	----	----	----	----	----	----	----

Answer	01	11	07	D0	00	04	FC	87
--------	----	----	----	----	----	----	----	----

4 바이트 읽어 넣기: 개시절대 어드레스 0x107D0.

Question	01	12	07	D0	00	04	B8	87
----------	----	----	----	----	----	----	----	----

Answer	01	12	04	01	02	03	04	58	7D
--------	----	----	----	----	----	----	----	----	----

C.9 전원 이상시의 관리

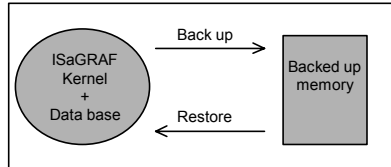
C.9.1 기본

전원 이상시의 관리는 이하 세가지의 관점에서 중요한 항목입니다.

- 프로세스 사양
- 하드웨어 능력
- 프로그램 수단

ISaGRAF 타겟에 의한 전원 이상시의 관리에 대해서 완전한 방법이 정해져 있는 것은 아닙니다. 각각의 어플리케이션 내에서 처리하려면, 적어도 하드웨어 상에서 대책을 세우는 것이 기본 컨셉트입니다.

전원 이상시에 시스템을 바르게 재시작할 수 있기 위해서는 세가지 문제를 해결해야만 합니다.



- 데이터의 백업
- 전원 이상이 발생한 것을 재 start 시에 검출
- 백업된 데이터의 복원

위의 두번째 문제점에 대한 해결방법은 표준 타겟 소프트웨어는 아닙니다. 시스템 개발자는 하드웨어 상황을 ISaGRAF 에서 엿세스할 수 있도록 C 언어에 의한 프로그램을 기술해서 ISaGRAF 에 편성하는 것을 고려해야 합니다.

중요한 포인트로서는 어느 타겟을 백업해야 하느냐를 정하는 일입니다. 이하는 두 종류의 타겟 카테고리를 나타냅니다.

- 프로그램 상태:

프로세스 변수, 전원 이상 일시, 어플리케이션 파라미터, 프로그램 변수, 카운트, 타임, 중간 데이터, 플러그 등.

- 프로그램 상태:

활성 스텝상태, 각 프로그램 상태 등.

이것들은 두개의 카테고리에서 이외에 생각할 수 있는 포인트를 이하에 나타냅니다.

C.9.2 Application 변수의 백업

유지 변수

workbench 에서 변수를 등록하고, 다시 내부 변수에 대해서는“ 유지(保持)” 속성을 설정할 수 있습니다.

“ 유지” 속성을 가진 내부변수는 타겟의 각 사이클 마지막에 지정된 메모리 area 에 copy 됩니다. 이 메모리 영역은 통상 배터리 백업되어 있는 RAM 영역이 됩니다.

ISaGRAF 에서 하나 이상의 변수가“ 유지” 속성을 갖고 있는 경우는 스타트업이 start 시에“ 유지” 변수 내용을 본니다.

- 직전에 동일 어플리케이션이 실행하고 있던 경우는 보관되고 있는 유지 변수내용이 각각 유지 변수에 액세스됩니다.
- 직전에 실행했던 어플리케이션이 동일 어플리케이션이 아닐 경우, 또는 어플리케이션이 실행하지 않았던 경우는 ISaGRAF 의 모든 유지 변수내용은 N U L L 이 됩니다.

각종 유지 변수가 보관되는 메모리 영역은 ISaGRAF workbench 로 정의됩니다. 프로그램 관리 창 「코드 생성」 메뉴인 「어플리케이션」 커맨드로 유지 변수 메모리 영역 설정이 가능합니다.

파라미터 에 사용되는 문자열의 의미는 이하와 같은 포맷으로 정의되고 있습니다.

boo_add, boo_size, ana_add, ana_size, tmr_add, tmr_size, msg_add, msg_size

boo_add: boolean 형 변수를 보관하기 위한 어드레스 (1 6 진수) 에서 0 이외.

boo_size: 지정된 어드레스에서 사용되는 바이트 수 (1 6 진수).
하나의 boolean 형 변수에 대해 1 바이트 점유

ana_add: 정수형 변수를 보관하기 위한 어드레스 (1 6 진수) 에서 0 이외.

ana_size:	지정된 어드레스에서 사용되는 바이트 수 (1 6 진수) . 하나의 정수형 변수에 대해 4 바이트 점유
tmr_add:	타임형 변수를 보관하기 위한 어드레스 (1 6 진수) 에서 0 이하.
tmr_size:	지정된 어드레스에서 사용되는 바이트 수 (1 6 진수) . 하나의 타임형 변수에 대해 5 바이트 점유
msg_add:	가변장 문자열형 변수를 보관하기 위한 어드레스 (1 6 진수) 에서 0 이외.
msg_size:	지정된 어드레스에서 사용되는 바이트 수 (1 6 진수) . 하나의 가변장 문자열형 변수에 대해 2 5 5 바이트 점유

요구사항:

- 모든 타입의 변수를 유지할 필요가 없는 경우에도 반드시 모든 타입의 필드를 지정할 필요가 있습니다. 유지하는 변수가 없는 경우는 사이즈를 0 으로 합니다. (단, 정수형 변수의 경우 사이즈는 4 가 최소값) 어드레스은 모두 0 이외가 됩니다.

예:

백업 (유지) 하는 변수를 이하와 같이 합니다.

20 boolean 형 변수

0 정수형 변수

0 타임형 변수

3 가변장 문자열형 변수

백업 메모리는 어드레스 0xA2F200 에 set 되어 있습니다.

관리 영역을 이하와 같이 설정합니다.

boolean 형 변수는 어드레스 0xA2F200, 사이즈는 2 0 바이트

정수형 변수는 어드레스 0xA2F214, 사이즈는 4 바이트.

타임형 변수를 위한 dummy 어드레스 0xA2F200 이고, 사이즈는 0 바이트

가변장 문자열형 변수는 어드레스 0x A2F218 이고, 사이즈 256*3 바이트

이 경우 workbench 측 (유지변수 어드레스 영역) 에 설정하는 값은 아래와 같습니다.

A2F200,14,A2F214,4,A2F200,0,A2F218,300

SYSTEM 함수 호출

만약, 어플리케이션 변수의 대부분이 보지변수가 되는 경우는 SYSTEM function 을 사용해서, 변수 set 을 정리해서 다루는 것도 가능합니다. 이 경우는 변수의 백업과 복원 어플리케이션 레벨에서 프로그래머가 기술할 필요가 있습니다.

우선, 지정된 또는 모든 타입의 변수에 대해서 이하와 같은 백업용 메모리 area 를 정의해야 합니다.

<new_address> := SYSTEM(SYS_INITxxx,<address>);

여기서,

– <address>는 백업메모리 어드레스 (16#에서 1 6 진수 포맷)를 지정. 이것은 반드시 짝수 어드레스이어야 합니다. 홀수 어드레스는 에러가 발생합니다.

– SYS_INITxxx 은 이하와 같습니다.

- * SYS_INITBOO boolean 형 변수에 대한 백업어드레스 지정
- * SYS_INITANA 정수형 변수에 대한 백업어드레스 지정
- * SYS_INITTMR 타임형 변수에 대한 백업어드레스 지정
- * SYS_INITALL boolean, 정수, 타임형의 모든 변수에 대한 백업어드레스 지정

– <new_address> 는 다음의 free 어드레스가 되돌아 옵니다. 즉, <address> + 백업된 변수의 사이즈 (바이트 수 = SYS_INITxxx 으로 지정). 필요한 메모리 백업용 사이즈를 확보합니다. 에러가 발생하면 <new_address>는 0 이 됩니다.

다음은 실제 변수를 백업합니다. 이 수속은 현재 타겟 사이클의 마지막에 합니다. trigger 되는 신호는 하드웨어에서 boolean 형 변수와 C 언어 function 에 집어낸 신호가 전원 이상을 검출한 것을 상정합니다. 이 경우 전원 이상을 검출한 후에 1 사이클 늦게 ISaGRAF 는 백업처리하게 됩니다.

<error> :=SYSTEM(SYS_SAVxxx,0);

여기서

– SYS_SAVxxx 는 이하와 같습니다.

- * SYS_SAVBOO boolean 형 변수의 백업조작
- * SYS_SAVANA 정수형 변수의 백업조작
- * SYS_SAVTMR 타임형 변수의 백업조작.
- * SYS_SAVALL boolean, 정수, 타임의 모든 형 변수의 백업조작.

– <error> 에러 발생시 (예를 들면, 이전에 SYS_INITXXX 가 호출되지 않는 경우) 는 0 이외의 값이 되돌아옵니다.

마지막에 백업된 변수를 복원합니다. 사이클 마지막에 한번만 됩니다. **<error> :=
SYSTEM(SYS_RESTxxx,0);**

여기서

– SYS_RESTxxx 는 이하와 같습니다.

* SYS_RESTBOO boolean 형 변수의 복원

* SYS_RESTANA 정수형 변수의 복원.

* SYS_RESTTMR 타임형 변수의 복원.

* SYS_RESTALL boolean, 정수, 타임의 모든 형 변수의 복원

– <error> 에러 발생시 (예를 들면, 이전에 SYS_INITXXX 가 호출되지 않는 경우) 는 0 이외의 값이 되돌아옵니다.

이하에 변수의 백업을 위해 SYSTEM function 커맨드를 정리합니다.

커맨드 (예약 키워드)	값	의미
SYS_INITBOO	16#20	boolean 형 변수의 백업초기화
SYS_SAVBOO	16#21	boolean 형 변수 보관
SYS_RESTBOO	16#22	boolean 형 변수의 복원
SYS_INITANA	16#24	정수형 변수의백업초기화
SYS_SAVANA	16#25	정수형 변수의 보관
SYS_RESTANA	16#26	정수형 변수의 복원
SYS_INITTMR	16#28	타임형 변수의 백업 초기화
SYS_SAVTMR	16#29	타임형 변수의 보관
SYS_RESTTMR	16#2A	타임형 변수의 복원
SYS_INITALL	16#2C	모든 변수의 백업初期化
SYS_SAVALL	16#2D	모든 변수의 보관
SYS_RESTALL	16#2E	모든 변수의 복원

커맨드 (예약 키 워드)	인수	되돌리기 값
SYS_INITxxx	메모리 어드레스	다음의 free 어드레스
SYS_SAVxxx	0	정상시는 0
SYS_RESTxxx	0	정상시는 0

Customized

C언어 function , function 블록을 사용해서 бат대리 백업 메모리 영역의 내용에 액세스 가능합니다. 따라서, 어플리케이션에서 변수로 지정된 메모리 영역에 대한 변수를 읽어내기, 적어 넣기 합니다.

예:

1) 어플리케이션 고유의 소속 :

백업, restore_temp, restore_date, restore_cpt 는 사용자 C언어 function 입니다.

백업(temperature, date, cnt); store 3 critical data

temperature := **restore_temp**(); restore temperature

date := **restore_date**(); restore date

cnt := **restore_cnt**(); restore counter

2) 일반적인 수속 :

백업_init, 백업, 백업_link, restore 는 사용자 C언어 function

save_id := **백업_init**(address, size); 메모리 백업 배열 배정

백업(save_id, cpt1, 3); cpt1 을 3 번째 요소로 보관.

rest_id := **백업_link**(address, size) 백업 메모리를 link

cpt1 := **restore**(rest_id, 3); 백업 값 cpt1 의 복원.

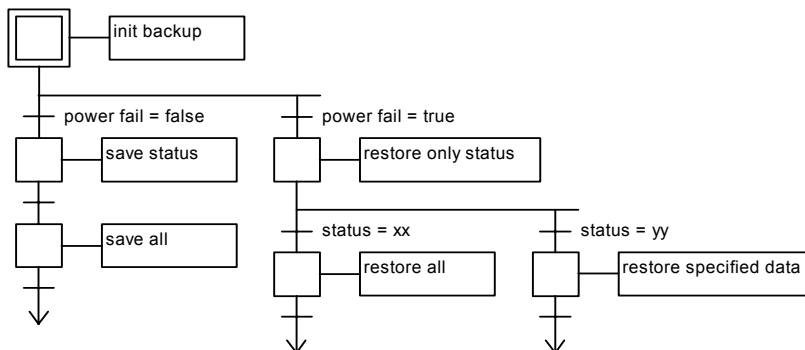
C.9.3 프로그램 백업

모든 프로그램의 모든 상태는 항상 백업 가능하지만, 이하 세가지 이유에서는 명령 할 수 없습니다.

- 어떤 프로세스는 재 start 전에 특정 수속을 할 필요가 있기 때문에.
- 모든 어플리케이션의 모든 상태에 대한 취급은 그다지 의미가 없기 때문에.
- 어떤 외부의 리소스, 즉, C 프로그램 등은 자동 재 start 가 불가능한 때문에.

통상은 재 start 에 필요하다고 생각되는 프로세스 상태를 정수형, boolean 형 변수에 백업해 둡니다. 그래서 완전하지는 않지만, 가장 낮은 한도의 프로세스 이미지에서 재 start 와 어플리케이션의 초기화 등이 가능합니다. 단, 완전 자동의 재 start 를 추구하는 것은 무리일지도 모릅니다.

예제:



C.10 부록: 에러 리스트와 설명

에러 리스트:

코드	메시지	타입
1	런타임 DB 용 메모리를 allocate 할 수 없습니다.	system
2	다른 어플리케이션 데이터 베이스가 지정되었습니다.(Motorola/Intel)	application
3	통신 메일박스를 allocate 할 수 없습니다.	system
4	Kernel 데이터 베이스를 link 할 수 없습니다.	system
5	Kernel 로 질문을 던지고 있는 시간이 time over 입니다.	system
6	Kernel 에서 답변을 기다리는 시간이 time over 입니다.	system
7	통신을 초기화할 수 없습니다.	system
8	유지 변수용 메모리를 allocate 할 수 없습니다.	application
9	어플리케이션이 정지했습니다.	application
10	과다한 N 또는 P 액션이 동시에 발생하고 있습니다.	application
11	과다한 set 액션이 동시에 발생하고 있습니다.	application
12	과다한 reset 액션이 동시에 발생하고 있습니다.	Application
13	미지의 TIC 명령입니다.	Application
16	데이터 읽어내기 request 에 응답할 수 없습니다.	system
17	데이터 적어 넣기 request 에 응답할 수 없습니다.	system
18	디버거섹션 request 에 응답할 수 없습니다.	system
19	MODBUS request 에 응답할 수 없습니다.	system
20	디버거어플리케이션 request 에 응답할 수 없습니다.	system
21	디버거에 응답할 수 없습니다.	system
22	Stack over Flow 입니다.	system
23	미지의 request 코드입니다.	system
24	isanet 통신 에러입니다.	system
25	통신의 같은 기간 에러입니다.	system
28	어플리케이션용에 메모리를 allocate 할 수 없습니다.	Application

29	어플리케이션 update 에 메모리를 allocate 할 수 없습니다.	application
30	미지의 OEM 키 코드입니다.	application
31	boolean 형 입력보드를 초기화할 수 없습니다.	application
32	정수형 입력보드를 초기화할 수 없습니다.	application
33	가변장 문자열형 입력보드를 초기화할 수 없습니다.	application
34	boolean 형 출력보드를 초기화할 수 없습니다.	application
35	정수형 출력보드를 초기화할 수 없습니다.	application
36	가변장 문자열형 출력보드를 초기화할 수 없습니다.	application
37	boolean 형 보드에서의 입력은 불가능합니다.	application
38	정수형 보드에서의 입력은 불가능합니다.	application
39	가변장 문자열형 보드에서의 입력은 불가능합니다.	application
40	Boolean 형 출력 변수의 출력은 불가능합니다.	application
41	정수형 출력 변수의 출력은 불가능합니다.	application
42	가변장 문자열형 출력 변수의 출력은 불가능합니다.	application
43	Boolean 형 변수를 조작할 수 없습니다.	application
44	정수형 변수를 조작할 수 없습니다.	application
45	가변장 문자열형 변수를 조작할 수 없습니다.	application
46	보드를 오픈 할 수 없습니다.	application
47	보드를 클로즈 할 수 없습니다.	application
50	Boolean 형 출력 변수의 표시는 불가능합니다.	program
51	정수형 출력 변수의 표시는 불가능합니다.	program
52	가변장 문자열형 출력 변수의 표시는 불가능합니다.	program
53	SFC 프로그램이 이미 시작되고 있는지, 동결되어 있습니다.	program
54	SFC 프로그램이 이미 동결되어 있을 수 있지만, start 는 불가능합니다.	program
55	SFC 프로그램이 이미 start 해 있을 수 있지만, 동결되어 있지 않습니다.	program
56	SFC 프로그램이 이미 정지시킬 수 있지만, start 해 있지는 않습니다.	program
61	미지의 시스템 request 코드	program
62	샘플링 주기가 over Flow 했습니다.	program
63	사용자 function 가 준비되어 있지 않습니다.	application

64	정수가 0 으로 되었습니다.	program
65	수치 변환 function 가 준비되어 있지 않습니다.	application
66	어플리케이션 블록이 준비되어 있지 않습니다.	application
67	준비 function 가 준비되어 있지 않습니다.	application
68	실수가 0 으로 되었습니다.	application
69	부정한 operation 파라미터 입니다.	application
70	어플리케이션 파라미터 는 수정되지 않습니다.	application
71	어플리케이션 I/O 과 라이브러리는 수정되지 않습니다.	application
72	어플리케이션 심벌은 수정되지 않습니다.	application
73	갱신할 수 없습니다.: 다른 boolean 형 변수 set 입니다.	application
74	갱신할 수 없습니다.: 다른 정수형 변수 set 입니다.	application
75	갱신할 수 없습니다.: 다른 타임형 변수 set 입니다.	application
76	갱신할 수 없습니다.: 다른 가변장 문자열형 변수 set 입니다.	application
77	갱신할 수 없습니다.: 새로운 어플리케이션이 발견되지 않습니다.	application
> 100	OEM 에러 코드	
	이하의 OEM 키의 에러 메시지입니다. 상세한 것은 I/O 보드의 기술메모 또는 공급원에 문의해 주십시오.	

에러에는 다음의 3 가지 타입이 있습니다.

- System 에러:

Target 소프트웨어 또는 하드웨어에 기인하는 에러입니다.

Target (P L C) 하드 reset 또는 다른 어플리케이션을 R U N시켜 같은 현상이 나타나는가를 확인합니다.

이 에러들이 발생한 경우는 ISaGRAF 지원으로 연락 바랍니다.

- Application 에러:

어플리케이션 파라미터 와 사이즈 및 내용에 기인하는 에러입니다.

이 경우는 다른 어플리케이션을 R U N 시켜서 에러가 없어지는 것을 확인해 주십시오. 만약, 계속해서 에러가 날 경우는 system 에러의 가능성도 있습니다.

- 프로그램에러:

프로그램 sequence 에 기인하는 에러입니다.

이 에러는 어플리케이션을 사이클 모드로 실행하거나, 어떤 특성의 프로그램을 정지시켜 꺼지게 하는 경우도 있습니다.

에러 설명:

1	런타임 데이터 베이스용에 메모리를 allocate 할 수 없습니다.	System
---	---------------------------------------	--------

메모리가 부족합니다. 하드웨어를 체크해 주십시오.

2	다른 어플리케이션 데이터 베이스가 지정되었습니다.(Motorola/Intel)	application
---	---	-------------

어플리케이션 코드 파일이 정상이 아닙니다. 예를 들면, INTEL 플랫폼에 MOTOROLA 코드가 다운로드된 때와 파일이 변경된 경우에 발생합니다.

3	통신 메일박스를 allocate 할 수 없습니다.	system
---	-----------------------------	--------

내부 태스크 통신용 isax4 의 메모리 스페이스를 확보할 수 없었던 경우에 발생합니다. 통신 태스크에 에러가 발생합니다.

4	Kernel 데이터 베이스를 link 할 수 없습니다.	system
---	--------------------------------	--------

지정된 slave 번호에서 Kernel 태스크가 없었던 경우에 발생합니다. 통신 태스크에 에러가 발생합니다.

5	Kernel 로 질문을 던지고 있는 시간이 time over 입니다.	system
---	--	--------

통신 태스크가 Kernel 에 대해 request 를 보낼 수 없습니다. Kernel 태스크가 실행되지 않을 가능성이 있습니다.

6	Kernel 에서 답변을 기다리는 시간이 time over 입니다.	system
---	---------------------------------------	--------

통신 태스크가 Kernel 에서 응답을 수신할 수 없습니다. Kernel 태스크가 실행되지 않을 가능성이 있습니다.

7	통신을 초기화할 수 없습니다.	system
---	------------------	--------

통신 layer 가 초기화되지 않았을 때 발생합니다. 또 통신 pass 가 지정되어 있지 않을 경우에도 발생합니다. 이 에러로 Kernel 태스크 실행을 정지하는 일은 없지만, 통신은 불가능합니다.

8	유지 변수용 메모리를 allocate 할 수 없습니다.	Application
---	--------------------------------	-------------

유지변수를 관리할 수 없습니다. 두 가지로 추측할 수 있습니다.

- 파라미터 로서 전달된 문자열이 문법적으로 바르지 않을 경우.
- 각 블록에서 지정된 메모리 사이즈가 충분치 않을 때.

우선, 유지변수 파라미터 문법을 확인해서 유지변수의 수를 줄이는 것을 생각해 보십시오.

9	어플리케이션이 정지했습니다.	Application
---	-----------------	-------------

어플리케이션이 디버거에서 정지된 경우에 경고 메시지가 나옵니다.

10	과다한 N 또는 P 액션이 동시에 발생하고 있습니다.	application
----	-------------------------------	-------------

1 회의 PLC 사이클 중에 (N), (P) 액션이 지나치게 많을 경우에 발생합니다. 문제 발생의 장소가 발견되는 경우에는 Cycle to Cycle 모드로 바꿀 수 있습니다. 동시에 가능한 액션 수는 (2 + S F C 프로그램 수 x 4) 입니다.

11	과다한 set 액션이 동시에 발생하고 있습니다.	application
----	----------------------------	-------------

1 회의 PLC 사이클 중에 set (S) 액션 <스텝이 액티브 된 때 실행된다> 이 지나치게 많을 경우에 발생합니다. 위의 No.10 을 참고해 주십시오.

12	과다한 reset 액션이 동시에 발생하고 있습니다.	application
----	------------------------------	-------------

1 회의 PLC 사이클 중에 reset (R) 액션 <스텝이 비액티브 된 때 실행된다> 이 지나치게 많을 경우에 발생합니다. 위의 No.10 을 참고해 주십시오.

13	미지의 TIC 명령입니다.	application
----	----------------	-------------

어플리케이션 코드 (T I C 코드) 에 에러가 검출되었습니다. 두 가지로 추측할 수 있습니다.

- 외부 프로그램이 어플리케이션 코드에 적어넣기 하고 있는 경우를 생각할 수 있습니다. 사이클 모드로 바꾸어 에러가 발생하는 장소를 특정해 주십시오. I / O 드라이브 파라미터가 틀리지 않은 지 확인해 주십시오.
- Target 실행 소프트웨어에는 한정된 instruction 만이 포함되어 있습니다. 어플리케이션이 포함되지 않은 instruction 을 지정했든지, 다른 변수 타입을 지정한 것을 생각할 수 있습니다.

16	데이터 읽어 내기 request 에 응답할 수 없습니다.	System
----	---------------------------------	--------

ISaGRAF 의 MODBUS request 의 function18 (파일 읽어내기) 에 따라 에러가 발생했습니다.
통신접속 및 Target 과 마스터의 설정을 확인해 주십시오.

17	데이터 적어 넣기 request 에 응답할 수 없습니다.	System
----	---------------------------------	--------

ISaGRAF 의 MODBUS request 의 function17 (파일 적어넣기) 에 따라 에러가 발생했습니다.
통신접속 및 Target 과 마스터의 설정을 확인해 주십시오.

18	디버거섹션 request 에 응답할 수 없습니다.	System
----	-----------------------------	--------

디버거 request 에 대해 응답시에 에러가 발생했습니다.
통신 접속 및 Target 과 마스터의 설정을 확인해 주십시오.

19	MODBUS request 에 응답할 수 없습니다.	System
----	------------------------------	--------

MODBUS request 에 대해 응답시에 에러가 발생했습니다.
통신 접속 및 Target 과 마스터의 설정을 확인해 주십시오.

20	디버거 어플리케이션 request 에 응답할 수 없습니다.	System
----	----------------------------------	--------

디버거 request 에 대해 응답시에 에러가 발생했습니다.
통신 접속 및 Target 과 마스터의 설정을 확인해 주십시오.

21	디버거에 응답할 수 없습니다.	System
----	------------------	--------

디버거 request 에 대해 응답시에 에러가 발생했습니다.
통신 접속 및 Target 과 마스터의 설정을 확인해 주십시오.

22	Stack over Flow 입니다.	System
----	----------------------	--------

프로그램 Stack over Flow 입니다.

23	미지의 request 코드입니다.	System
----	--------------------	--------

디버거에서 request 가 미지의 내용입니다.

24	isanet 통신 에러입니다.	System
----	------------------	--------

이 에러는 디버거가 달릴 때마다 표시됩니다. 이 경우 시스템은 정상입니다. 그렇지 않으면 isanet 통신 에러가 발생합니다. 접속을 확인하고, 시스템설정 내용을 Target 과 workbench 측에서 체크합니다.

만약, 두번째 에러 필드가 표시되어 있으면, 이하와 같은 상황을 알게 됩니다.

- 1: 송신 및 수신중의 에러
- 2: 소켓을 생성중인 에러
- 3: 소켓의 binding 또는 listening 중인 에러
- 4: New Client 를 접수중인 에러

25	통신의 같은 기간 에러입니다.	System
----	------------------	--------

Target 과 workbench 측에서 통신의 같은 기간 에러입니다. Target 과 workbench 통신 접속 상태와 시스템 설정파라미터 등을 확인해 주십시오.

28	어플리케이션용에 메모리를 allocate 할 수 없습니다.	application
----	----------------------------------	-------------

메모리 부족입니다. 어플리케이션 사이즈에 대한 하드웨어 메모리 사이즈를 확인해 주십시오.

29	어플리케이션 update 에 메모리를 allocate 할 수 없습니다.	application
----	---	-------------

메모리 부족입니다. 어플리케이션 사이즈에 대한 하드웨어 메모리 사이즈를 확인해 주십시오.

30	미지의 OEM 키 코드입니다.	application
----	------------------	-------------

어플리케이션이 Target 소프트웨어에서 확인할 수 없는 OEM코드의 I/O 드라이브를 사용하고 있습니다. workbench I/O 접속 에디터로 보드를“ virtual” 에 설정해서 문제의 I/O 보드를 검출해 주십시오. workbench 라이브러리가 Target 버전과 맞지 않을 가능성이 있습니다.

31	boolean 형 입력보드를 초기화할 수 없습니다.	application
----	------------------------------	-------------

논리값 입력보드의 초기화에 실패했습니다. workbench I/O 접속 에디터로 논리값 입력보드 파라미터 설정을 확인해 주십시오.

32	정수형 입력보드를 초기화할 수 없습니다.	application
----	------------------------	-------------

정수형 입력보드를 초기화에 실패했습니다. workbench I/O 접속 에디터로 정수형 입력보드 파라미터 설정을 확인해 주십시오.

33	가변장 문자열형 입력보드를 초기화할 수 없습니다.	application
----	-----------------------------	-------------

가변장 문자열형 입력보드를 초기화에 실패했습니다. workbench I/O 접속 에디터로 가변장 문자열형 입력보드 파라미터 설정을 확인해 주십시오.

34	boolean 형 출력보드를 초기화할 수 없습니다.	application
----	------------------------------	-------------

논리값 출력보드의 초기화에 실패했습니다. workbench I/O 접속 에디터로 논리값 출력보드 파라미터 설정을 확인해 주십시오.

35	정수형 출력보드를 초기화할 수 없습니다.	application
----	------------------------	-------------

정수형 출력보드의 초기화에 실패했습니다. workbench I/O 접속 에디터로 정수형 출력보드 파라미터 설정을 확인해 주십시오.

36	가변장 문자열형 출력보드를 초기화할 수 없습니다.	Application
----	-----------------------------	-------------

가변장 문자열형 출력보드에 실패했습니다. workbench I/O 접속 에디터로 가변장 문자열형 출력보드 파라미터 설정을 확인해 주십시오.

37	Boolean 형 보드에서의 입력은 불가능합니다.	application
----	-----------------------------	-------------

논리값 입력보드 refresh 중에 에러가 발생했습니다. workbench I/O 접속 에디터로 보드의 파라미터 설정을 확인해 주십시오.

38	정수형 보드에서의 입력은 불가능합니다.	application
----	-----------------------	-------------

정수형 입력보드 refresh 중에 에러가 발생했습니다. workbench I/O 접속 에디터로 보드의 파라미터 설정을 확인해 주십시오.

39	가변장 문자열형 보드에서의 입력은 불가능합니다.	application
----	----------------------------	-------------

가변장 문자열형 입력보드 refresh 중에 에러가 발생했습니다. workbench I/O 접속 에디터로 보드의 파라미터 설정을 확인해 주십시오.

40	boolean 형 출력변수의 출력은 불가능합니다.	application
----	-----------------------------	-------------

논리값 출력보드 update 중에 에러가 발생했습니다. workbench I/O 접속 에디터로 보드의 파라미터 설정을 확인해 주십시오.

41	정수형 출력변수의 출력은 불가능합니다.	application
----	-----------------------	-------------

정수형 출력보드 update 중에 에러가 발생했습니다. workbench I/O 접속 에디터로 보드의 파라미터 설정을 확인해 주십시오.

42	가변장 문자열형 출력변수의 출력은 불가능합니다.	application
----	----------------------------	-------------

가변장 문자열형 출력보드 update 중에 에러가 발생했습니다. workbench I/O 접속 에디터로 보드의 파라미터 설정을 확인해 주십시오.

43	boolean 형 변수를 조작할 수 없습니다.	application
----	---------------------------	-------------

논리값 변수에 대해서 **OPERATE** call 중에 에러가 발생했습니다. **OPERATE** 파라미터 및 보드의 기술메모를 참조 바랍니다.

44	정수형 변수를 조작할 수 없습니다.	application
----	---------------------	-------------

정수형 변수에 대해서 **OPERATE** call 중에 에러가 발생했습니다. **OPERATE** 파라미터 및 보드의 기술메모를 참조 바랍니다.

45	가변장 문자열형 변수를 조작할 수 없습니다.	application
----	--------------------------	-------------

가변장 문자열형 변수에 대해서 **OPERATE** call 중에 에러가 발생했습니다. **OPERATE** 파라미터 및 보드의 기술메모를 참조 바랍니다.

46	보드를 오픈 할 수 없습니다.	application
----	------------------	-------------

어플리케이션이 Target 을 확인할 수 없는 I/O 보드를 사용하고 있습니다. Workbench I/O 접속 에디터를 체크 바랍니다. Workbench 라이브러리가 Target 버전과 맞지 않을 가능성이 있습니다.

47	보드를 클로즈 할 수 없습니다.	application
----	-------------------	-------------

어플리케이션이 Target 을 확인할 수 없는 I/O 보드를 사용하고 있습니다. Workbench I/O 접속 에디터를 체크 바랍니다. Workbench 라이브러리가 Target 버전과 맞지 않을 가능성이 있습니다.

50	Boolean 형 출력변수의 표시는 불가능합니다.	program
----	-----------------------------	---------

두개의 S F C sequence 가 동일한 논리값 출력변수로서 같은 사이클 내에 적어 넣으려고 합니다. 이러한 경우는 프로그램 계층에서 상위에 있는 것이 우선도가 높게 됩니다. 두개의 S F C 프로그램이 동일 레벨이면, 결과는 예측할 수 없습니다.

51	정수형 출력변수의 표시는 불가능합니다.	program
----	-----------------------	---------

두개의 SFC sequence 가 동일한 정수형 출력변수로서 같은 사이클 내에 적어 넣으려고 합니다. 이러한 경우는 프로그램 계층에서 상위에 있는 것이 우선도가 높게 됩니다. 두개의 SFC 프로그램이 동일 레벨이면, 결과는 예측할 수 없습니다.

52	가변장 문자열형 출력변수의 표시는 불가능합니다.	program
----	----------------------------	---------

두개의 SFC sequence 가 동일한 가변장 문자열형 출력변수로서 같은 사이클 내에 적어 넣으려고 합니다. 이러한 경우는 프로그램 계층에서 상위에 있는 것이 우선도가 높게 됩니다. 두개의 SFC 프로그램이 동일 레벨이면, 결과는 예측할 수 없습니다.

61	미지의 시스템 request 코드	Program
----	--------------------	---------

프로그램이 부정의 코드를 인수로 한 **SYSTEM** call 을 사용하고 있습니다.

62	샘플링 주기가 over Flow 했습니다.	Program
----	-------------------------	---------

PLC 사이클 타임이 workbench 에서 지정한 사이클 타임을 넘었습니다. 멀티태스크 시스템에서 이것은 Kernel 에 충분한 CPU 시간이 주어지지 않았다는 것을 의미합니다. 싱글 태스크 시스템에서는 PLC 사이클에 포함되는 operation 이 과다한 것을 의미합니다.

본 메시지를 소거하기 위해 아래와 같은 대책을 생각할 수 있습니다.

- 에러가 검출되는 장소에서 operation 을 줄입니다. 예를 들면, 액티브한 token 의 수, TRUE 가 되는 transition 의 수를 줄이거나, 복잡한 처리를 최적화하기도 합니다.
- Kernel 태스크 이외의 태스크 처리시간을 Kernel 태스크 쪽으로 줍니다.
- 통신 태스크의 부하를 줄여서 Kernel 태스크에 시간을 줍니다.
- 다이나믹하게 사이클 타임을 변경해서, 다른 프로세스 Stage 와 맞는 사이클 타임이 되도록 설정합니다.
- 사이클 타임을 0 으로 설정해서 Over-프로젝트를 무시합니다.

63	사용자 function 가 준비되어 있지 않습니다.	Application
----	------------------------------	-------------

어플리케이션이 Target 의 미지의 C 언어 function 를 사용하고 있습니다. workbench 라이브러리버전이 Target 과 맞지 않을 가능성이 있습니다.

64	정수가 0 으로 되었습니다.	program
----	-----------------	---------

어플리케이션에서 정수형을 0 으로 제산하려고 했습니다. 이 경우 정수형 최대값이 결과로 바뀌어 놓입니다.

65	수치 변환 function 가 준비되어 있지 않습니다.	application
----	--------------------------------	-------------

어플리케이션이 Target 의 미지의 C 언어 변환 function 를 사용하고 있습니다. workbench 라이브러리버전이 Target 과 맞지 않을 가능성이 있습니다. 이 경우 변환되지않습니다.

66	어플리케이션 블록이 준비되어 있지 않습니다.	Application
----	--------------------------	-------------

어플리케이션이 Target 의 미지의 function 블록을 사용하고 있습니다. workbench 라이브러리버전이 Target 과 맞지 않을 가능성이 있습니다.

67	표준 function 가 준비되어 있지 않습니다.	application
----	-----------------------------	-------------

어플리케이션이 Target 의 미지의 function 블록을 사용하고 있습니다. 단, 이 function 블록은 표준 F B 입니다. 문의 바랍니다.

72	어플리케이션 심벌은 수정되지 않습니다.	application
----	-----------------------	-------------

어플리케이션을 변경하려고 했습니다만, 심벌 테이블이 다르기 때문에 새로운 어플리케이션으로 변경할 수 없었습니다. 변수, function 블록 instance 등을 추가, 삭제되어 있을 가능성이 있습니다.

73	갱신할 수 없습니다.: 다른 boolean 형 변수 set 입니다.	application
----	---------------------------------------	-------------

어플리케이션을 변경하려고 했습니다만, 불가능했습니다. 논리값 변수가 추가, 삭제되어 있을 가능성이 있습니다.

74	갱신할 수 없습니다.: 다른 정수형 변수 set 입니다.	application
----	---------------------------------	-------------

어플리케이션을 변경하려고 했습니다만, 불가능했습니다. 정수형 변수가 추가, 삭제되어 있을 가능성이 있습니다.

75	갱신할 수 없습니다.: 다른 타임형 변수 set 입니다.	application
----	---------------------------------	-------------

어플리케이션을 변경하려고 했습니다만, 불가능했습니다. 타임형 변수가 추가, 삭제되어 있을 가능성이 있습니다.

76	갱신할 수 없습니다.: 다른 가변장 문자열형 변수 set 입니다.	application
----	--------------------------------------	-------------

어플리케이션을 변경하려고 했습니다만, 불가능했습니다. 가변장 문자열형 변수가 추가, 삭제되어 있을 가능성이 있습니다.

77	갱신할 수 없습니다.: 새로운 어플리케이션이 발견되지 않았습니	application
----	------------------------------------	-------------

다. 다운로

D.용어집

D. 용어설명

Action	SFC 프로그램 스텝이 Active 때 실행되는 스테이트먼트 리스트와 assignments 리스트
Action (FC) FC Action	Flow 차트의 심벌. Flow 차트 실행시 명령이 기술되는 오브젝트.
Activity of a step 스텝 활성상태	SFC token 으로 마크된 스텝 상태. 스텝에 할당되어 있는 Action 는 그 상태에 기초해서 실행됩니다.
Alias	변수 커맨드의 일부 (선두 1 6 바이트) 로 ladder 에디터에서 변수 커맨드의 표시로 사용됩니다.
Analog	변수의 타입. 정수형과 실수형이 있습니다.
Attribute 속성	변수의 속성으로 내부, 입력, 출력,정수가 있습니다.
Begin section	사이클릭 프로그램 섹션으로 Target 사이클 최초로 실행되는 섹션
Beginning step 개시 스텝	SFC (또는 매크로 스텝) 최초의 스텝으로 직접 transition 을 가지지 않습니다.
Boolean Boolean 형 값	변수 타입. TRUE 와 F A L S E 상태를 가집니다.
Boolean action	SFC action. 스텝 액티브 상태에 따라 Boolean 형 변수가 할당됩니다.
Breakpoint	debug 시에 사용자가 SFC 스텝과 transition 에 설정할 수 있는 일시 정지 버튼. Target 시스템은 SFC token 이 설정된 포이트에 이동하면 정지합니다.
C function C 언어 function	C 언어로 기술된 function.ISaGRAF 프로그램에서 호출됩니다. 표준 C 언어 function.이외, 사용자가 독자적으로 개발하는 것도 가능합니다.

C language C 언어	고급언어의 하나. C 언어 function , C 언어 F B 를 개발해서 ISaGRAF 와 link 하는 것도 가능합니다.
C source code	C 언어 프로그램 소스 코드 텍스트 파일
C source header	C 언어 프로그램의 정의와 타입을 기술한 헤더파일 (* . H 파일) .
Cell	SFC, F B D, L D 그래픽컬 에디터 상에서 심벌 배치에 따른 최소단위 area.
Child SFC program	FatherSFC 프로그램에서 컨트롤되는 ChildSFC 프로그램.
Clearing a transition transition 의 통과	프로그램 실행시에 SFC 직전 스텝의 token 이 제거되어 직후 스텝에 token 이 set 되는 것.
Code Generation 코드 생성	Target 시스템으로 실행되는 중간 코드 (T I C 코드) 를 생성하는 것.
Coil	L D 프로그램으로 출력변수를 할당하는 심벌.
Comment	프로그램 중에 포함할 수 있는 텍스트. 프로그램 실행에 영향을 주지 않습니다.
Comment (SFC)	SFC 스텝과 transition 에 부속하는 텍스트. 프로그램 실행에 영향을 주지 않습니다.
Common 공통	정의 워드 범위의 하나. 프로젝트 내의 어떤 프로그램에서도 다릅니다.
Condition (for a transition) Transition 조건	SFC transition 에 배정된 논리값 표현. 이 조건이 F A L S E 일때 transition 는 통과할 수 없습니다.
Connector (FC) FC Connector	Flow 차트의 function 과 테스트로의 점프를 의미하는 그래픽 심벌. 점프전에 references 번호가 붙은 작은 원으로 표시됩니다.
Constant expression 정의 표현	정의 표현.

Contact 접점	L D 프로그램에서 입력변수를 나타내는 심벌.
Conversion 수치 변환	아나로그 입력과 출력변수에 부칠 수 있는 필터. 입력과 출력이 갱신될 때 자동적으로 사용됩니다.
Conversion function 변환 function	C 언어로 기술된 수치변환 function . 아나로그 입출력을 부칠 수 있습니다.
Conversion table 수치변환 테이블	아나로그 입출력변수에 부칠 수 있는 선형의 변환을 표현하는 포인트 (X, Y) 집합.
Cross references	ISaGRAF workbench 에 따라 계산되는 모음에 선언되어 있는 변수 집계 리포터
Current result (IL) 현재결과(IL)	I L 프로그램에서의 instruction 계산결과 (현재 결과) . 현재 결과는 다음의 instruction 으로 변경됩니다. 또한 변수로 할당되어서도 사용됩니다.
Cycle timing	Target 에서의 프로그램 실행 사이클 (주기)
Cycle to cycle mode 사이클 모드	프로그램 실행 모드의 하나. 이 모드에서는 디버거에서의 지시로 1 사이클마다 실행이 가능합니다.
Cyclic	프로그램 속성의 하나. 매회 실행됩니다.
Decision (FC)	(테스트라 함) Flow 차트조건판단을 의미하는 심벌. Flow는 TES/NO 의 출력에서 접속됩니다.
Defined word 정의 워드	프로그램에서 사용되는 표현을 바껴 놓는 식별자.
Delayed operation (IL) 지연조작(IL)	I L 프로그램 operation 의 하나. ")가 프로그램 뒤에 나올 때까지 지연시키는 조작을 한다.
Diary 수정이력	프로그램 수정시에 남은 수정이력 텍스트 파일.

Dictionary	프로젝트에서 각종 타입의 변수와 정의 워드를 등록해서 정리.
모음	
Directly Represented Variable	I/O 변수를 모음에 선언하지 않고, I/O 보드의 I/O 채널을 직접 %표현으로 지정하는 것.
직접표현 변수	예 : %lxs.c (s:슬롯 번호, c:채널 번호)
Edge	boolean 형 변수의 변화 상태. 오르기는 FALSE 상태에서 TRUE 상태로 되는 것. 내리기는 TRUE 상태에서 FALSE 상태로 되는 것을 나타냅니다.
End section	프로그램 섹션의 하나. Target 사이클 마지막에 실행되는 사이클릭 섹션
Ending step	SFC (또는 매크로 스텝) 의 마지막스텝. 직후의 transition 은 종료 스텝
종료 스텝	
Expression	값을 평가하기 위한 표현.
표현	
Father SFC program	기타 SFC 프로그램 (child SFC 프로그램) 을 컨트롤할 수 있는 SFC 프로그램.
FBD	function 블록 다이어그램 언어
FC	Flow 차트.
Flow Chart	처리 Flow 를 그래픽으로 표현한 프로그램 언어. 프로그램의 내용을 기술하는 스텝과 조건을 기술해 처리 Flow 를 결정하는 테스트, 그리고 그것들을 연결하는 Flow 에서 구성됩니다.
Function block	FBD 언어로 그래픽 Component.
Functional Block Diagram	기본 functionBlock 이 접속된 다이어그램
Global	변수와 정의 워드의 범위. 프로젝트내의 모든 프로그램으로 다름.

Hierarchy 계층	프로젝트를 구성하고 있는 프로그램의 계층. 계층은 프로그램간의 친자관계를 나타내고 있습니다.
I/O board	방향성 (입력 또는 출력) 과 타입을 가진 보드. I/O 보드의 파라미터 는 ISaGRAF 라이브러리에 기록되고 있습니다.
I/O channel	I/O 보드의 접속 포인트. 각 I/O 채널은 하나의 I/O 변수를 가질 수 있습니다.
I/O connection I/O 접속	프로그램의 변수와 Target 시스템상의 I/O 보드의 채널간의 변수 mapping.
I/O variable I/O 변수	입출력 디바이스에 할당된 변수. I/O 변수는 반드시 어느 곳의 I/O 보드에 접속되어야만 합니다.
Identifier 식별자	프로그램에서의 변수와 정수를 나타내는 유니크한 워드
IL 명령 리스트	Instruction 리스트 언어
Initial situation 초기 상태	SFC 의 초기 스텝상태. 프로그램이 스타트하기 전의 상태.
Initial step	SFC 프로그램 스텝 내의 프로그램이 스타트한때 처음으로 액티브되는 스텝.
Input 입력	변수의 속성. 입력 디바이스에 해당된다.
Instruction	I L 프로그램의 기본명령. 1 행으로 적힘.
Instruction List	낮은 레벨의 언어. 명령리스트.
Integer 정수	아나로그 변수 클래스. 부호 첨가한 3 2 bit 포맷
Internal 내부	변수의 속성. 입출력 디바이스에 할당되지 않는 내부변수.

Jump to a step	SFC 에서 지정된 스텝으로 점프를 나타내는 화살표 (↓) 심벌. 점프 전에 reference 번호가 붙는다.
Keyword	언어의 예약어
Label (IL)	I L instruction 행의 처음에 부여 instruction 행을 나타내는 식별자. 점프 (J M P) operation 에도 사용됩니다.
Ladder Diagram	논리식의 그래픽컬한 표현언어. 컨택트와 코일을 접속한 다이어그램.
LD Ladder 그림언어	Ladder 그림언어
Level 1 of the SFC SFC 레벨 1	SFC 프로그램 메인 기술. 레벨 1 에서는 스텝과 transition 접속과 각각의 커맨드를 나타냅니다.
Level 2 of the SFC SFC 레벨 2	SFC 프로그램 상세히 기술. SFC 스텝의 액션 기술과 transition 의 boolean 조건이 기술됩니다.
라이브러리	하드와 소프트 리소스의 집합. 프로그램 에디터에서 액세스 가능합니다.
Local	변수와 정의 워드의 범위. 프로젝트의 어떤 프로그램에서만 사용 가능합니다.
Locked I/O	입출력 변수가 대응하는 I/O 디바이스에서 논리적으로 나뉘져 있는 상태. 디버거에서 록 커맨드를 보낼 수 있습니다.
Macro step	메인 SFC 차트와 별도로 기술되는 스텝과 transition 그룹. 메인 SFC 에서 호출됩니다.
Matrix	그래픽컬 프로그램 에디터에 따른 편집 area 를 셀에 구분되어 있는 상태.
Message 가변장 문자형	변수의 타입. 가변장 문자열을 가진 변수. 문자열 리스트라고도 합니다.

Modbus	마스터 slave 통신 protocol 의 하나. ISaGRAF 는 M o d b u s 의 slave 가 됩니다. 이 경우 마스터는 ISaGRAF 이외의 외부시스템입니다.
Modifier (IL)	I L operation 마지막에 놓인 1 문자 캐릭터 키 워드. 이 문자로 operation 의미를 수식합니다.
Network address	각 변수에 부친 옵션적인 1 6 진수 어드레스. 이 어드레스는 Target 시스템이 M o d b u s protocol 외부시스템 (마스터) 과 접속되는 경우에 유니크한 변수로 다루려는 것입니다.
Non-stored action	SFC 액션. 대응해 있는 스텝이 액티브 상태일 때 매회 Target 사이클시에 실행되는 액션.
OEM key code (I/O board)	ISaGRAF 라이브러리에 등록되는 I/O 보드를 인식하는 코드. 16 진수 16bit 에서 표현. 이 코드로 보드의 서브 라이브러리를 인식합니다.
OEM 파라미터 (I/O board)	I/O 보드의 파라미터 . I/O 보드의 드라이브에 필요한 정보로 I/O 접속 시에 설정할 필요가 있습니다. 정수와 변수 등이 배당됩니다.
Operation (IL) 명령(IL)	IL 언어의 본 instruction. 통상, 하나의 Operation 에는 하나의 Operand 가 관계하고 있습니다.
Operand (IL)	IL instruction 에서 처리되는 변수와 정수표현.
Output 출력	변수의 속성. Target 시스템의 출력 변수에 할당됩니다.
파라미터 (C function)	C 언어 function 에 주어진 방향을 가진 값. 각 파라미터 는 타입이 있습니다.
파라미터 (I/O board)	I/O 보드의 사용자정의 또는 정수파라미터 . 사용자정의 파라미터 는 I/O 접속시에 정의해야만 합니다.
Parent program	
Power rail 모선	ladder 프로그램에서의 좌우 모선.

Program	프로젝트내의 프로그램 기본 Unit. 프로그램은 하나의 언어로 기술되는 프로젝트 계층구조 안에 위치하게 된다.
Project	ISaGRAF 어플리케이션의 하나를 나타내고, 필요한 모든 정보 (프로그램, 변수, Target 코드 등) 를 포함하고 있습니다.
Pulse action	SFC 액션. 대응하는 SFC 스텝이 액티브 상태가 된 때만 1 회 실행되는 액션.
Quick LD	버전 3 . 2 에서 추가된 Father ladder 그림전용 에디터. function 키와 커서 키만 ladder 를 작성할 수 있습니다.
Range 범위	ISaGRAF 에서의 정의 종료의 범위에는 공통, 글로벌, local 의 세가지가 있습니다.
Real 실수형	아나로그 변수 클래스. 유동 소수점이 표현되는 3 2 bit 포맷
Real board	Target 시스템에서 I/O 디바이스에 물리적으로 접속된 I/O 보드.
Real time mode	Target 시스템을 통상 실행모드. Target 의 사이클은 사이클 타임마다 작동됩니다.
Reference number (SFC)	SFC 스텝과 트랜지션을 식별하는 번호. 1 에서 6 5 5 3 5 까지 취급합니다.
Register (IL)	IL operation 중의 IL Register 내의 현재 결과
Return value of a sub-program 서버프로그램의 되돌리기 값	서버프로그램 실행 마지막에 되돌아 오는 값. 되돌리기 값은 Father 프로그램에서 사용됩니다.
Run time error	ISaGRAFTarget 시스템이 실행중에 검출되는 어플리케이션 에러.
Section	프로그램 그룹. 같은 룰로 실행되는 프로그램군. IsaGRAF 에는 다섯가지 섹션 (Begin, sequence, End, function, F B) 이 있습니다.

Separator	텍스트 언어 내에서 사용되는 끊기 캐릭터로 operation 에 필요한 Separator (액티브 Separator) 와 불필요한 Separator (인액티브 Separator) 가 있습니다.
Sequential Function Chart	스텝과 스텝사이를 link 하는 transition 으로 프로세스를 기술한 그래픽 다이어그램. 스텝엔 액션이 배당되고, transition 에는 boolean 조건이 할당됩니다.
Sequential section	프로젝트를 구성하는 프로그램 그룹. SFC 언어의 다이내믹 룰에 따른 프로그램.
SFC	Sequential function 차트언어
ST 구조화 텍스트	구조화 텍스트 언어
Statement	기본 ST operation
Step	SFC 기본 Component. 스텝은 대상 프로세스의 어떤 상태를 나타냅니다. 정방향으로 나타냅니다. reference 번호에서 참조됩니다. 스텝의 액티브 상태에 따라 대응하는 액션 실행이 컨트롤 됩니다.
String 문자열	메시지변수를 구성하는 문자열
Structured Text 구조화 텍스트	구조화 텍스트언어. 변수 Assignments, If/Then/Else 와 function call 등을 지원하고 있습니다.
Sub-program	SFC 이외에 기술된 프로그램으로 다른 Father 프로그램에서 호출됩니다.
Target	ISaGRAF workbench 에서 만들어진 어플리케이션의 실행환경. ISaGRAF 제어 Kernel 소프트가 workbench 에서의 어플리케이션 코드 (중간 코드) 를 해석, 실행하고 있습니다. 컴파일러형 Target 도 있습니다.

Target cycle	사이클 타임내에서 실행하는 Target 처리 사이클
Technical note 기술메모	ISaGRAF 라이브러리각 element (C언어 function, F B, 변환 function ,I/O 보드) 에 대한 사용자가이드. Element 설계자가 기술합니다.
Test(FC) FC 테스트	Flow 차트의 조건판단을 의미하는 심벌. Flow 는 YES/NO 의 출력에서 접속됩니다.
Timer 타입형	변수의 타입. 타임을 갖고 있고, 런타임시에 자동으로 갱신됩니다.
Token (SFC)	SFC 프로그램의 액티브 스텝을 나타내는 심벌. ● 으로 표현됩니다.
Toolbox	그래피컬 에디터 창에서 사용되는 각종 심벌의 선택을 아이콘화 해서 정리한 것.
Top level program	프로젝트를 구성하는 프로그램에서 프로그램 계층 (각각의 섹션에 존재) 의 최상위 레벨. top 레벨 프로그램은 시스템에 따라 작동합니다.
Transition	SFC 기본 component. SFC 스텝간의 이동조건을 나타냅니다. reference 번호에서 참조됩니다. 각 transition 에는 boolean 조건이 할당됩니다.
Type	변수의 타입. 4 개의 기본타입 (boolean 형, 정수 / 실수형, 타입형, 가변장 문자열형) 이 있습니다.
Validity of a transition transition 평가	SFC transition 의 속성. 직전의 스텝이 액티브시 항상 transition 는 평가됩니다.
Variable	프로젝트내의 프로그램으로 다루는 기본 데이터.
Virtual board	Target 상에서 I/O 디바이스와 논리적으로 접속되지 않는 I/O 보드. (동작 확인시에 real 보드에서 virtual 보드로 바꿀 수 있습니다.)

E. 색인

기호

-	B-293
\$ sequence	B-225
\$ sequence	B-225
%	A-115, B-226
&	B-290
) operation (IL)	B-285
*	B-294
/	B-294
:=	A-173
:= (ST assignment)	B-270
+	B-292
<	B-298
<=	B-299
<>	B-302
=	B-301
>	B-299
>=	B-300
>=1	B-291

숫자

1	B-292
1 gain	B-289
1 0 진수	B-224
1 6 진수 표시	C-413
1 사이클 실행	A-139

A

ABS	B-329
Absolute value	B-329
Access right	A-202
Acknowledge	C-410
ACOS	B-333
Action	A-59, A-64, B-235, B-240, B-246, B-247, D-476
Activity duration	B-231, B-276
Activity of a step	B-230, B-231, B-243, B-276, D-476
Addition	B-292
Alias	A-75,D-476

ANA	B-303
Analog	A-103, A-167, B-223, B-224, B-227, C-417, C-418, D-476
AND	B-290
AND_MASK	B-295
AnyTarget	A-129
Application size	C-412
Application	C-465
Arc cosine	B-333
Arc tangent	B-334
Archive	A-28, A-192
Archive drive	A-193
Archive File	A-194
ARCREATE	B-355
Array creation	B-355
Array reading	B-356
Array writing	B-357
ARREAD	B-356
ARWRITE	B-357
ASCII	B-346
Assignment	B-289
Assignment (in ST,=)	B-270
ATAN	B-334
Attribute	D-476
AVERAGE	B-322
a 접점	B-257

B

Background picture	A-153
백업	A-28, A-192, A-194
Bargraph	A-154
Base	B-223
Baud rate	A-44
Begin	B-245
Begin section	A-30, B-218, D-476
Begin 섹션	B-218
Beginning step	A-49, A-50, B-235, D-476
Binary selector	B-346
BinaryFile	A-128
Bit field	A-155
Bitmap	A-153
BLINK	B-326
Board	A-112, A-113
Board 파라미터	A-114
Board type	A-113
Body of a macro step	B-235
BOO	B-302
Boolean	A-102, A-167, B-227, D-476

Boolean action	A-55, B-236, D-476
Breakpoint	A-137, A-140, A-142, D-476
BY	B-274
b 접점	B-257

C

C code	A-125
C 컴파일러	C-444
C function	A-190, A-194, C-422, D-476
C function block	A-190, A-194
C language	A-208, C-419, C-420, C-425, C-433, C-435, C-444, D-476
C source code	A-183, C-420, C-435, C-445, D-476
C source header	A-183, C-419, C-425, C-433, C-445, D-477
CAL operator (IL)	B-286
CASE	B-272
Cat	B-307
Cell	D-477
Channel	A-114, A-116
Channel comment	A-114
CHAR	B-347
Child	A-31
Child SFC program	B-219, D-477
Clear text	A-89
Clear variable	A-101
Clearing a transition	A-143, B-242, D-477
Clipboard	A-89, A-101
CMP	B-319
Code	A-122
Code Generation	A-38, D-477
Code generator	A-121
Coil	A-69, A-80, B-257, D-477
Coil type	A-73
Comment	B-248, B-266, B-281, D-477
Comment (SFC)	B-230, B-231, D-477
Common	A-98, A-193, D-477
Communication	A-43, A-136, A-140, A-161, A-208, C-410, C-412
Comparison	B-319
Compile	A-38
컴파일러	A-121
컴파일러 message	A-126
컴파일러 option	A-40, A-161
Compression	A-193
Condition	B-246
Condition (for a transition)	B-240, D-477
Connection	A-81, A-82, A-83

Connector	A-62, B-248, D-477
Constant expression	B-223, D-477
Contact	A-68, A-79, B-257, D-477
Contact negative edge	B-258
Contact Positive edge	B-258
Contact type	A-73
Convergence	A-47, A-50, B-233
Conversion	A-119, D-477
Conversion ASCII -> character	B-347
Conversion character -> ASCII	B-346
Conversion function	A-191, A-194, C-417, D-477
Conversion table	A-118, A-120, D-477
Convert to boolean	B-302
Convert to integer	B-303
Convert to message	B-306
Convert to real	B-304
Convert to timer	B-305
Copy FBD	A-83
Copy FC	A-64
Copy LD	A-74
Copy 라이브러리 element	A-180
Copy program	A-36
Copy SFC	A-51
Copy text	A-89
Copy variable	A-101
Corner	A-81
COS	B-335
Cosine	B-335
Counter down	B-315
Counter up	B-314
Counter up/down	B-316
Cross references	A-40, A-132, D-477
CTD	B-315
CTU	B-314
CTUD	B-316
Current result (IL)	B-281, B-282, D-478
Curve	A-153, A-154, A-158
Cut FBD	A-83
Cut FC	A-64
Cut LD	A-74
Cut SFC	A-51
Cut text	A-89
Cut variable	A-101
Cycle	A-174, B-222
Cycle profiler	A-170
Cycle timing	A-38, A-39, A-139, A-141, A-170, B-307, C-418, C-422, C-430, D-478
Cycle to cycle	A-38
Cycle to cycle mode	A-40, A-137, A-139, D-478
Cyclic	B-218, D-478
C 컴파일러예	C-449

C 소스 코드	D-476
C 소스 헤드	D-477
C function	A-194
C functionBlock	A-190,A-194,C-429
C function	A-190
C 언어	A-208, C-419, C-420, C-425, C-433, C-435,D-476
C 언어 F B interface	C-432, C-433
C 언어 F B call	C-440
C 언어 F B 소스 코드	C-435
C 언어 F B 제한	C-442
C 언어 컴파일러	C-416, C-444
C 언어 소스 코드	A-125,A-183,C-420, C-445
C 언어 소스 헤드	A-183, C-419, C-425, C-433,C-445
C 언어 function interface	C-425
C 언어 function 소스 코드	C-426
C 언어 function Block call	C-431
C 언어 프로그램	C-416
C 언어 function	C-422,D-476

D

DAY_TIME	B-354
DDE	A-148,C-407
DDE (NT target)	C-412
D D E 어드레스 loop	C-408,C-410
D D E request	C-408
Debug	A-42
Debug workspace	A-42
디버거	A-136, A-164, A-166
Decimal	B-224
Decision	A-59, A-63, A-64, B-246, D-478
Declaration	A-34, A-97
Defined word	A-34, A-98, A-102, A-104, B-229, D-478
Delayed operation (IL)	B-282, D-478
DELETE	B-348
Delete board	A-113
Delete FBD	A-83
Delete FC	A-64
Delete LD	A-74

Delete 라이브러리 element	A-180
Delete program	A-37
Delete SFC	A-51
Deleted style	A-87
DERIVATE	B-325
Descriptor	A-23, A-41
Diagnosis	A-164
Diary	A-24, A-34, D-478
Dictionary	A-34, A-92, A-97, A-132, A-134, C-418, C-430, D-478
Differentiation	B-325
Direct contact	B-257
Directly represented variable	A-115, B-226, D-478
Disabled transition	B-242
Disk	A-13
Dissociate	A-155
Divergence	A-47, A-50, B-232
Division	B-294
DO	B-273, B-274
Document	A-25, A-41, A-195
DOS Target	C-373
Download	A-138, A-166
Dump	A-149

E

Edge	D-478
Edge contact	B-258
Edit project	A-24
Edit variable	A-101
ELSE	B-272
ELSIF	B-272
Embedded source code	A-161
EN	A-70
Enabled transition	B-242
End	A-177, B-245
End section	A-30, B-218, D-478
End 섹션	B-218, D-478
END_CASE	B-272
END_FOR	B-274
END_IF	B-272
END_REPEAT	B-274
END_WHILE	B-273
Ending step	A-49, A-50, B-235, D-478
End of file detection	B-360
Enhanced operator (AND,&)	B-290
ENO	A-70
ENO 출력	A-70
EN 입력	A-70

EQ	B-301
Error	A-126
Ethernet	A-44
Execute one cycle	A-139
Execution order	A-84
EXIT	B-275
Exponent	B-329
Export	A-108
Export function	A-37
Export function block	A-37
Expression	D-478
EXPT	B-329

F

F_CLOSE	B-359
F_EOF	B-360
F_OPEN	B-358
F_TRIG	B-313
F_WOPEN	B-358
FA_READ	B-361
FA_WRITE	B-362
FALSE	A-102, A-142
Father SFC program	B-243, D-478
FBD	A-32, A-78, A-98, B-218, B-221, B-251, C-423, C-431, D-478
FBD comment	A-82
FBD editor	A-78, A-92
FBD/LD 에디터	A-78
FBD 에디터	A-92
FBD Cut	A-83
FBD Copy	A-83
FBD 커맨드	A-82
FBD 심벌의 선택	A-81
FBD 심벌의 배치	A-81
FBD 삭제	A-83
FBD 실행순	A-84
FBD 붙이기	A-83
FC	A-32, A-59, B-218, B-221, D-478
FC comment	A-62
FC connector	A-62
FC editor	A-59
FC I/O 조작 Block	B-247
FC link	A-62
FC sub-program	A-31, B-247
FC 에디터	A-59

FC 오브젝트의 이동	A-62
FC 오브젝트의 삽입	A-61
FC Cut	A-64
FC copy	A-64
FC 커맨드	A-62, B-248
FC 서버프로그램	A-31, B-247
FC Link	A-62
FC 삭제	A-64
FC 사이즈 변경	A-63
FC 붙이기	A-64
File close	B-359
File open	B-358
File read	B-361, B-364
File write	B-362, B-366
Find	A-52, A-64, A-74, A-84
FIND	B-348
Flow	A-62, B-245, B-248, B-249
Flow Chart	A-59, D-479
Flow Chart editor	A-59
FM_READ	B-364
FM_WRITE	B-366
Font	A-199
FOR	B-274
From	A-129
Function	A-34, A-37, A-188, A-194
Function block	A-34, A-37, A-69, A-80, A-84, A-98, A-102, A-188, A-194, B-221, B-251, B-268, C-429, D-479
Function block call in IL	B-286
Function Block instance	A-105, C-430
Function Block section	A-30, B-218
Function section	A-30, B-218
Functional Block Diagram	B-251, D-479

G

gain 1	B-289
Gallery	A-56
GE	B-300
Get version number	A-138
GFREEZE	B-219, B-243, B-278
GKILL	B-219, B-243, B-278
Global	A-97, A-98, A-132, B-222, B-225, D-479
Go to	A-52, A-64, A-90
Goto	A-175
Graphic	A-153, A-158
Greater or equal	B-300
Greater than	B-299

Grid	A-71
Group	A-155
GRST	B-219, B-243, B-279
GSTART	B-219, B-243, B-277
GSTATUS	B-243, B-279
GT	B-299

H

Hexadecimal values	C-413
hidden variable	C-435
Hierarchy	A-29, A-35, B-218, B-243, D-479
History	A-24, A-41
HYSTER	B-322
Hysteresis	B-322, B-323

I

I/O	A-40, A-113, A-114, A-134
I/O board	A-112, A-166, A-185, A-194, A-205, D-479
I/O channel	A-112, A-166, D-479
I/O channel OPERATE	B-309
I/O complex equipment	A-184
I/O configuration	A-24, A-183, A-194
I/O connection	A-112, D-479
I/O equipment	A-194
I/O specific	B-246, B-247
I/O variable	A-112, A-140, A-166, A-205, D-479
I/O wiring	A-40
I/O Simulation	C-410, C-412
I/O 채널	A-112, A-166, D-479
I/O 채널 커맨드	A-114
I/O 보드	A-112, A-113, A-166, A-185, A-194, A-205, D-479
I/O 보드의 이동	A-113
I/O 구성	A-24, A-183, A-194
I/O 접속	A-40, A-112, D-479
I/O 장치기	A-184, A-194
I/O Lock	A-140
I/O 변수	A-112, A-140, A-166, A-205, C-417, C-418, D-479
Icon	A-15, A-154
Identifier	D-479
IEC 언어	A-188
IF	A-176, B-248, B-272

IL	A-32, A-89, A-151, B-218, B-221, B-240, B-241, B-281, D-479
IL editor	A-92
IL 에디터	A-92
IL 레지스터	B-281
Import	A-108
Initial situation	B-231, B-242, D-479
Initial step	B-231, B-242, D-479
Input	A-112, A-134, A-170, B-222, D-479
INSERT	B-349
Insert coil	A-72
Insert contact	A-72
Insert FBD	A-84
Insert FBD element	A-81
Insert FC element	A-61
Insert rung	A-73
Insert slot	A-113
Insert variable	A-54
설치	A-12
Instance	A-102
Instruction	B-281, D-479
Instruction List	B-281, D-480
Integer	A-102, A-103, B-223, D-480
INTEGRAL	B-324
Interface	A-34
Internal	D-480
Inverted coil	B-259
Inverted contact	B-257
IO variable	C-417, C-418
Is equal	B-301
Is not equal	B-302
ISaGRAF directories	A-208
ISaGRAF.INI (NT target)	C-401

J

JMP	B-284
Jump	A-69, A-80, B-252, B-263
Jump to a step	A-48, B-232, D-480

K

Keyword	B-225, B-282, D-480
---------	---------------------

L

Label	A-80, A-175, B-252, B-263
Label (IL)	B-281, D-480

Ladder Diagram	B-255, D-480
Language	A-32, A-188, B-221
LD	A-32, A-56, A-66, A-68, A-78, B-218, B-221, B-255, B-283, D-480
LD Cut	A-74
LD Copy	A-74
LD 삭제	A-74
LD 붙이기	A-74
LE	B-299
LEFT	B-350
Less or equal	B-299
Less than	B-298
Level 1 of the SFC	A-47, B-230, B-231, D-480
Level 2	A-52, A-64
Level 2 of the SFC	A-52, B-235, D-480
Level of protection	A-201
라이브러리	A-28, A-37, A-113, A-114, A-132, A-167, A-178, A-193, D-480
라이브러리 manager	A-178, C-416, C-418, C-423, C-431
LIM_ALARM	B-323
LIMIT	B-341
Link	A-43, A-81, A-82, A-83, A-161, B-245, B-248, B-249
Link (FBD)	B-252
Link (LD)	B-255
Link (SFC)	B-232
List of variables	A-149, A-151, A-155
Local	A-97, A-98, A-132, B-225, D-480
Locked I/O	A-140, A-142, A-205, D-480
LOG	B-331
Logarithm	B-331
LT	B-298
M	
Macro step	A-49, A-50, B-234, D-480
Make	A-38
Make application	A-121
Mask on integer bits (and)	B-295
Mask on integer bits (not)	B-297
Mask on integer bits (or)	B-296
Mask on integer bits (xor)	B-297
Matrix	D-480
Matrix Cell	A-50
MAX	B-340
Maximum	B-340
Memory	A-12
Message	A-102, A-104, A-149, A-167, B-224, D-480
Message concatenation	B-307
Message length	B-352
Metafile	A-153

MID	B-351
MIN	B-340
Minimum	B-340
MLEN	B-352
MOD	B-342
MODBUS	A-107, D-481
MODBUS 1bit 적어넣기	C-454
MODBUS 1 워드 적어넣기	C-454
MODBUS N bit 읽어내기	C-453
MODBUS N 워드 적어넣기	C-454
MODBUS N 워드 읽어내기	C-453
MODBUS Slave	C-451
MODBUS 데이터 적어넣기	C-455
MODBUS 데이터 읽어내기	C-456
MODBUS 파일 전송	C-455
MODBUS function 코드	C-452
MODBUS Protocol	C-451
MODBUS 마스터	C-451
Modification tracking	A-86
Modified style	A-87
Modifier (IL)	B-281, B-282, D-481
Modulo	B-342
Move board	A-113
Move FBD	A-82
Move FC	A-62
Move LD	A-82
Move program	A-35
Move project	A-24
Move SFC	A-52
Move SpotLight	A-155
MSG	B-306
Multiplexer with 4 entries	B-342
Multiplexer with 8 entries	B-343
Multiplication	B-294
MUX4	B-342
MUX8	B-343

N

N qualifier	A-54
NE	B-302
NEG	B-289
Negated link	A-81, A-82
Negation	B-289
Negation (FBD)	B-253

Negative contact	B-258
Network address	A-99, A-102, A-107, D-481
New function	A-32
New function block	A-32
New 라이브러리 element	A-179
New program	A-32
New project	A-24
New rung	A-71
New variable	A-101
Non stored	A-54
Non-stored action	B-238, D-481
Normal style	A-87
NOT	A-81, A-82
NOT_MASK	B-297
NT (protection key)	A-16
NT Target	C-400
NT Target Help 메뉴	C-411
NT Target Message 메뉴	C-410
NT Target Option 메뉴	C-410
NT Target Start/Stop 버튼	C-411
NT Target View/Information	C-411
NT Target View 메뉴	C-410
NT Target 아이콘	C-411
NT Target 메인 창	C-409

O

ODD	B-344
OEM key code	A-185, A-186, D-481
OEM 파라미터 (I/O board)	A-186, D-481
O E M 키 코드	A-185, A-186
O E M 파라미터 (I/O 보드)	A-186
OF	B-272
Off-delay timing	B-318
On Line	A-42, A-136
On line modification	A-139, A-144
On-delay timing	B-317
Open program	A-34, A-134
Operand (IL)	B-281, B-282, D-481
OPERATE I/O channel	B-309
Operation (IL)	B-282, D-481
Operator (IL)	B-281
Optimizer	A-122, A-124
Options (Code generator)	A-122

Options (디버거)	A-140
Options (Simulator)	A-169
OR	A-79, B-291
OR_MASK	B-296
OR 접속	A-79
OS-9Target	C-378
Other 라이브러리	A-178
Other program	A-93
Output	A-112, A-134, B-222, D-481

P

P qualifier	A-54
P0 qualifier	A-54
P0 어플리케이션 qualifier	A-54
P1 qualifier	A-54
P1 어플리케이션 qualifier	A-54
Page	A-198
파라미터	A-34
파라미터 (C function)	A-181, A-190, D-481
파라미터 (function block)	A-181, A-190
파라미터 (I/O board)	A-181, A-186, D-481
Parent program	B-219, D-481
Parenthesis	B-267, B-282
Parity	A-44
Parity test odd/even	B-344
Password	A-26, A-116, A-180, A-201
Paste FBD	A-83
Paste FC	A-64
Paste LD	A-74
Paste SFC	A-51
Paste text	A-89
Paste variable	A-101
Point	A-119, A-120
Positive coil	B-261
Positive contact	B-258
POW	B-330
Power calculation	B-330
Power rail	A-68, A-69, A-79, B-255, D-481
Print	A-25, A-41, A-173, A-195, A-198
Print (history)	A-195
Print program	A-94
Print variables	A-101
PrintTime	A-174
Priority	C-410
Program	A-29, A-92, A-170, B-218, D-481

프로그램 comment	A-33
프로그램 management	A-29
프로그램 syntax	A-92, A-122
프로그램에러	C-466
Project	A-193, A-195, B-218, D-482
Project descriptor	A-24, A-41
Project document	A-41
Project group	A-27
Project list	A-23
Project management	A-23
Protection	A-116
Protection key	A-16
Protection level	A-202
Pulse	A-54
Pulse action	B-237, D-482
Pulse timing	B-319

Q

Quick declaration	A-105
Quick LD	A-56, A-66, A-68, D-482
Quick LD editor	A-92
Quick LD 에디터	A-92

R

R (reset)	B-284
R_TRIG	B-312
RAND	B-345
Random number	B-345
Range	A-97, A-98, A-99, D-482
Real	A-102, A-103, B-224, B-304, D-482
Real board	A-113, A-166, A-205, D-482
Real time	A-38
Real time mode	A-40, A-137, A-139, D-482
Re-calculate	A-133
REDGE	B-269
Reference number	B-230, B-231, B-235, D-482
Register (IL)	B-281, D-482
Rename 라이브러리 element	A-180
Rename program	A-35
Renumber	A-52, A-64
REPEAT	B-248, B-274
Replace	A-52, A-64, A-74, A-84, B-352
Replace text	A-90
Reset coil	B-261
Resize FC	A-63

Resize SpotLight	A-155
Resource	A-40,A-126
Resource definition File	A-127
Restore	A-28, A-192, A-194
RET	B-285
Retain	A-40, C-459
RETURN	A-70, A-81,B-252, B-262, B-271
Return value	D-482
RIGHT	B-353
Right mouse button	A-205
ROL	B-337
ROR	B-337
Rotation left	B-337
Rotation right	B-337
RS	B-311
RS232C	A-44
Run-time	A-38
Run-time error	A-38,A-39, A-136, A-141, B-308, D-482
Rung	A-68, A-69, A-74, A-82
Rung comment	A-71, A-75
Rung label	A-71

S

S (set)	B-283
Save list	A-149
Scientific	B-224
Script	A-171, A-172
Search text	A-90
Search variable	A-100
Section	A-29, B-218, D-482
Section selection	A-33
SEL	B-346
Select FBD element	A-81
Select FC element	A-61
Select SpotLight	A-155
SEMA	B-313
SEMAPHORE	B-313
Separator	B-266, D-482
Separator (project)	A-23
Sequential	A-46, B-218, B-230
Sequential Function Chart	B-230, D-482
Sequential section	A-30, B-218, D-483
Serial link	A-44
Set coil	B-260
SFC	A-32, A-46, A-124, A-142, A-198, B-218, B-221, B-230, C-423, D-483
SFC action	A-55, B-238
SFC child	A-31
SFC editor	A-92
SFC evolution rules	B-242

SFC gallery	A-56
SFC action	A-55, B-238
SFC editor	A-92
SFC gallery	A-56
SFC 심벌의 이동	A-52
SFC 다이나믹 톨	B-242
SFC token	B-230
SFC 레벨 1	A-47, B-230, B-231, D-480
SFC 레벨 2	A-47, A-52, B-235, D-480
SFC Father 프로그램	B-243
S h e l l 프로젝트	C-387
Shift left	B-338
Shift right	B-339
SHL	B-338
SHR	B-339
SIG_GEN	B-326
Signal generator	B-326
Simulate I/O	C-404, C-410, C-412
Simulator	A-42, A-122, A-136, A-166, A-170, A-171, C-418, C-422
SIN	B-335
Sine	B-335
Slave	C-410
Slave number	A-43, C-412
SlavesLink	C-394
Slot	A-113, A-116
Sort	A-101
Spawn	C-390, C-393
SpotLight	A-153
Spy	A-149, A-151, A-153
Spy variable	A-149
SQRT	B-331
Square root	B-331
SR	B-310
SSR 글로벌 변수	C-397
ST	A-32, A-56, A-89, A-151, B-218, B-221, B-266, B-283, C-423, C-431, D-483
ST editor	A-92
Stack of integer analogs	B-320
STACKINT	B-320
Start application	A-138, A-166
Starting mode	A-40
Statement	B-266, D-483
Step	A-52, A-142, A-143, B-230, D-483
Stop application	A-138, A-166
String	A-104, B-224, D-483
String length	B-352
Structured Text	B-266, D-483

Style	A-86, A-156
ST 에디터	A-92
Sub-program	A-30, B-218, B-219, B-239, B-242, B-247, B-253, B-267, B-286, D-483
Sub-string delete	B-348
Sub-string extraction (left)	B-350
Sub-string extraction (middle)	B-351
Sub-string extraction (right)	B-353
Sub-string find	B-348
Sub-string insert	B-349
Sub-string replace	B-352
Subtraction	B-293
Symbols	A-210
SYSTEM	B-307
SYSTEM function	C-460
System 에러	C-465
SYSTEM function	C-460

T

Table of contents	A-196, A-197
TAN	B-336
Tangent	B-336
Target	A-122, A-129, D-483
Target cycle	D-483
Technical note	A-114, A-180, C-417, C-418, C-423, C-431, D-483
Test	A-59, A-63, A-64, A-136, B-246
Text display	A-154
Text editor	A-89
Text file	A-128
THEN	B-272
TIC code	A-122
T I C 코드	A-121, A-122
Time unit	B-224
Time-out	A-43
Timer	A-102, A-103, B-224, B-228, D-483
TMR	B-305
TO	A-129, B-274
TOF	B-318
Token (SFC)	B-230, D-483
TON	B-317
Toolbox	D-483
Tools menu	A-41
Top level program	A-29, A-30, D-484
Touch	A-38
TP	B-319
Transition	A-52, A-142, A-143, B-231, D-484
TRUE	A-102, A-142

TRUNC	B-332
Truncate decimal part	B-332
tst_main_ex	C-393
TSTART	B-276
TSTOP	B-277
Type	A-97, A-99, A-112, A-132, B-223, D-484

U

Ulong Data	A-127
UNTIL	B-274
Update	A-139, A-144
Upload	A-160
Upload (options)	A-162
Upload (prepare)	A-161

V

Validity of a board	A-113
Validity of a transition	D-484
Variable	A-34, A-54, A-73, A-81, A-84, A-90, A-97, A-132, A-134, A-141, B-218, B-225, B-251, D-484
Variable name	B-225, C-413
VarList	A-127
Verify	A-38
Virtual board	A-113, A-166, A-205, D-484
VxWorks Target	C-388

W

Wait	A-175
WHILE	B-248, B-273
width	A-77
Windows31/95-RTTarget	C-369
Workbench Limitation	A-214

X

XOR	B-292
XOR_MASK	B-297
Z	
Zoom	A-66, A-75, A-85